¡¡¡ Te enseñamos cómo crear los niveles del juego del siglo !!!

Nuevo

CÓDIGO JUEGO

Número 5 Octubre 94

875 pts
Canarias 900 pts

La primera revista del mundo que enseña a programar videojuegos

UN CURSO SENSACIONAL

DOOM



Aprende a programar la Sound Blaster

AHORA 16 páginas más

100 pts menos

8 424094 009756 00005

Ajatel S.L.

Nuevo CÓDIGO JUEGO
La primera revista del mundo que enseña a programar videojuegos

Editorial: AJATEL S.L.

Director
Ram n L pez
Subdirectores
Joan Esquerra
Miguel A. Lobato
Directora de arte
Maite L pez Vidal
Colaboradores:
JosØ Manuel GinØs
David Brioso
JosØ A. Brihuega Parodi
Eduard SÆnchez Palaz n
Alvaro Mateos
Raul Reales
Alfonso Salgado Cachafeiro
Mar a Jesœs Cano Rueda
Javier Laynez
Alberto Garc a
Leopoldo FernÆndez Prieto
Fotograf a
Manuel Gallego Palacios
Corresponsales USA
Luis Vazquez
Vicente Santos
Jefe de publicidad
Joan Esquerra
ASINFO: TelØfono/FAX: 956-813129
Subscripciones
Miguel L. Moreno
TelØfono/FAX: 956-840246
Filmaci n
Prieto Hermanos S.C.
Paseo de las Delicias, 15
TelØfono: 956-186105
Imprime
ALTAMIRA
Carretera de Barcelona, Km 11200
28022 Madrid, Tel. 91-747 33 33
Distribuye
DISPA A, S.L. S.C.
Avenida de Valdelaparra, 27
Edificio II, local 9
Alcobendas, Madrid
Tel. 91-6620602

ISSN: 1134-0975
Precio de este ejemplar 875 pts, Canarias 900 pts
Dep sito Legal:CA-286/94

C digo Juego
Apartado de Correos 114
11520 Rota, CÆdiz
TelØfono/FAX:956-840246
TelØfono/FAX:956-813129

C digo Juego no se hace necesariamente solidaria con las opiniones vertidas en los art culos firmados. EstÆ prohibido cualquier tipo de reproducci n de toda o parte de esta publicaci n por cualquiera que sea el medio,

SUMARIO

7º curso

Ensamblador PC (5ª parte)

En la 5ª entrega de este curso aprendemos más sobre la segmentación de la memoria.

5º curso

VGA (5ª parte)

Las interrupciones que nos brinda el BIOS de la VGA son de una gran utilidad para nuestros programas.

64 curso

Los debuggers (5ª parte)

Hemos decido separar los cursos de ensamblador de Atari y Amiga aunque tengan el mismo microprocesador. Así unos y otros tendréis un curso más específico.

44 curso

MIDI (5ª parte)

José Manuel, nuestro experto músico se adentra en el estudio de la tabla de implementación MIDI.

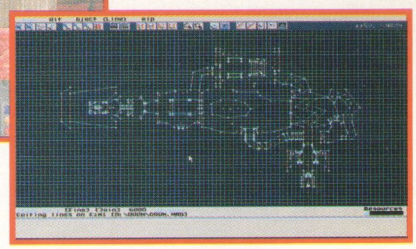
Epic Pinball: Un ju

Octubre 1994

Año 1, número 5

Editorial 4
La tarjeta VGA 5
Carta del Lector 14
Tintilla 15
Así se hace un juego PC 16
Historia de los juegos 20
Un gran juego 22
El libro del mes 25
El juego del mes 26
Curso lenguaje Pascal 28
Lenguaje C 36
Curso de Windows 33
Macroassembler 6.0 48
Curso de sonidos 50
El contestador 53
El buen SOFT 55
Así se hace un juego C 56
El MSDOS y los juegos 60
Los Debuggers 64
El hardware de Amiga 79





El superjuego DOOM a tus pies

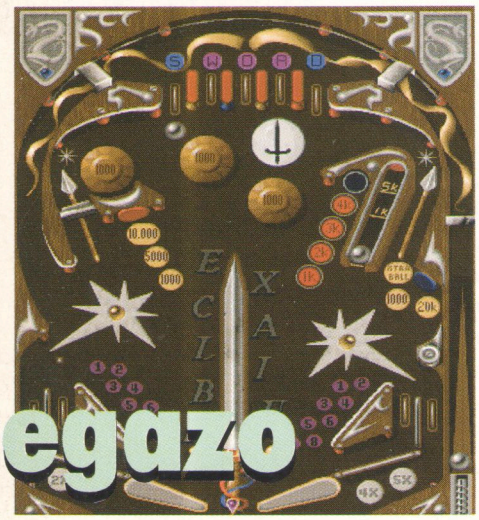
página 73

Gracias a nuestro colaborador, tenéis el primero de cuantos cursos existen sobre el manejo de los editores que permiten la creación de los niveles del juego del siglo; DOOM de ID Software. No te lo pierdas porque mes a mes vas a llevar el terror a tu ordenador. Ah , y en el número siguiente de CJ daremos las bases de un concurso sobre el DOOM que va a ser una bomba de relojería.

Programación de la Soundblaster

Pag. 40

Seguimos haciendo caso a las peticiones que recibimos en nuestra redacción sobre más y nuevas secciones. Esta vez le toca a un periférico archiconocido por todos los amantes del PC y los buenos juegos. De la compañía Creative Labs, nos llegó hace años esta tarjeta maravillosa, que dotó al PC de sus primeros sonidos serios. Aprende con nosotros a programarla y a sacarle todo el partido.



Editorial

Por:

Ramón López

De armas tomar

Hacer que un juego de ordenador pueda tener la misma calidad y aceptación que uno de consola es tarea más que difícil. En primer lugar, la consola tiene mayor difusión en el mercado lúdico y por tanto todo lo que se crea para ella se vende con más facilidad. En segundo lugar, y esto representa una ventaja aún mayor, el jugador no tiene que saber nada de bytes de memoria extendida, conflictos de interrupciones hardware, o RAM insuficiente, tan sólo insertar el cartucho y apretar el botón de start.



Cualquiera puede jugar con una, sin que para ello se tengan que poseer los conocimientos de Peter Norton, el famoso programador. Por esta razón hay sucesos que no dejan de sorprendernos. ¡Un juego de PC que ha sido capaz de romper barreras! Barreras tan fuertes como la de lograr un realismo singular en una máquina que, como todos sabemos, no está (o habría que decir "estaba") preparada para que los programas de diversión corran por sus venas. Pero lo que más sorprende de todo, es el hecho que no se intente sacar a este juego todo el partido que se le puede lograr mediante su comercialización masiva. Pero eso es algo que ya casi todos conocéis; DOOM se distribuye como shareware. Su versión completa puede ser encontrada en varios "sites" de la Internet, pero lo que es su distribución en España no ha llegado todavía. Una verdadera lástima, porque el juego completo lo merece, me refiero a un profundo y sonoro aplauso. Pero ahí no acaba lo bueno de este juego. La forma en que ha sido programado ha dado pie a otra fiebre. Nos explicamos. El juego de DOOM se divide en dos bloques principales. Una se daría en llamar la máquina, es decir, el programa o rutinas que hacen funcionar el programa realmente. La segunda sería el guión que decide en que posiciónes del mapa están los objetos (bidones de gasolina, escaleras y ascensores, paredes, armas, puertas, salidas de nivel) así como la localización y comportamiento de los enemigos. Toda esta información sobre el mapa de un determinado nivel se guarda en un fichero de extensión .WAD. Los estudiosos del tema han sabido descifrar esta información de forma que mediante la programación y creación de estos archivos que os he mencionado, podemos desarrollar de principio a fin nuestros propios niveles.

Sí, como lo oís. La imaginación se puede desatar a la hora de situar los diferentes elementos del juego de forma que los niveles más alucinantes salgan de vuestra cabecita. Pues bien, cuando lo tengas terminado y te guste como ha quedado, nos lo envías en un disquete a la dirección de la revista y al mejor de todos los recibidos le daremos un premio y lo incluiremos en un próximo disquete que haga aparición en Código Juego. Jamás, hasta ahora, ningún otro juego ha logrado reflejar de forma tan real, los sentimientos visuales y sonoros que un soldado puede sentir a la hora de huir de una gran fortaleza. Sentimientos que transmite rápidamente al intrépido jugador que se sienta ante sus mandos y que ve, con gran desaliento, que es presa de las garras de los alienígenas una y otra vez. Ahora sí, en el momento que nos hacemos con las riendas de juego, la sangre empieza a correr a manos de nuestra pistola, escopeta recortada, cañon, etc. Para los que no hayáis jugado con anterioridad, os recomendamos que os lo busquéis. ¡Ah!, y por supuesto no olvidéis leer el artículo que le dedicamos en Código Juego número 3.

En definitiva, que con nuestro curso váis a disfrutar de lo lindo. En cuanto al resto de novedades de este ejemplar que ahora tenéis entre vuestras manos, baste deciros que tiene 16 páginas más como fue firme promesa desde hace varios números atrás. Poco a poco seguiremos creciendo mientras nuestro presupuesto nos lo permita, siempre con la idea de aumentar las secciones para que encontréis en ella mucho más material de estudio y disfrute para un mes completo.

Otra grata sorpresa que habréis encontrado en la portada es la del precio. Hemos rebajado 100 pts, para que la revista sea más asequible y pueda llegar a todos. Así que ya véis. El titular de mi editorial del número 1, "Con una promesa en mente", se ha ido cumpliendo un poco más cada mes. Tenéis en entre estas páginas el listado completo de un capturador de pantallas llamado HUNTER (el cazador). Con él podréis estudiar las pantallas de otros juegos para escudriñar a fondo como se construyen los dibujos, los fondos, logotipos y los sprites. Además, de entre todas las líneas de código fuente del programa de nuestro nuevo colaborador, David Brioso, y también una sección esperada: La Sound Blaster a vuestros pies. Espero que este número os guste, porque hemos trabajado mucho para que así sea. Una abrazo de éste, vuestro director.

Ramón López.*****

Los ojos del ordenador:

Aprende a manejar la tarjeta



En artículos anteriores hemos hecho una introducción a las tarjetas de vídeo en general y a la VGA en particular. A partir de ahora entramos de lleno en la pura programación de este hardware para que nuestros juegos tengan todo el aspecto profesional que merecen.



Vamos a continuar con el estudio de las funciones que nos ofrece el BIOS que viene incluido (cosa que se conoce como firmware) dentro de la memoria ROM de la VGA.

El mes pasado adelantamos la utilidad de la primera de ellas, que nos permitía cambiar el modo de vídeo, ajustando así el formato de visualización de la pantalla a el tipo de aplicación que fuesemos a realizar.

En esta serie de artículos, los dos modos de vídeo que se estudiarán más a fondo son el 13h y el 13X, más conocido como modo X. Ello se debe a que la gran mayoría de los juegos actuales se ajustan a este formato, y por tanto, necesitarán más atención por nuestra parte. Sin embargo, los modos de texto también son útiles, sobre todo cuando al principio del juego queremos hacer un mensaje de advertencia o de entrada de datos por parte del usuario (por ejemplo: tipo de tarjeta de sonido, configuración de la misma, tipo de tarjeta de vídeo, etc.).

Ahora, y con la interrupción 10h, función 01h, vamos a aprender como establecer el tamaño y forma de nuestro cursor en modo texto. Como ya todos sabéis, el cursor indica cual será el punto de entrada de nuestras peticiones desde el teclado. Se trata, pues, de un puntero parpadeante (para que se vea mejor) que siempre, o casi siempre, nos aparecerá en nuestra pantalla.

Por defecto, el cursor aparece como una raya de dos líneas de alta en la parte inferior de la celdilla de carácter. Veamos esto más despacio, fijándonos en la figura que se adjunta en la página siguiente.

En el modo de texto 03h, que es en el que arranca el ordenador por omisión se pueden visualizar caracteres de hasta 14 líne-

as de alto. En los modos CGA, son de 8 líneas y en los MCGA y VGA se pueden alcanzar resoluciones de hasta 16 líneas.



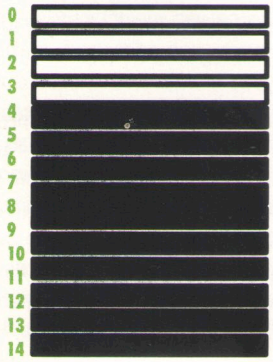
COMO CAMBIAR EL CURSOR

Para acceder a los servicios BIOS desde lenguaje ensamblador ya sabemos que deberemos llamar a la interrupción 10h con los datos que se detallan a continuación:

AH=01h; función 1 de la int 10h
AL=irrelevante
CH=línea de comienzo del cursor
CL=línea de final del cursor

ejemplo:

queremos un cursor con la siguiente forma en el modo de texto 03h:



...la sección de programa que se encargaría de ello sería algo así como:

```
cursor_gordo:   MOV AH,01H
                MOV CH,07H
                MOV CL,0EH ;14 dec
                INT 10H
```

Nota: los valores están siempre en un rango máximo de 0 a 15, es decir de 0000 binario a 1111 binario. Si el bit 5 de CH se pone a 1, esto indicará al BIOS que deseamos hacer desaparecer el cursor de la pantalla. Recuerda esto para cuando quieras lograr un efecto de cursor fantasma.

Te recomiendo que hagas un programa que sucesivamente vaya incrementando estos valores de forma que puedas obervar los diferentes aspectos del cursor.



Con este servicio del BIOS de la VGA se puede colocar el cursor en cualquier lugar de la pantalla mediante el uso de coordenadas de filas y columnas.
Aquí debemos hacer una aclaración. Dependiendo del tamaño de nuestra pantalla, 40x25, 80x25, etc, tanto mayor será el espacio que ocupan en la memoria de vídeo. Cada carácter ocupa dos bytes en memoria:

1) El byte ASCII
2) El byte de atributos



El primero le dice al hardware de la tarjeta VGA de que forma habrá de componer el carácter en pantalla en función de los valores de la "forma" que tiene guardados en memoria ROM. Por ello en modo texto podemos tener tan sólo 256 caracteres máximo.
El segundo informa sobre el color de fondo de ese carácter en concreto y del color del propio dibujo del caracter. Además informa sobre la carácteristica de parpadéo e intensidad.
Pero esto lo podéis ver mejor en la figura que aparece en la parte inferior de esta página.
Los bits 0,1 y 2 de este byte indican el color de el primer plano, o sea lo que sería el caracter en sí de acuerdo con esta tabla:

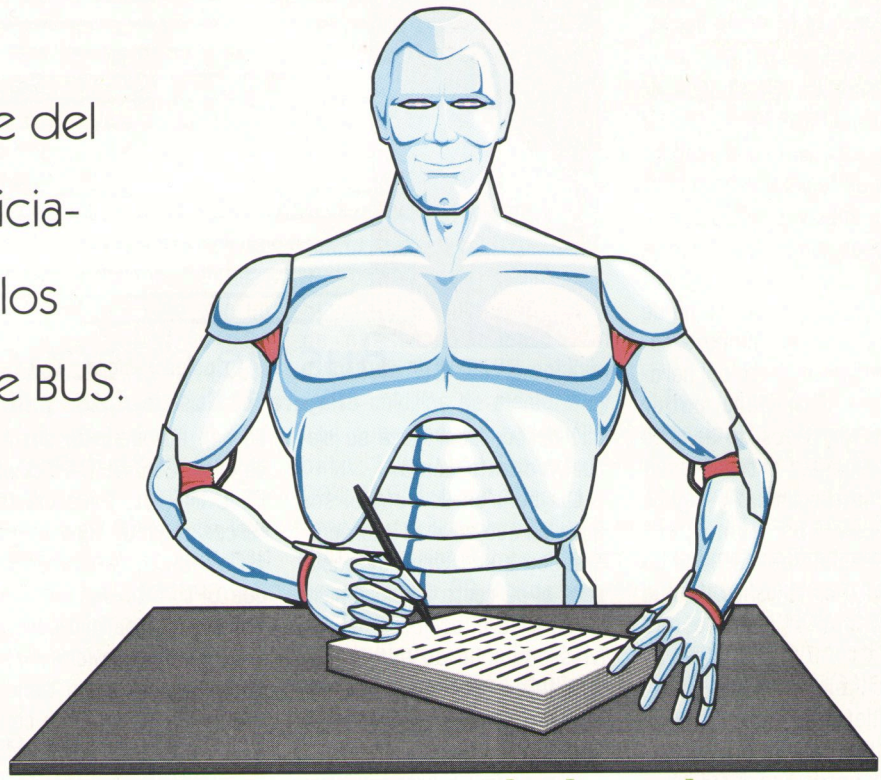
000	Negro
001	Azul
010	Verde
100	Rojo
011	Cian (Verde y azul)
101	Magenta
110	Marrón (Amarillo oscuro)
111	Gris claro

Si el bit 4, o de intensidad del primer plano está encendido los valores son:

1000	Gris oscuro
1001	Azul Claro
1010	Verde Claro
1100	Rojo Claro
1011	Cian Claro
1101	Magenta Claro
1110	Amarillo puro
1111	Blanco con Brillo

Estos son los dieciséis colores que podemos ver en los modos gráficos.
Además, si se enciende el bit de parpadéo, se alternará automáticamente entre los colores de fondo y primer plano..
El mes que viene continuaremos.****

Continuamos este mes con el estudio de la segmentación de memoria por parte del procesador, en iniciamos el análisis de los diferentes tipos de BUS.



Curso de lenguaje ensamblador para PC

Por:

Ramón López

5ª PARTE

Hemos entrado ya en materia importante. El correcto entendimiento de la forma en que el microprocesador segmenta la memoria en vital para las sucesivas explicaciones de este curso, así que hagamos un alto en el camino para ahondar en este tema.

El microprocesador Intel 8086, el cual ha servido de base para todos los que han surgido con posterioridad, está formado por un hardware complicadísimo que, aunque no estaría mal conocer aunque sólo sea por encima, no nos incumbe de momento. Tan sólo, es conveniente conocer varios aspectos del mismo aunque sólo sea de manera superficial. Uno de ellos es los tipos de BUS con los que cuenta, ya que éste está totalmente relacionado con la forma tan extraña en que la CPU trabaja con la memoria.

EL BUS SE PODRÍA REPRESENTAR COMO UN CONJUNTO DE LÍNEAS O CABLES POR LOS QUE FLUYE LA INFORMACIÓN

TIPOS DE BUS

Un bus, para todos aquellos que hasta ahora no os hayáis tropezado con esta palabreja, es un autobús. Sí, aunque os suene raro, la mejor forma de asociar éste con su verdadera función es la de utilizar el símil de un transporte público, sólo que no de personas, sino de datos.

El microprocesador se ha de conectar con el mundo exterior, es decir el resto del hardware del PC, para poder enviarle dos tipos de señales principalmente; datos (números e instrucciones) y signos de control, así como para poder recibir estas señales también. Por lo tanto, lo que circula por el bus (lo que transporta el autobús) es de ida y vuelta.

Los datos e instrucciones son conocidos de antes, instrucciones código máquina y datos que éstas utilizan y que se guardan por lo general en la memoria y de ésta última es de donde se "traen" hacia el microprocesador para su ejecución o tratamiento.

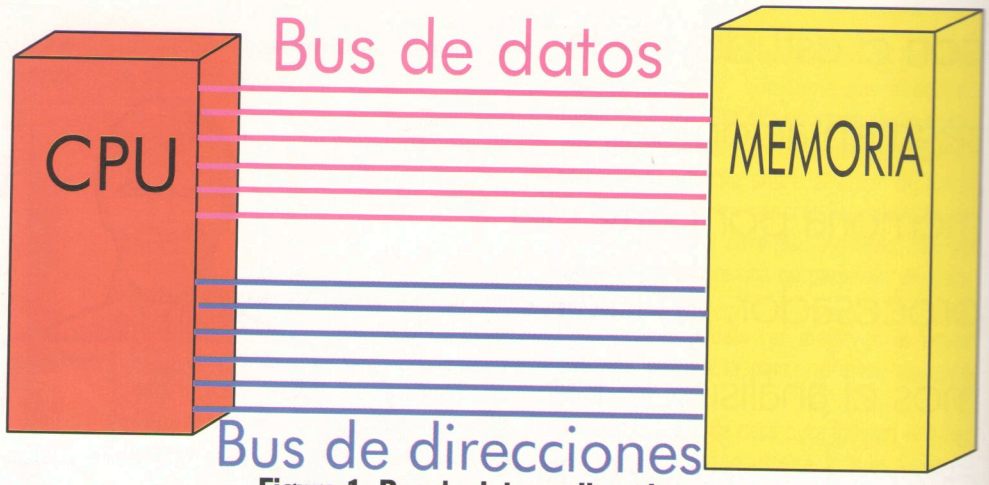


Figura 1, Bus de datos y direcciones

Las señales de control son aquellas que envía el microprocesador a sus chips de apoyo para el correcto funcionamiento de todo el conjunto del PC. Más adelante se estudiarán también, pero por ahora vamos a centrarnos en las primeras.

El microprocesador cuenta con un bus (que no es sino un conjunto de "cables" eléctricos) exclusivo para cada tarea. Según lo que hemos visto, ya se han autodefinido dos tipos de ellos; BUS DE DATOS y BUS DE CONTROL. Pero existe un tercer tipo muy importante.

Cuando la CPU (recordad que CPU= Unidad Central de Proceso, o sea microprocesador) necesita enviar un dato a la memoria primero tiene que decirle a los chips que controlan la ésta, a que dirección o posición se está haciendo referencia. Sabemos, que la memoria del PC es como una gran hilera de cajas etiquetadas con su número de orden. Cuando el procesador ha de ejecutar la instrucción siguiente:

MOV (1234H), AX

...tendrá que avisar a sus controladores de memoria que en la dirección 1234h, y no en otra, se va ha introducir el contenido que en ese momento tenga el registro acumulador AX. Pero como envía la CPU el dato-dirección 1234h. Para ello se implementó un bus exclusivo para el transporte de estas señales que, como no, se llamó BUS DE DIRECCIONES.

Cuando el microprocesador quiere introducir un dato en la memoria por el bus de datos circula éste, a la par que por el de direcciones viaja la posición de memoria a la que hacemos referencia.

Gracias al Bus de datos, el microprocesador puede tomar información de la memoria.

Y gracias al bus de direcciónes se puede indicar a que celdilla de memoria nos estamos refiriendo.

LINEAS DE BUS

Sabemos que un bus, sea del tipo que sea, no es más que circuitería. Mirándolo desde el punto de vista lógico, ya he comentado que el físico no nos importa de momento, son como líneas paralelas de cable que parten desde el microprocesador y que van a conectarse con la memoria (nota: indudablemente el bus se conecta con más elementos pero vamos a estudiar tan sólo la conexión CPU y memoria por el momento). En el esquema lo podéis ver con mayor facilidad. En él podéis ver dos de los tipos, que son los que nos interesan por ahora.

Lo que realmente importa a cerca de el bus es su tamaño. El bus original de PC de IBM constaba de 20 líneas de direcciones y 8 líneas datos. Pero, ¿qué es lo que circula realmente por el bus? Pues señales eléctricas que son interpretadas en formato binario. Es decir, las diferencias de voltaje que circulan por las líneas del bus indican si se trata de un cero o un uno. Supongamos que queremos enviar el dato 1234h por el bus de 20 bits (o 20 líneas) de direcciones. Como ya sabemos, el número hexadecimal anterior se puede pasar a formato binario de la siguiente forma:

4= 0100
3= 0011
2= 0010
1= 0001

Luego nuestro número binario será 0001001000110100. Como tenemos 16 bits sólo, rellenaremos con ceros los 4 bits superiores del número de 20 bits que va ha pasar por el bus como se muestra en la figura 2. Los cuatro bits superiores a los que he hecho mención están marcados con color celeste de fondo.

Para enviar un dato por el bus de 8 bits de PC (recordad que por ahora siempre estamos haciendo mención al microprocesador 8086 inicial, los demás, 286, 386 y 486, presentan características diferentes que veremos en su momento) tendremos que usar la misma lógica sólo que en sólo 8 líneas. Es decir, es imposible enviar el dato 1234h por dicho bus. Es demasiado estrecho para ello, pero sí el dato 34h, ó 5Fh, ó 2Ah, etc. ¿Véis la importancia del ancho de bus? A mayor número de líneas mayor será la capacidad del microprocesador de enviar datos y direcciones de una sóla vez, ganando en potencia y rapidez.

RELACION ENTRE EL BUS Y LA SEGMENTACION

Como apunté antes, la segmentación de la memoria se debe a la forma en que el microprocesador tiene estructurado el bus de direcciones. Sólo tenemos 20 líneas, que hacen que sólo podamos indicar direcciones de 20 bits. Lo que ocurre es que los registros internos del microprocesador son de 16 bits. Con esta cantidad de bits se pueden indicar un máximo de 65536 posiciones de memoria. Para que lo veáis más claro, vamos a poner un ejemplo. Supongamos que la memoria de nuestro PC es de un megabyte (1048576 posiciones o celdillas de memoria) y que queremos introducir el dato 10h en la posición 234.556 decimal. Pasamos este número a hexadecimal, 3943Ch, y nos damos cuenta de que ocupa 20 dígitos binarios. Ahora la instrucción en ensamblador que llevaría a cabo la operación sería la siguiente:

```
MOV AX, 3943Ch
MOV (AX), 10h
```

Pero nuestro ensamblador nos produce un error indicando que el número es demasiado grande. Intentamos meter un zapato grande en un pie pequeño.

Para lograrlo se echa mano a un truco. El microprocesador siempre descompone los direcciones en dos partes:

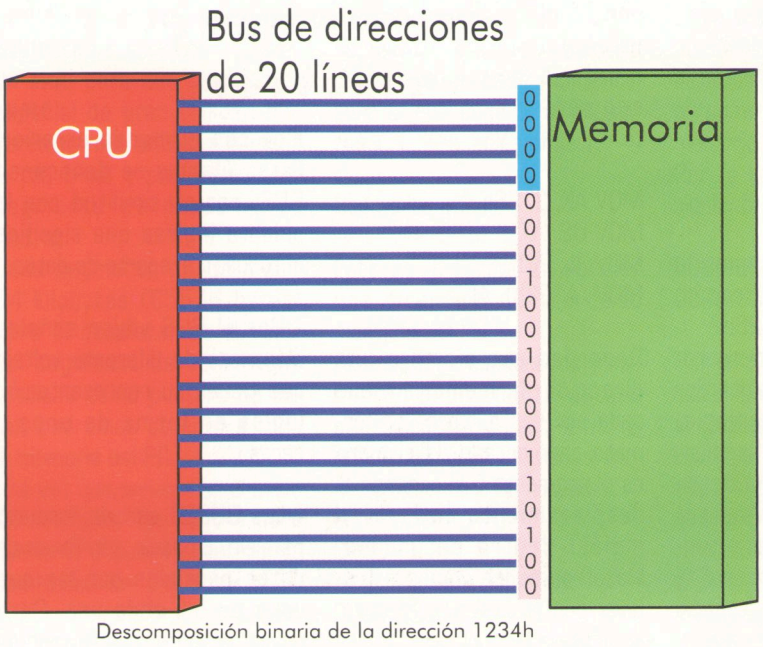
1) La que indique el registro de segmento
2) La que indique un registro, un dato implícito, etc

Un ejemplo:

Para acceder a la posición anterior hay que hacer la siguiente operación. En primer lugar dejar los dos bytes inferiores (16 bits) aislados. Tendríamos 943Ch por un lado, y después tomar el byte superior y guardarlo en el registro de segmento en cuestión, es este caso DS, pero antes deberemos dividir el número por 16. ¡Qué follón madre mía!

Vamos a explicarlo al revés y ya veréis como se entiende mejor.

Supongamos que queremos introducir un dato en una posición de memoria dada, por ejemplo , 0300:943C, que como véis está en formato segmentado. Sabemos, por la constitución del bus de direcciones que la dirección final o efectiva que pasará por este bus ha de ser de 20 bits, con lo que se puede direccionar un megabyte. Es decir hemos de construir un dato de 20 bits a partir de dos de 16 bits. Para ello se realiza la opera-



Descomposición binaria de la dirección 1234h

Figura 2, Bus de 20 líneas

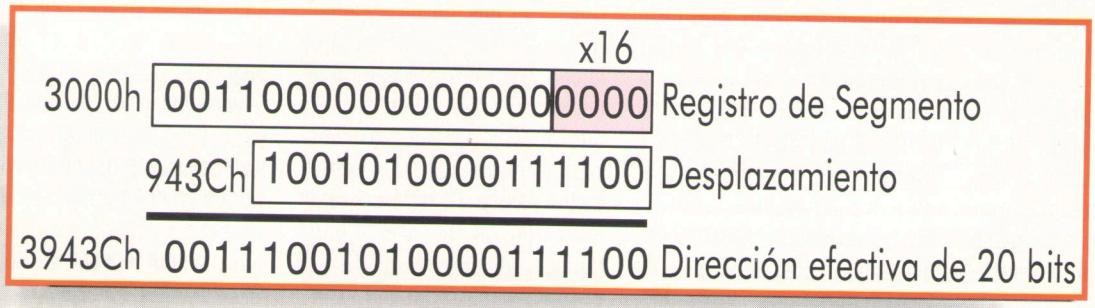


Figura 3, paso de dirección segmentada a dirección efectiva de 20 bits

ción que podéis observar en la figura 3. Se multiplica el registro de segmento por dieciseis y se le suma el desplazamiento que venga indicado por un registro o dato implícito.De esta forma tan especial el microprocesador puede enviar por el bus de direcciones el dato que indicará a los elementos gestores de la memoria, cual es la posición de memoria en concreto, a la que está haciendo referencia. Por lo tanto siempre indicaremos una dirección mediante la unión de dos datos.

Un ejemplo:

Si queremos guardar el número hexadecimal 23h en la posición 3726:A123 de memoria, ¿qué tendremos que hacer?

Ya sabemos que la orden sería algo así como:

```
MOV AL, 23h
MOV (A123h), AL
```

...pero, ¿cómo indicamos la primera parte de la dirección, es decir, el número 3726h?

Cuando indicamos al microprocesador que queremos guardar un número en la memoria, siempre lo hará en el segmento de datos que está indicado por el registro DS. Si siempre vamos a utilizar el mismo segmento (un segmento tiene 64k de longuitud, luego hay espacio para bastantes datos), entonces no hará falta más que inicializar DS con ese valor al principio. Si por el contrario, en medio de nuestro programa queremos guardar el dato en otro segmento distinto del habitual que hemos usado desde el principio de nuestro programa, deberemos cargar DS expresamente con el valor del nuevo segmento:

```
MOV AX, 3726h
MOV DS, AX
MOV AL, 23h
MOV (A123h), AL
```

Recordad que siempre una dirección de memoria viene dada por dos números en formato segmentado. El primero es el segmento y el segundo el desplazamiento (offset en inglés) dentro del mismo. Nosotros nunca vamos a trabajar con la dirección efectiva, es decir, la dirección de 20 bits que el microprocesador obtiene de las dos datos segemento:desplazamiento.

Bien, pues una vez que esto queda claro, hagamos un alto en el camino para repasar los puntos anteriores.

LA TRANSFORMACIÓN DE DIRECCIÓN SEGMENTADA A DIRECCIÓN EFECTIVA PERMANECE SIEMPRE TRANSPARENTE AL PROGRAMADOR. SÓLO LA CPU REALIZA ESE TRABAJO, SIN NUESTRA INTERVENCIÓN.

- El microprocesador tiene una serie de líneas eléctricas con las que se comunica con el mundo exterior. Estas últimas se denominan BUS, por su similitud con la palabra inglesa que significa autobús, transporte de datos.

- Por el BUS viajan señales eléctricas de diferente voltaje que tienen su representación lógica en forma de unos y ceros.

- Los BUSES son de formato paralelo. Es decir, son líneas de datos paralelos por las que viaja un bit cada vez. Para enviar un byte necesitaríamos 8 líneas, una palabra (word) necesita 16 líneas, una doble palabra 32 líneas, y así sucesivamente. Por ello se dice que un ordenador es de 16, 32 ó 64 bits. Mayor BUS significa mayor potencia y rapidez.

- A mayor ancho de BUS, mayor capacidad tendrá la CPU para comunicarse con el resto de los elementos que estén conectados a éste, ya que es capaz de enviar más bits de una sóla vez. Para que lo entendáis mejor, si con un ancho de BUS de datos de 8 bits (como el antiguo 8086) hace falta 4 tiempos para enviar una doble palabra a la memoria, con un ancho de 32 bits tan sólo se requiere uno de estos tiempos.

- El microprocesador tranforma nuestras direcciones segmentadas (seg:offset) en direcciones efectivas de 20 bits. Con esta cantidad de líneas en el BUS de direcciones se pueden acceder 1.048.576 posiciones de memoria, que equivalen justamente a un megabyte. Para la transformación se multiplica el segmento por 16 (que es igual que desplazar cuatro posiciones binarias a la izquierda dicho número) y se le suma el desplazamiento. Nosotros, los programadores, ni siquiera nos ente-

ramos de este proceso. Sin embargo, esta es la causa de la utilización de la segmentación de memoria, la cual ya no se utiliza cuando cuando trabajamos con el ordenador en modo protegido. Por ahora, deberemos acostumbrarnos al modelo segmentado de memoria.

- El microprocesador utiliza los registros de segmento como los valores por defecto ante ciertas operaciones del microprocesador. Por ejemplo, la orden MOV (1234h), FFh, guardará el dato en la posición DS:1234h, es decir, en el segmento que ya en ese momento indique DS y en el desplazamiento 1234h. Habrá que tener cuidado con esta característica a la hora de guardar un dato en un segmento diferente al que indique el registro de segmento.

PARA NENES PEQUEÑOS

Paso ahora a explicar lo anterior de forma que hasta un niño de pecho lo pueda entender. Esto es algo a lo que se ha hecho incapié en múltiples cartas que he recibido a mi nombre y concernientes a esta sección en particular.
Por ello, y para que veáis que os hago caso, siempre incluiré este apartado al final de mi curso y que espero os sea de gran ayuda.
Manos a la obra:
El microprocesador es como un gran pulpo pero cuyos tentáculos son pequeños. Puede realizar operaciones de gran potencia, pero con restricciones. Una de ellas es el hecho de que no puede manejar la memoria en su totalidad. Para ello, utiliza 4 tentáculos a los que se conoce como registros de segmento. Estos de llaman:
1) Registro de segmento de datos:DS
2) Registro de Segmento de código: CS
3)Registro de segmento de pila: SS
4)Registro de segmento extra. ES

Los cuatro brazos de nuestro pulpo actúan como punteros de la memoria y en cada uno de estos segmentos o trozos del total de la RAM y ROM del PC guarda un tipo de información diferente a los ojos del microprocesador. Es decir, para tenerlo todo bien ordenado, tendremos un lugar para el programa a ejecutar (código), un lugar para los datos que éste va a utilizar (datos), un sitio para una estructura especial de gran utilidad que veremos más adelante llamada la pila (Pila) y por último otro segmento de datos extra (extra).
Cada segmento puede medir 64 kilobytes (65536 bytes) pero si desplazamos el valor del segmento de código que funciona como puntero de inicio, podremos acceder a toda la memoria del PC.
Una vez que señalamos el inicio de nuestros segmentos para que la CPU pueda saber donde comienzan las estructuras de nuestro programa, tan sólo tendremos que indicarle a que

Por defecto los se introduce en el segmento de datps

1

```
MOV (DIRECCIÓN), DATO
```

SEGMENTO DE DATOS ------- DS

la dirección será DS:DIRECCIÓN

2

```
MOV AX, NUEVO_SEGMEN
MOV DS, AX
MOV (DIRECCIÓN), DATO
```

SEGMENTO ANTERIOR | NUEVO_SEGMEN

bye en concreto dentro del segmento, estamos haciendo referencia en cada momento, para lo que usamos el desplazamiento (en inglés offset).

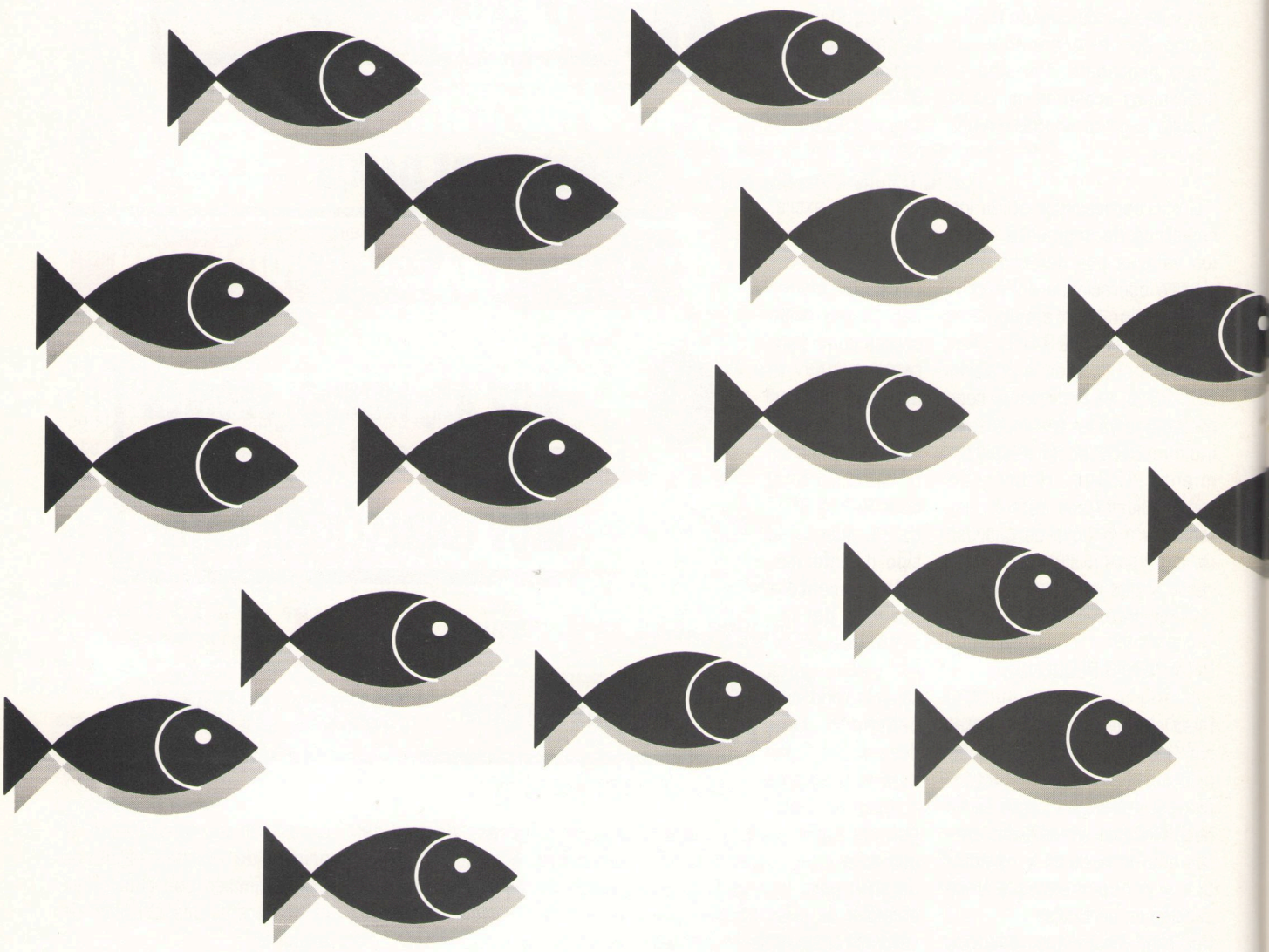
Segmento:offset

Esta es la forma de señalar una dirección dentro de la memoria del PC.
Algunos más espabilados ya se habrán dado cuenta de que una dirección de memoria se puede nombrar de varias formas diferentes. Por ejemplo:
la dirección
0000:0010
es igual que la dirección:
0001:0000
Lo único que hemos hecho es cambiar el segmento, o sea, el primer de los dos números hexadecimales que conforman el "nombre de una dirección de memoria".
Por lo tanto, las direcciones siguientes también son iguales:

0000:0300
0030:0000

Por ello siempre deberemos saber los datos; el segmento y el desplazamiento.
Bien os dejo de aburrir por este mes, no sin advertiros que la materia sobre la segmentación de memoria es de gran importancia en el aprendizaje del ensamblador de PC.
Estudiad un poquito.

No vaya en contra de todos...

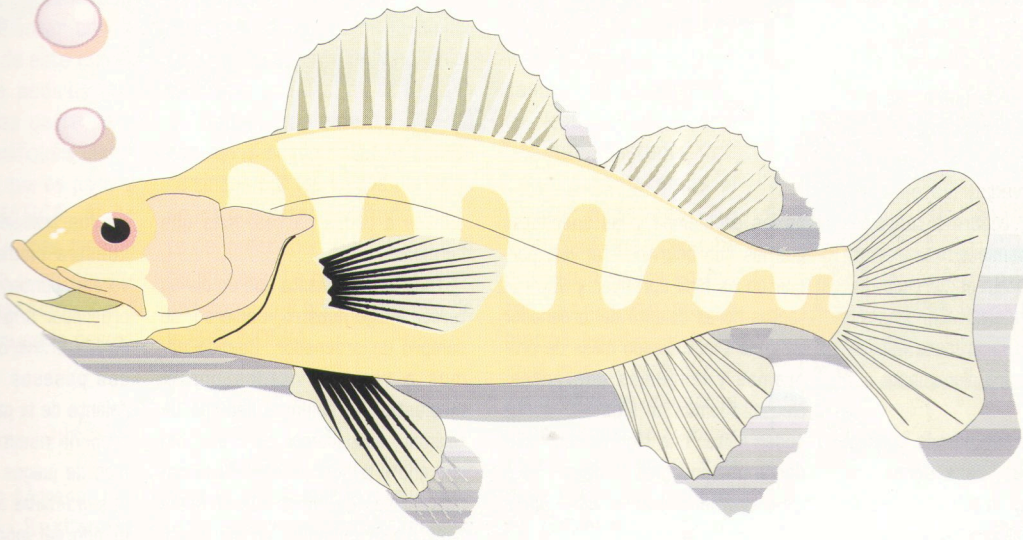


Asinfo

Asesoramiento y suministros informáticos

Estamos en el teléfono
y
FAX
956-813129

C/ María Auxiliadora 38
11520, Rota, (Cádiz)

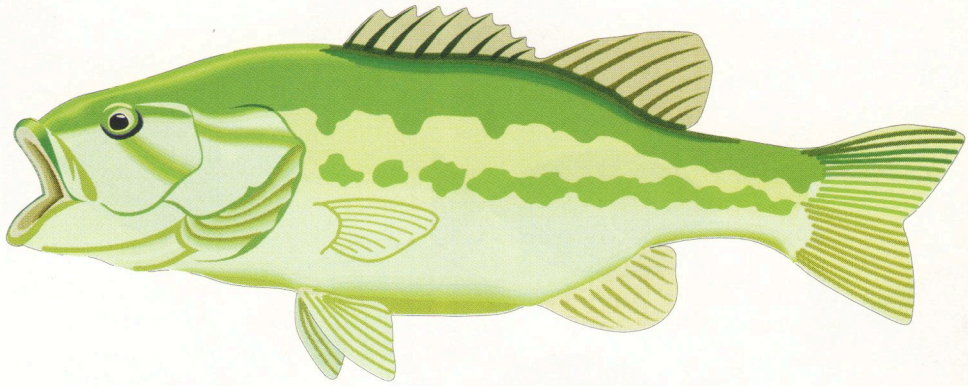


En Asinfo encontrará su tienda amiga, su tienda de informática.
Con nosotros resolverá todos sus problemas con los ordenadores y los periféricos.
Con nosotros obtendrá los mejores precios, las mejores ofertas.
Pida gratuitamente nuestro catálogo de productos.

CARTA DEL LECTOR

Por:

Jack Ladoire
Alabama



Antes de que mi español incorrecto manche las páginas de vuestra revista, me gustaría felicitaros, me refiero a todos los que hacéis posible que la revista Código Juego salga al mercado, por la fantástica iniciativa que habéis tenido creando la primera publicación que se dedica plenamente a la programación.

Mi sorpresa fue grande cuando, hace unos días, pase cerca una librería situada en los aledaños de la facultad de informática de Sevilla. Paseé la mirada, como de costumbre, por toda la hilera de revistas y cuando ya me iba ví una revista de color rojo con una muchacha que tocaba el violín en la portada. La cogí de la repisa y no daba crédito a mis ojos. ¡Programación de videojuegos! Llevo años queriendo tener tiempo libre suficiente, cosa que cada vez me resulta más difícil debido a mis estudios, para poder dedicarlo al estudio del lenguaje ensamblador y la creación de todo tipo de aplicaciones con este potente medio.

Rápidamente, y tras leer vuestra revista, os llamé para pediros los números atrasados que no había adquirido hasta el momento. Realmente os felicito por el trabajo que estáis haciendo. Tras leer las revistas de cabo a rabo, me gustó mucho la sección en la que los lectores mandan sus cartas contándoos los problemas y las anécdotas que les han ocurrido. Tal vez, porque no es fácil aprender y llevarse tantas horas delante del ordenador mirando y estudiando miles de programas y lenguajes diferentes. Necesitábamos un sitio así, donde poder soltar una lágrima, un lugar donde nos podamos confesar. Pues bien, ávido de soltar mi llanto particular, ahí va mi historia.

Comencé a interesarme por la informática de una forma muy particular. Allí en Alabama, de donde yo procedo, asistía al High School, lo que aquí es el Instituto de Enseñanza Media. Todas las asignaturas que teníamos eran un verdadero rollazo (¿cuál no?), así que me aprovechaba de vivir en un pueblo pequeño, rodeado de campo por todos, y me gustaría resaltar lo de "todos", lo sitios. Magníficos arboles, alucinantes arrollos y pantanos adornaban nuestros paisajes, que eran y son de una belleza extraodinaria. Con tanto sitio bonito y tantas cosas que se podían hacer, ¿cómo iba yo a aguantar en el caluroso instituto? Así que, a la menor ocasión, me quitaba de en medio y me marchaba de pesca con Richard, un buen amigo mío. Mi amigo Richard y yo éramos uña y carne, como decís aquí en España. Juntos a todas partes, más que hermanos.

Todo iba bien hasta que un día un tercer amigo nuestro, Cameron, se compró un ordenador. Cuando fuimos a verlo, invitados por él mediante una solemne llamada de teléfono, acabábamos de ir a un rio cercano a pescar lo que allí llamamos "bass" que creo que se llama "perca" en español. No me gustó nada el ordenador. Es decir, no me llamó la atención en lo más mínimo. En cuatro o cinco ocasiones fuí a casa de Camerón y cada vez que me enseñaba algo en el ordenador me aburría de muerte. La cosas fueron así hasta varios meses después. Richard y yo quedamos en ir al cine a ver una película que nos habían recomendado. El nombre "War Games" (juegos de gerra) que tal vez os suene. Yo la he visto como unas veinte veces desde entonces. A partir de ese momento, mi visión aventurera de la vida tipo Tom Sawyer, hizo que mirase a los ordenadores de otra forma. Introducirme en el ordenador del Pentágono y cambiar el destino de mundo desde el teclado de un simple ordenador en el dormitorio de tu casa, era algo que definitivamente llamó mi atención. Se acabó la pesca para Richard y para mí. Entre los dos, trabajando en un Burger de nuestra ciudad durante todo el verano, conseguimos dinero para un ordenador original de IBM, es decir de los primeros PC's. Parecíamos dos posesos. Todo el santo día delante de la pantalla del ordenador creando nuestros propios programas de juegos en BASIC. También, nos gustaba introducirnos en el mundo del módem y las comunicaciones con otros usuarios. Acudimos a un par de ferias de informática cercanas y conocimos muchos amigos. Nos dimos cuenta entonces que los ordenadores no tienen nada con la película que despertó nuestros apentencias por este mundo, y que todo aquello era ficción, pero aún así seguimos con las mismas ganas de trabajar en este mundillo. Richard está en una universidad de Alabama, estudiando informática y yo me vine a España dado que mi padre es de aquí. Seguimos en contacto directo dado que los dos estamos conectados por la Internet. Estoy muy contento en vuestro pais donde también los ordenadores despiertan pasiones. Lo que si echaré siempre de menos son aquellas enormes percas que capturabamos mi amigo y yo allá en la vieja Alabama.

La tintilla informática

¡QUE VIENE LA ERGONOMIA!

Por:

Miguel A. Lobato

Antes de comenzar con la carta queja de este número, quisiera pediros que cuando enviéis una carta para esta sección, especifiquéis claramente en el sobre que es para la TINTILLA INFORMATICA. Al no indicarse, se tarda más tiempo en su clasificación y en consecuencia la contestación será más lenta de lo que sería deseable.

EXPOSICION DE LOS HECHOS

Francisco Javier Moreno, nos escribe desde Santander (Cantabria), nos comenta los problemas que viene padeciendo desde hace algún tiempo tanto en su espalda como en su cuello. Francis, como le gusta que le llamen sus amigos, es administrativo en una empresa de productos cárnicos. Le viene pegando a la informática desde hace unos tres años, y ahora es cuando se le han agudizado sus molestias.

"Después de trabajar en la empresa, le dedico unas pocas horas en mi casa al ordenata , al terminar me encuentro tan cargado de espalda y cuello que debo tumbarme en el sofá del salón para descansar, casi no puedo moverme durante un rato, es insoportable el dolor".

Francis ha decidido ante estos episodios repetitivos visitar al médico de la Seguridad Social, os podréis imaginar, tras esperar su turno como buen ciudadano, el doctor lo reconoció "a medias" (nos confiesa) y le envió al traumatólogo para que le practicasen una exploración más exhaustiva. El resultado de las radiografías, así como la palpación del traumatólogo no dejaban lugar a dudas, no tenía nada de nada, simplemente que su espalda y cuello estaban sobrecargados por los largos periodos en los que estaba sentado delante del ordenador del trabajo o el de su casa. Le recetó descanso, una pomada y que hiciera uso de la ERGONOMIA.

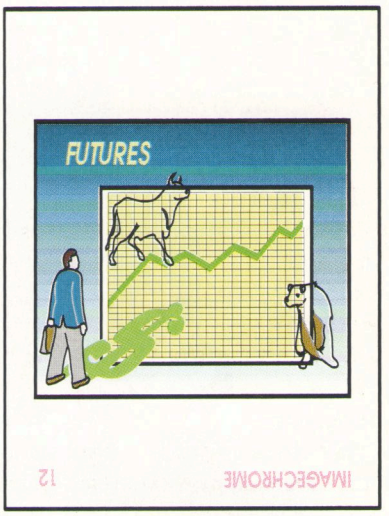
COMENTARIO

La palabreja se las trae ¿No os parece? Al leer la carta, tuve irremediablemente que recurrir al diccionario que tengo en casa, preguntándome qué tendría que ver el dolor de espalda de Francis con esa palabra que se me antojaba de la era espacial. La definición que encontré es la siguiente:

"Ciencia del trabajo.Aquella parte de la ciencia social que trata de la organización del trabajo, de la racionalización del mismo y de los métodos industriales y la orientación profesional. Se estudia en ella, no sólo el elemento psicotécnico (que conduce a la orientación y selección de personal), sino también todo aquello que podría llamarse organización y ambiente de trabajo: homogeneidad de los equipos, condiciones de seguridad, la iluminación... y todos aquellos factores que pueden conducir al aumento del rendimiento con la mejor economía del esfuerzo."

¡Tanto rollo! Para decirle que necesita una silla en condiciones y un protector de pantalla que cuiden su columna y sus ojos. ¡Jo...! con el nombrecito.

Realmente si lo pensamos con detenimiento, la cosa nos podría resultar bastante útil. Cuando compramos el primer ordenador, lo único que nos preocuba era uno o varios enchufes, una mesa más o menos amplia y poca cosa más. A medida que avanzabamos nuestras necesidades de espacio fueron creciendo, con unas estanterias, material de papeleria, disquetes, una silla cómoda, etc. por lo que nos pasamos a un cuarto casi sólo para la informática. A decir verdad, la comodidad se basaba en aquellos entonces casi exclusivamente en tener todo lo más a mano posible, sin ir al cuarto del hermano/a o al salón para coger el libro que no nos cabe porque pensabamos en un uso mínimo.



Los equipos cada vez son mas pequeños y mas potentes, por lo que el espacio que teneníamos era suficiente, cuando nos dijeron que las pantallas podían producir malestar en los ojos.. nos compramos el filtro de pantalla. La variedad de estos en el mercado es amplísima, por lo que hacer referencia a ellos me parece que está fuera de lugar.

Nos seguimos olvidando de aspectos tan cruciales que hasta ahora habían estado en segundo plano como:

1.- la iluminación. Debe ser adecuada y que no molesten los reflejos. Poner luces indirectas siempre que se pueda porque la exposición directa es igualmente perjudicial a los ojos.

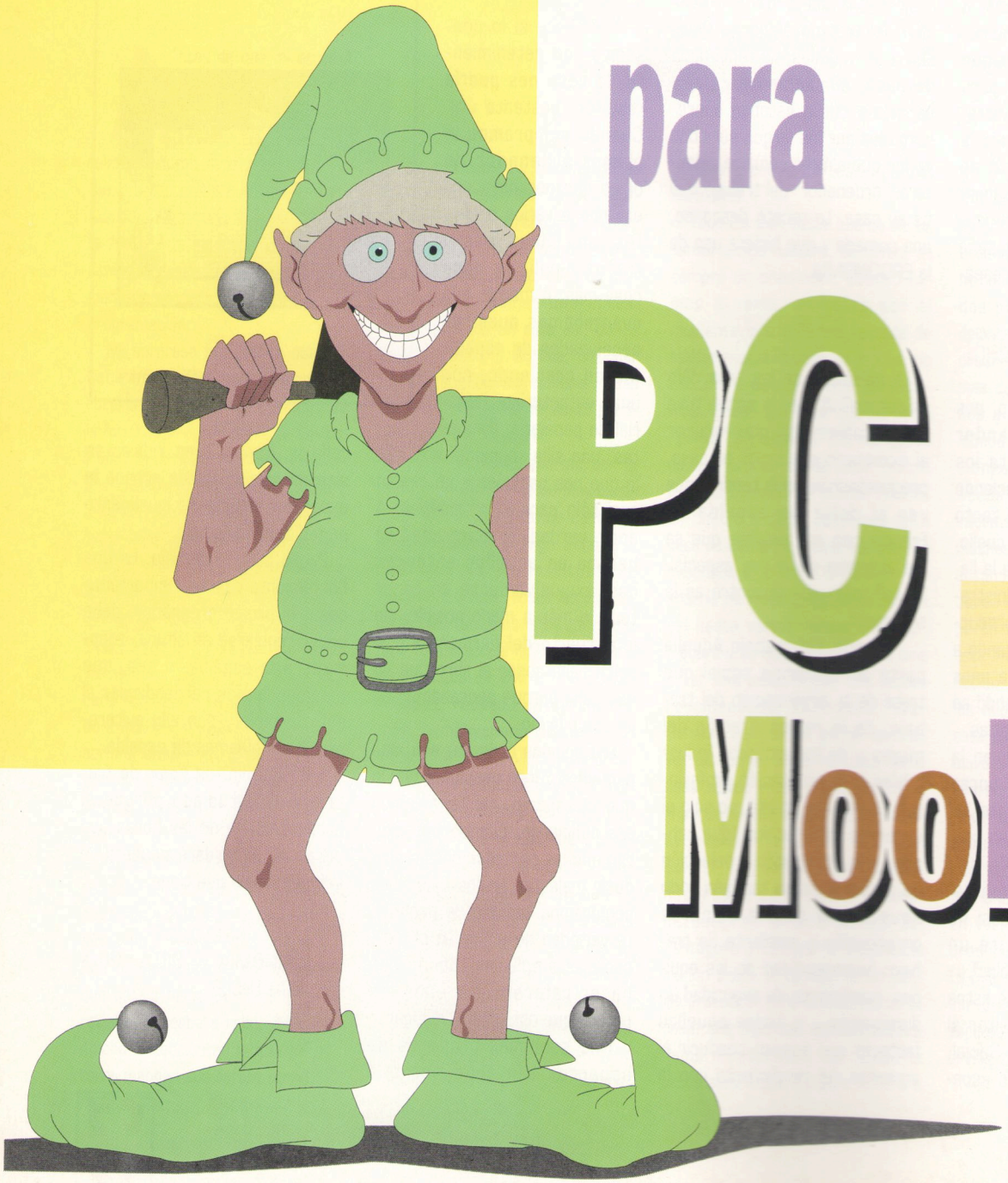
2.- La silla de trabajo. Es una herramienta casi tan importante como el propio ordenador, debe poder regularse en altura, adaptarse a la espalda, o sea, que estemos lo más perpendicular al suelo posible y con ello evitaremos los problemas de espalda.

En definitiva, un lugar de trabajo muy estudiado porque si hacéis balance de las horas que pasais en él, os daréis cuenta de la importancia que tiene.

MORALEJA

ERGONOMIA = SILLA+PROTECTOR+LUCES...

Así se hace un juego para PC



5ª PARTE

Mookie

Por:

Ramón López

Ya llebamos un largo recorrido juntos. Hemos aprendido muchos aspectos importantes de la programación de videojuegos utilizando como medio el potentísimo lenguaje ensamblador para ordenadores PC. Temas como la apertura, lectura y cierre de archivos, comprobación del sistema operativo de la máquina, actualización de un vector de interrupción, comprobación del teclado, etc, se han desglosado dentro de el listado MAIN.ASM que os fue entregado en el número uno de la revista. Si habéis estudiado un poco dicho listado os habréis percatado de que, aunque se trata de un programa corto hasta el momento, es capaz de hacer muchas cosas.

A fin de explicar estas pequeñas materias, hemos tenido, necesariamente, que desviarnos poco a poco de nuestro tema principal; los efectos de fundidos. Por ello, y para no prolongar más su estudio y pasar ya a otros temas, como es el uso de la tarjeta de sonido en nuestro juego MOOKIE, vamos a dar por concluido este repaso con el número de hoy.

Como hemos visto ya, la estructura de funcionamiento del efecto "fading" se apoya en el uso de tres tablas de datos a las que hemos llamado DACTABLE1, 2 y 3. Cada una con un tamaño de 768, bytes correspondientes a otros tantos registros R, G y B de color del DAC de la VGA, nos sirven para almacenar temporalmente los datos que se dirigen hacia la memoria de vídeo del PC y en consecuencia a la pantalla.

Pues bien, los datos RGB originales de la pantalla que queremos presentar ya sabemos que se encuentran en la cabecera del archivo gráfico con extensión .SCR, el cual está en formato crudo, es decir sin comprimir. Estos bytes de la paleta sabemos que se cargan en DACTABLE2. Pero, ¿cuál es el proceso que sufren?.

DEGRADADO PROGRESIVO

Lo que vamos a hacer es lo siguiente. Se trata de convertir un dato de DACTABLE2 en su correspondiente de DACTABLE1 pero poco a poco, gracias a el transcurso de 3 subrutinas: LUZ, DIMMER y DACVIDEO. Hemos de distinguir, llegados a este punto, entre los dos tipos de fundidos que vamos a llevar a cabo.

FUNDIDO A COLOR

Cuando tenemos una pantalla "negra", o sea, sin nada en ella, y queremos que los colores vayan apareciendo progresivamente se denomina fundido a color.

```
;*****************************************************************
; Subrutina LUZ, encargada de apagar o encender una serie de registros del DAC
; para producir efectos de FADING (fundidos)
; Entradas: AX, Registro de DAC a partir del cual empezar
;           CX, numero de registros a tocar
;           BL, flag que indica fundido a negro =0
;                              de negro a color =1
;*****************************************************************
LUZ             PROC    NEAR
                MOV     BP,AX
                MOV     DX,SEG DATA
                MOV     DS,DX
                MOV     ES,DX
                MOV     [NO_REGS1],CX
                MOV     DX,AX
                SHL     AX,1
                ADD     AX,DX
;AX=AX*3
                MOV     DX,CX
                SHL     CX,1
                ADD     CX,DX
;CX=CX*3
                MOV     DI,OFFSET DACTABLE3
                ADD     DI,AX
                MOV     SI,OFFSET DACTABLE1
                ADD     SI,AX
                AND     BL,BL
                JNZ     LUZ_A
                MOV     BH,AL
                PUSH    CX
                PUSH    DI
                MOV     DI,OFFSET DACTABLE2
                ADD     DI,AX
                XOR     AL,AL
                REP     STOSB
;INTRODUCE CEROS PARA
 FUNDIDO A NEGRO
                POP     DI
                POP     CX
                MOV     AL,BH

LUZ_A:          XOR     BX,BX

LUZ_1:          PUSH    SI
                PUSH    DI
                PUSH    AX
                PUSH    CX

                PUSH    DI
```

```
LUZ_2:          LODSB
                PUSH    SI
                ADD     SI,767
                MOV     AH,[SI]
                POP     SI
                XCHG    AH,AL
                CALL    DIMMER
                LOOP    LUZ_2

                MOV     CX,[NO_REGS]
                POP     SI
                MOV     AX,BP
                CALL    DACVIDEO
                POP     CX
                POP     AX
                POP     DI
                POP     SI
                ADD     BX,4
                CMP     BX,40H
                JLE     LUZ_1
                RET
LUZ             ENDP


DACVIDEO        PROC NEAR
                pushf
                MOV     AH,AL
                CALL    RETRAZO
                MOV     AL,AH
                MOV     DX,03C8H
                OUT     DX,AL
        ;REGISTRO DE COMIENZO
                INC     DL
DAC_1:          LODSW
                OUT     DX,AL
                MOV     AL,AH
                OUT     DX,AL
                LODSB
                OUT     DX,AL
                LOOP    DAC_1
                popf
                RET
DACVIDEO        ENDP


DIMMER          PROC NEAR
                MOV     DH,AH
                MOV     DL,CL
                PUSH    DX
                SUB     AH,AL
                XOR     AL,AL
                MOV     CL,6
                SAR     AX,CL
                IMUL    BX
                POP     DX
                SUB     DH,AH
                MOV     AL,DH
                STOSB
                MOV     CL,DL
                RET
DIMMER          ENDP
```

Forma de llamar a subrutina LUZ

```
MOV AX, REGISTRO COMIENZO
MOV CX, NUMERO REGISTROS
MOV BL, FLAG DE EFECTOS
```

En DACTABLE2 deberemos tener los datos RGB de los colores finales y en DACTABLE1 los actuales, es decir, negro para cada uno de los 256 colores de la paleta. Por ello, y dado que el negro se representa como Rojo=0, Blue=0 y Green=0 en DACTABLE1 tendremos 768 ceros.

FUNDIDO A NEGRO

Se trata del efecto contrario. Los colores finales ya están en la pantalla y tenemos que irlos oscureciendo paso a paso. Para ello, en DACTABLE1 tendremos los colores que están en la pantalla y en DACTABLE2 los negros.

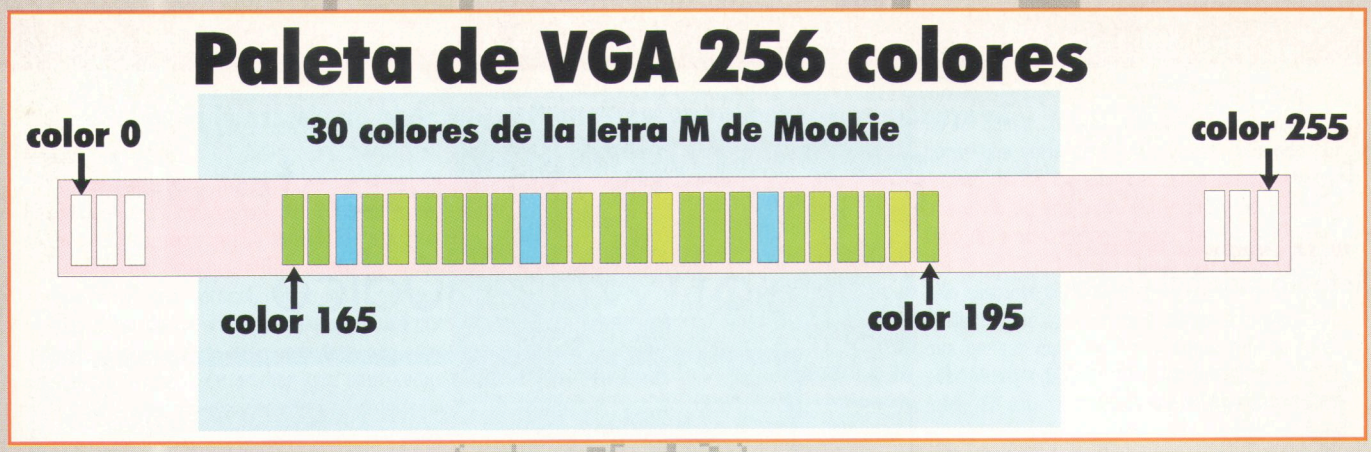
Si te das cuenta en DACTABLE1 tendremos los colores iniciales, o sea, los que en ese momento se pueden ver en la pantalla. En DACTABLE2 los colores finales, en los que queremos que se conviertan los que están en DACTABLE1 y en DACTABLE3 iremos guardando los colores intermedios. Por tanto, durante el efecto los valores de las tablas 1 y dos no cambian, tan sólo los datos de DACTABLE3 irán cambiando a cada paso o escalón del apagado o encendido de la pantalla.

DACTABLE3

Vamos a ver como actuar sobre los datos RGB de DACTABLE1 y 2 para que el efecto se produzca.

Si estudiásteis los comentarios del artículo sobre la VGA de los números anteriores de Código Juego, ya sabréis que los colores son más luminosos cuanto más alto es el componente RGB. Así, dados dos colores como (R=100 G=100 B=100) y (R=200 G=200 B=200) el segundo es mucho más cercano al blanco que el primero. Por tanto, el blanco absoluto será R=255 G=255 y B=255 y el negro, como vimos antes, será todo lo contrario, con los tres componentes a cero. De lo que se trata en el efecto es de realizar un proceso progresivo de apagado o encendido del color en cuestión de manera que ofrezca la ilusión de una perdida de luminosidad. Para ello realizaremos un bucle de forma que cambiemos el número de registros que deseemos pero en un cierto número constante de pasos. Si hiciésemos el cambio repentinamente no advertiríamos nada ya que sería muy rápido.

La subrutina encargada de realizar este apagado o encendido se llama LUZ y que podéis ver listada en CJ1 y en este número también. Este trozo de código es muy potente ya que permite apagar o encender todos o un cierto



número de registros del DAC. Esta posibilidad es utilizada al principio de MOOKIE cuando hacemos que las letras aparezcan una a una.

La forma de llamar correctamente a LUZ la podéis ver en la figura. En AX se indica a partir de que registro del DAC le queréis aplicar el efecto. En CX el número de registros que se verán afectados y en BL indicaremos si el efecto es de apagado o de encendido.

LUZ va tomando los colores RGB de cada registro de color del DAC de DACTABLE1 y 2 y se los pasa a la subrutina DIMMER (apagador en inglés) que es la que realiza los cálculos necesarios con estos valores, mediante el uso de rotaciones de tipo aritmético (SAR).

Hay que decir que los valores de las tres tablas se corresponden uno a uno. Es decir, el registro de color 0 del DAC esta guardado en la posición cero de DACTABLE1, y en la misma posición de DACTABLE2 y 3. También, se guardan como RED, GREEN, BLUE, o sea con este mismo orden.

Finalmente, cuando ya hemos calculado todos los datos, los cuales han ido almacenándose en DACTABLE3 gracias a la orden STOSB que hay en la subrutina DIMMER, se llama a DACVIDEO que es la que pone los datos calculados en los registros de color de la VGA. Una vez que la primera andanada se ha sacado por pantalla se vuelven a realizar los cálculos y se repite la operación. Pero ¿cuántas veces se hace el bucle?. Eso depende de el registro BX que está situado justo al final de la subrutina LUZ:

```
ADD BX, 4
CMP BX, 40H
JLE LUZ_1
RET
```

Si os dáis cuenta, el registro se va incrementando en saltos de 4 hasta llegar a 40h (64 dec), por lo que el bucle se realizará en 16 pasos (4*16=64). Si alteramos este valor, el número de veces en que se realize el bucle variará, enlenteciéndose o aligerándose. Esto queda a tu gusto.

En la tabla adjunta puedes ver el funcionamiento de la subrutina LUZ de forma esquematizada. Algunos detractores de desperdiciar la memoria habrán puesto el grito en el cielo cuando he utilizado 3 tablas de 768 bytes, pero la verdad es que merece la pena, ya que el efecto que se logra es de gran plasticidad y va a ser utilizado exhaustivamente a los largo de MOOKIE.

Hasta el mes que viene, la Soundblaster nos espera.****



2ª PARTE

Te ayudamos a escoger que juego realizar echando un vistazo al pasado

El pasado y presente del software lúdico.

Por:

Antonio Marchal Cortes



Bienvenidos a este alucinante viaje por la historia de los videojuegos.

Antes de nada me gustaría explicar que esta serie no trata de pasar lista a todos los videojuegos, sino solo a aquellos que resaltaron por algo en especial, bien por su idea, bien por ser los primeros o por haber introducido algo nuevo en este apasionante mundo del software de entretenimiento. Por eso, si me paso algo importante, por ejemplo, el Arkanoid o el Gaunlet, avisadme. Pero no os extrañéis si no hablamos de todos los Arkanoid_s, ni de todos los Gaunlet_s. Sin más preámbulos pasemos a la cuestión que nos interesa.

El mes pasado estuvimos hablando de la primeras máquinas, así como de las primeras máquinas "decentes", que salieron al mercado. Estás, como las que vamos a hablar hoy, estaban programadas en placas de 8 bit, lo que conlleva una calidad bastante inferior a la que hoy tenemos. Pero lo más importante de ellas, era su adictividad y originalidad.

Arranquemos de una vez. Como anuncie el mes pasado hoy veremos otras máquinas, pero antes haremos un inciso para hablar de ordenadores. Veremos su historia brevemente, así como las características más relevantes de cada uno de ellos.

Según dice los entendidos, el primer ordenador fue el abaco. Esto no es una marca, sino un aparato muy antiguo que utilizan los chinos para calcular cantidades grandes. Este chisme era un rectángulo hueco de madera, con varillas y bolas. Se manejaba, haciendo que cada varilla hiciera la misión de cifra, siendo una varilla las unidades, otra las decenas, otras la centenas, etc. Y había algunos que incluso podían hacer multiplicaciones y divisiones.

Como ya he dicho, el abaco es muy antiguo y no puede considerarse como un ordenador, tal como hoy lo conocemos. El primer ordenador, esta vez mecánico, fue realizado hace menos de doscientos, y esta compuesto de tuercas y engranajes. Sumaba, restaba, multiplicaba, y poco más.

El siguiente ordenador fue ya eléctrico, y funcionaba con bombillas. Este es lo más parecido, a lo que hoy en día es un ordenador, sino fuera por su tamaño, ocupaba un piso entero, y su peso de varias toneladas. Funcionaba con tarjetas perforadas solamente.

Después de esto se consiguió meter todas esas toneladas de bombillas, en una pastilla de silicio, apareciendo los chips.

Por aquel tiempo, los ordenadores sólo lo usaban gente del gobierno o militares, como el pentágono. Por el prohibitivo precio, los ordenadores estaban vedados al consumidor, hasta que un buen día apareció el señor Sir Clive. Este señor fue el primero que comercializó los ordenadores, vendiéndolos aproximadamente alrededor de las sesenta mil pesetas. El primero que sacó fue el ZX80, y después el ZX81. Estos eran muy parecidos, excepto en la memoria, mientras el primero tenia 1K, el segundo podía llegar hasta los 16K. Los dos eran en blanco y negro, no podían manejar pixels, y su apariencia exterior, era más bien pobre, ya que por no tener, no tenia ni teclas, sino una especie de membrana plana.

Después sacó el mítico Spectrum. Primeramente en su versión 16K, para luego pasar a la de 48K. Esto ya era un gran avance, este ordenador tenia ocho colores, resolución de 256x178, sonido beeper, y podía cargar los juegos desde cassete.

Junto con este gran boom del Spectrum, salieron el MSX, Commodore 64, Amstrad, etc. Todos ellos de 8 bits, y con características muy parecidas, resaltando el Commodore en el aspecto sonoro, aunque este ordenador era muy complicado de programar, lo que le causa un gran detrimento en contra del Spectrum.

Poco después, aparecieron los ordenadores de 16 bits, como el Amiga 500, y el Atari. Estos eran una delicia: música en estereo, gráficos con miles de colores, megas de memoria... todo un lujo. Aunque también era un lujo para el bolsillo, lo que los hacía prohibitivos para algunos bolsillos, que se tenían que conformar con Spectrums, Amstrads, y demás. Mientras el tío Clive, se fue a la bancarrota, y fue absorbida por Amstrad, este último, sacó dos nuevos Spectrum, el +2, con cassete, y el +3, con disquetera de 3 pulgadas. Eston también venían con sonido estereo, y además tenían cientos de miles de juegos de los spectrums anteriores, intentando así competir con Amiga y Atari. Pero entonces hicieron aparición las consolas, barriendo el mercado. Los Amiga bajaron de precio, los Atari también, los Spectrum y demás máquinas de ocho bits desaparecieron. Y por último, aparecieron los PC para el consumidor medio, gracias a la bajada de precio.

Estos PC, siempre habían sido maquina de oficina, lo que hacía que para los videojuegos no estuviera preparado. Eso lo resolvieron rápidamente los vendedores, sacando nuevas placas para mejora gráficos, sonido, etc. Que es como los conocemos hoy en día.

Una vez visto toda la historia, por encima, de los ordenadores, pasemos a lo que dejamos pendiente. Los videojuegos:

Mario Bros, fue, como su nombre indica, la primera máquina donde apareció nuestro conocido amigo Mario. Aparecía con su hermano Luigi, y su misión era acabar con todas las tortugas, cangrejos y moscas que salían por las tuberías. Para ello debía de, primero atontarlas dandole un golpe por abajo, y después darles una patada para mandarlas al vacío. También había otros enemigos,

como bolas de fuego y placas de hielo, que debías esquivar.

Commando es otra máquina que hizo historia, fue una primera máquinas bélicas. Y fue tan famosa, que incluso sacaron una película de ella, protagonizada por "cachitas Suarsenaguer". Con vista aérea y scroll vertical, debías de acabar con oleadas de soldados enemigos que aparecían por la selva. Teniendo al final de cada fase que abrir una puerta a base de acabar con cantidades industriales de enemigos. Era cantidad de difícil a la vez que adictiva.

Donkey Kong JR fue la segunda parte de Donkey Kong, esta vez hacías el papel de hijo de Donkey Kong, a este último lo tenia secuestrado Mario Bros, esta es el único videojuego en que Mario hace de malo. JR tenia que ir por unas lianas, intentado salvar a su padre.

Yier ar Kung-fu, era otro videojuego de artes marciales. Destacaba que en cada fase, el enemigo al cual te tenias que enfrentar, cambiaba, siendo estos cantidad de "simpáticos". Era bastante difícil, lo cual conllevaba una adicción bastante grande.

Moon Cresta, basado en la naves espaciales, tenia scroll vertical y vista aérea. Tenía enemigo de fin de fase, y era terriblemente difícil. Aunque la misión en principio era fácil, acabar con todo lo que apareciera en pantalla antes de que acabase contigo.

Green Beret, fue otra maquina bélica que resaltó. Tenía scroll horizontal, y tu misión era salvar a unos presos en un campo de concentración ruso. Al final de fase tenías un enemigo especial, más difícil que los demás. Era un juego muy adictivo.

Ghost & Goblins, es otro videojuego que hizo historia. En este, un terrible demonio había secuestrado a la princesa, que era tu prometida. Y tú como noble caballero debías salvarla. Con scroll horizontal, tus enemigos eran fantasmas que salían de la tumba, además de pajarracos y otros seres de mal agüero. Al final de cada fase, como iba ya siendo normal, había un enemigo más fuerte que los demás. Esta máquina era superadictiva, además de superdificil. Y fue tan buena, que todavía hay versiones de ella, e incluso sacaron una segunda parte.

Wonder Boy, fue otra maquina de Sega. En esta llevabas a un chico maravilloso el cual tenia que rescata a su chica. Tenía varios niveles de dificultad, pero estos se repartían en cuatro y el monstruo final de fase. Era entre arcade y plataformas, porque aunque lo más importante era acabar con los enemigos, los saltos debían de ser también muy precisos.

Némesis, era otro juego de naves que llego a ser muy famoso. No añadía nada nuevo al tema, excepto su adiccion, la cual le hizo famoso.

City Connection fue una forma distinta de entender los juegos de coches. Con scroll lateral, tenias que ir pintando unas plataformas, mientras esquivabas a una serie de coches de policía. Cada



Algunas pantallas de los juegos más famosos de Spectrum, un ordenador que hizo época.

fase el fondo era de un país, lo que le daba cierta variedad, era bastante adictivo.

La maquina de Popeye, que es la que después versionaron en consolas, ya que para los ordenadores, cogieron los mismos personajes, la misma temática, pero distinta estructura, pero de esa versión ya hablaremos más adelante. La que nos ocupa era una máquina bastante adictiva, por su originalidad. Tenía cuatro fases, y en cada una de ellas Olivia te lanzaba objetos de diversa clase, que tu debías recoger, siempre intenta de evitar al temido Brutus. Podías coger espinacas, y saltar de piso a piso, o optar por las escaleras

Gyrodine. Una máquina muy al estilo Moon Cresta, solo que esta vez, sustituías la nave espacial por un helicóptero, y los enemigos esta vez eran aviones. Tenía varios tipo de disparo, y dispara misiles, lo que le daba una nota de innovación.

Burger Time resalto por su originalidad y sentido del humor. Tú eras un cocinero que debía preparar una serie de hamburguesa, pasando por encima de los ingredientes, eras perseguido por una serie de enemigos que eran, y aquí esta el tope de gracia, desde huevos fritos a salchichas. Era muy difícil, lo que le daba una cotas de adicción bastante considerable.

Circus, fue otra máquina bastante original. Hay que pensar que como por aquel tiempo había poco videojuego, casi todos eran bastante originales. En esta tu representabas el papel de payaso, sin animo de ofender a nadie, el cual debía de pasar una serie de pruebas, siendo estas salto de aro de fuego montado en un león, salto de mono en el cable, equilibrio encima de barriles, salto de trampolín a caballo, y por ultimo salto de trapecio a trapecio. Era muy entretenida, a la vez que difícil y adictiva.

Zippy Race fue una máquina al estilo Spy Hunter pero más tranquila. En ella llevabas a una moto que debía llegar a diferentes ciudades de E.E.U.U., tenía dos partes, la primera con vista aérea y la segunda con vista trasera de la moto y de fondo la ciudad. Era muy difícil y adictiva, ya que los coches que eran los enemigos, tenían bastante mala leche.

Road Fighter muy al estilo al anterior, también tenía vista aérea, aunque aquí llevabas a un pequeñísimo coche hasta la meta de cada uno de los cuatro niveles que tenía. Era muy sencillo, en la concepción del juego, porque a la hora de jugar era muy difícil y adictivo.

West Bank fue una de las primeras maquinas del oeste, que luego seria versionado para algunos ordenadores. En ella hacías el papel de sheriff, estabas en un banco con 12 puertas, debían ingresar dinero en todas ellas, teniendo también que protegerlas de bandidos, todo esto siempre en grupo de 3 puertas.Era bastante original, adictiva, además de una dificultad creciente.

Lunar, era una maquina al estilo asteroid, en cuanto a gráficos, ya que estos era lineas. Tu misión consistía en intentar aterrizar una nave en los parajes lunares. Aunque simple, era una maquina bastante adictiva.

Emsamble Cohete fue otra máquina de corte espacial. Debías acabar con decenas de enemigos, como en todas, pero la novedad estaba en que entre fase y fase debías ensamblar nuevos trozos, lo que hacía tu nave más potente.

Tapper, fue una máquina que aunque no fuese muy conocida, quien la conocía se prendaba de ella, ya que era superadictiva, a la vez que simpática y entretenida. En ella hacías el papel de camarero que tenía que atender una serie de barras por la que venían gente sedienta, tu tenías que servir cerveza a todos antes de que alguno llegara hasta tu y te pegase una paliza.

Bueno, no hay más espacio, nos han quedado varias máquinas por ver de las que prometí el mes pasado, pero no importa porque os espero en la HISTORIA DE LOS VIDEOJUEGOS (3ª parte).

Diversión a raudales

Un aplauso para...

Por:

Javier Laynez

EPIC PINBALL





Algunos de los objetos que encontramos en el juego

EPIC PINBALL ver 2.0

¿Quién no se ha gastado alguna vez los cuartos, jugando en un recreativo a las máquinas de bolas?

Pues para aquellos pinbalmaníacos y los que aún no lo son, les damos la oportunidad de conocer este paquete, que en su interior trae un "Enigma".

Como tantas otras casas que sacan juegos de Pinball, ya que es uno de los juegos mas clásicos y que nunca pasan de moda, comentaremos este paquete, que trae ocho Pinballs diferentes de la casa Epic en su versión 2.0.

En la instalación, que requiere como mínimo un 386 , tarjeta VGA y disco duro, se puede adaptar a varios tipos de tarjeta, entre las que están Sound Blaster, SB Pro , Gravis Ultrasound, etc. de esta manera podemos disfrutar de los efectos de sonido y de la música que nos acompaña en los diferentes juegos.

Esta versión trae un archivo ".pif " para poder ejecutar el paquete bajo entorno Windows.

Entre otras opciones de juego nos permite elegir, como es normal de 1 a 4 jugadores, e independientemente en cada Pinball tres tipos de ángulos para los flippers, y elegir entre jugar 3 o 5 bolas.

Mientras se juega puedes cambiar la velocidad de la bola en el momento que desees (pulsando ALT-T), ademas tener la pantalla completa del juego, ocultando el marcador pulsando la tecla "Enter", dejandote introducir mas en el juego sin que por ello, en cada momento te va informando de los punto y de lo próximo que deberías hacer para conseguir mas puntos o un extra, dependiendo del pinball que te encuentres.

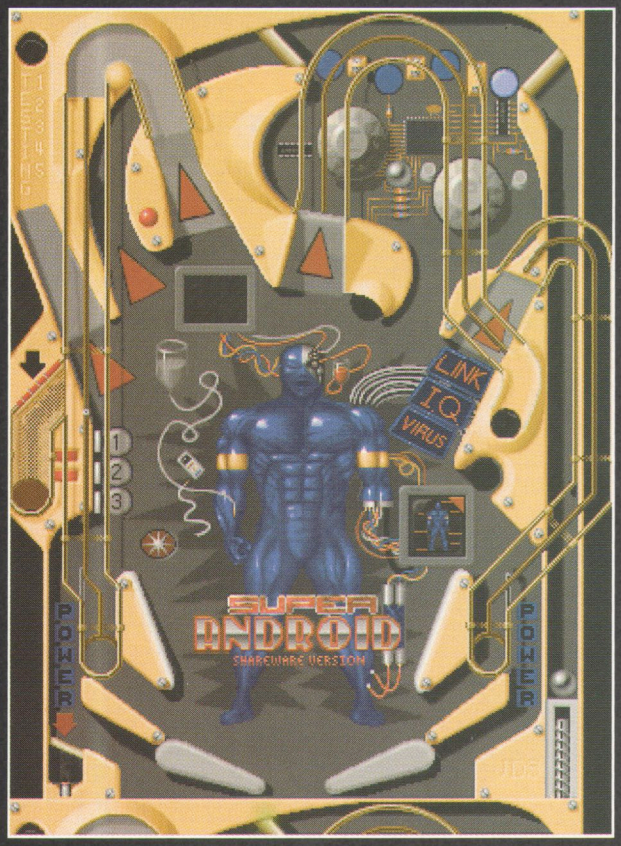
Una forma de conseguir mas puntos, es cuando mueves los flippers, va cambiando las posiciones de las luces de los pasillos superiores, y si eres habilidoso, puedes conseguir un buen tanteo cuando pase bola por el pasillo iluminado, pero eso depende de ti.

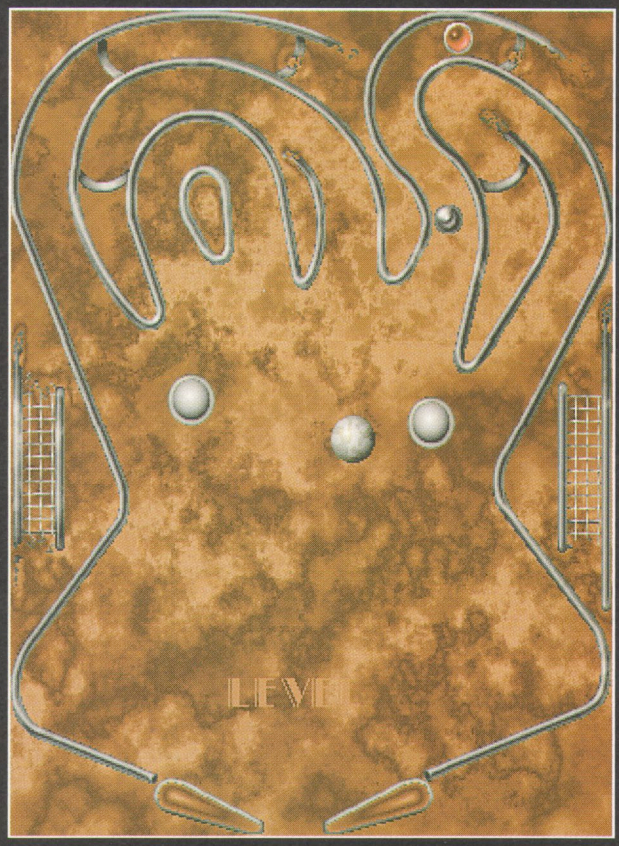
Puedes ver todo el pinball utilizando las teclas cursoras, con un scroll vertical.

En cada juego vienen explicadas las reglas de juego, pero aquí te daremos una pequeña orientación.

ANDROIDE:
(Android)

Así es como se llama nuestro primer juego. El objetivo es crear nuestro androide, instalando y activando todos los componentes físicos y ademas comprobar su





Android y Enigma, dos de los tableros que os podréis encontrar en el juego

funciones computerizadas.

OLLA DE ORO:
(Pot of Gold)

Este juego esta basado en la historia del duende del arcoiris que tenia una olla llena de monedas de oro. Así que lo que debemos conseguir es coger el máximo numero de monedas de oro en el final del Arcoiris avanzando a través de los colores, atrapando duendes y recolectando patas de conejo.

EXCALIBUR:

Con este Pinball nos trasladamos a la época del Rey Arturo y de sus Caballeros de la mesa redonda, la finalidad es conseguir sacar la espada Excalibur de su piedra, consiguiendo letra por letra la palabra completa.

GOLPES Y LLAMAS:
(Crash and Burn)

A los amantes de los coches, podrán disfrutar de de este pinball. La finalidad es como cualquier piloto de Formula Uno. Conducir la bola por las rampas y dar tres vueltas para ganar la carrera.

MAGIA:
(Magic)

Nos trasladamos al mundo de la ilusión y de las cartas. Con este juego debemos conseguir aumentar la puntuación dándole al ojo de buey rojo que tenemos en el centro para obtener el valor de los ases así consiguiendo desde 100 hasta 400 puntos, y dando a las paredes rebotadoras para ir cambiando de As.

PINBALL DE LA SELVA:
(Jungle Pinball)

Estamos frente a una jungla y no sabemos cuántos peligros nos encontraremos, pero tampoco cuantos premios.

PROFUNDIDADES DEL MAR:
(Deep Sea)

Debemos llevar nuestro barco a buen puerto y ademas conseguir tesoros y joyas en las profundidades del océano e intentar esquivar los peligros del mar.

ENIGMA:

Este es el no tan pequeño Enigma del paquete, a este le vamos a dedicar un interés especial ya que hay que ser un poco mas habilidoso, ya que no se espera qué saldrá obstruyendonos el paso -así su nombre-. Lo primero que te sorprende son los canales de la parte superior son cuatro y entra por un lado, la bola desaparece y sale por otro, estas rampas (de izquierda a derecha se le llama: Alfa, Beta, Delta y Gamma). Se puntúan de forma diferente y no prefijada, aumentando según subes de nivel. El juego es una pista totalmente libre de obstáculos hasta que empieza el juego, donde van apareciendo los diferentes objetos de forma aleatoria.

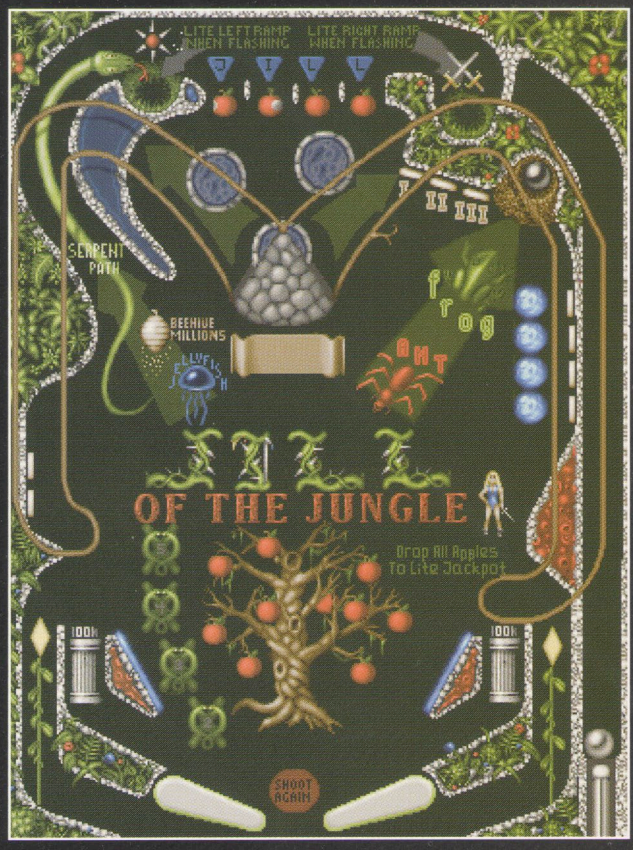
Son cuatro los niveles que debes pasar para terminar, este pinball a diferencia de otros su finalidad no es la de conseguir mas puntos sino pasar de fase consiguiendo un objetivo que es diferente en cada nivel lo que hay que combinar la habilidad y la estrategia y no la suerte, ya que existen elementos que en cada nivel aparecen más y en posiciones no fijas que nos ayudan o que nos impide terminar la partida.

Un ejemplo de objetos que aumentan nuestro marcador son:

Pergamino Místico = 1 Mill
Piruleta = 3 Mill
Oso = 3 Mill
Botón de Golpéame = 5 Mill

Un aplauso para...





Otros objetos como el pozo cautivo, que mantiene la bola un turno dando así otra bola extra en la próxima jugada, y unas zapatillas que te hacen pasar de nivel.

Y por ultimo objetos que te obstruyen el camino como:

Bola rebote: te obstruye el recorrido de la bola

Mina: Te hace estallar la bola, perdiendo así el turno.

Pozo repeledor : Como si de un imán que repele hace que la bola se varíe de dirección.

Traga bola: Si caen en este objeto la bola desaparece y pierdes el turno.

Resorte : despide la bola y no deja entrar en el canal.

La bola va cambiando de color según el numero de bola que sea pero al llegar a la ultima bola esta como camuflada con el fondo y es difícil seguirla.

NIVEL 1:

El objetivo es disparar a todas las gemas rojas que aparezcan en secuencia, y cuando aparezcan unas zapatillas rojas -es la puerta para pasar al nivel 2-, surgirán objetos que te impedirán el paso como las Bolas rebore o las Minas que harán explotar la bola, pero también puedes conseguir puntos extras con otros objetos.

NIVEL 2:

En este nivel debes destruir la barrera de ladrillos que aparecerá dos o tres veces. Te impedirán el paso en un momento dado una especie de pozo repeledor que no te dejara manejar bien la bola y dirigirla a su destino y también por supuesto las minas. Al final aparecerá una gema verde que te dejará pasar al nivel 3.

NIVEL 3:

Aquí lo que hay que destruir son pequeños rombos. La dificultad es que están en sitios muy difícil de dar y sobre todo eso, son pequeños, pero no todo queda aquí, ya que las minas que impiden acceder a los rombos están en lugares fáciles de dar. Aquí hallarás varios bonos especiales en varios objetos que encontraras. Para terminar la fase solo debes de darle a la gema que aparezca y por fin entraremos al nivel 4.

NIVEL 4:

Aqui tu objetivo son también las gemas. Debes destruir cuatro, pero cuidado, en el pozo repeledor te pueden hacer difíciles las jugadas si no eres habilidoso. Después destruye al osito teddy, y cuando el pozo cautivo aparezca esto te hará pasar de nivel.

En cada juego de Pinball aparece una música que te acompaña ademas de increíbles efectos de sonido, es un juego que toca muchos gustos ,ciencia ficción, mar, jungla , edad media, etc. y si no te identificas con ninguno de ellos...

"¡¡ Chico eres un Enigma !!"

Por:

Redacción de CJ

La bibliografía sobre programación y sobre todos aquellos temas relacionados con esta es enormemente abundante. Os ayudamos en esta sección a escojer aquellos manuales que tienen más calidad

El libro del mes

Aprender a programar correctamente el vídeo del PC es tarea difícil. No sólo porque el elemento hardware en cuestión presenta infinitos "no se como" sino porque la documentación que sobre él existe es escasa, controvertida y enrevesada. Un programador novel deberá, pues, armarse de paciencia y horas de estudio, para comenzar a desglosar los muchos aspectos oscuros que se esconden tras las siglas MDA, CGA, EGA, MCGA y VGA.

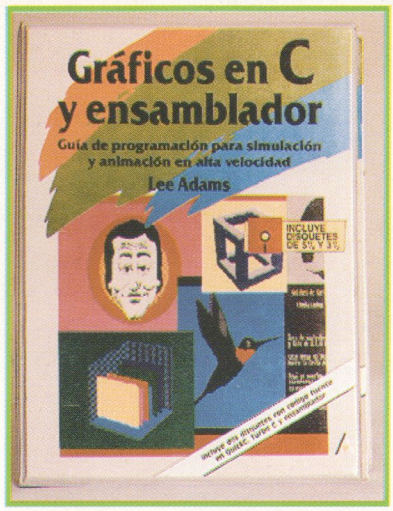
La materia es peliaguda. Resulta que cuando te conviertes en un buen programador, ya sea de bajo nivel o alto nivel, y realizas tu primer proyecto importante, la presentación en pantalla de tu obra deja mucho que desear. Es en ese momento cuando lamentas haber descuidado el aprendizaje de la utilización de toda la potencia gráfica que el PC es capaz de brindarte. ¿Qué hacer ahora?

Lo primero es empezar por una obra como la que hoy comentamos. Un libro compacto, sencillo pero a la vez profundo, lleno de ejemplos, explicaciones y referencias, que nos va a ayudar bastante a subir esos primeros escalones tan complicados en todo aprendizaje inicial.

EL LIBRO

Lee Adams, el autor, es la persona ideal para explicarnos que son y como se usan los gráficos. Quiero decir que del amplio y ramificado espectro de los temas informáticos, el Sr. Adams es un experto en su materia, como lo demuestra el hecho que su vida profesional ha rondado siempre alrededor del la animación, infografía, gráficos tridimensionales y efectos visuales computerizados, trabajando para la televisión y el cine.

La estructura del libro, de casi seiscientas páginas, es sencilla pero bien escogida. El autor comienza introduciéndonos en los conceptos básicos sobre la manera en que el PC



es capaz de de presentar la información en pantalla. "Gráficos frente a texto" lo llama. Seguidamente comenta qué es una tarjeta gráfica, cómo funciona, así como otros conceptos fundamentales referentes a la idiosincracia del hardware de vídeo de los ordenadores PC y compatibles.

Las explicaciones, tan sólo en un principio, rozán la sencillez más absoluta y poco a poco se transforman en verdaderos legados de la programación con ejemplos y programas completos que se incluyen en los dos disquetes de acompañamiento del libro. Los lenguajes utilizados son el QuickC, TurboC y Ensamblador.

Tras el análisis del Hardware, Adams se adentra en el software y las herramientas de programación. Nos hace una leve referencia a la estructura de un programa en C, explicándonos las tres ventajas que el considera fundamentales para la elección de este lenguaje como nuestra principal herramienta de ayuda a la hora de crear nuestras aplicaciones gráficas.

Es entonces cuando pasa a la explicación específica de los dos compiladores que utiliza a lo largo del libro; Microsoft QuickC y Borland TurboC. Para ello toca temas como el mapa de memoria del sistema, el entorno del MSDOS, las librerías, mensajes de error más frecuentes y la creación de ficheros ejecutables tipo .EXE, que si bien no tendrían porque estar en este libro, sí ayudan un poco a todos aquellos que no sepan todavía programar o simplemente para refrescar ideas y conceptos.

Es a partir de este momento cuando el autor se adentra en los temas con sustancia del libro. Comienza con el diseño gráfico y los conceptos esenciales del mismo; salida por pantalla, selección de los colores a utilizar, dibujo sobre la pantalla, gestión de los gráficos de fondo, utilización de las ventanas y finalmente temas avanzados como las bases de datos gráficas, grabación en disco de las imágenes creadas, etc.

Un apartado que merece especial atención es el de los gráficos tridimensionales, materia ésta que se nota apasiona al autor. Multitud de ejemplos prácticos se incluyen es esta sección del libro, acompañados de explicaciones que podríamos tachar de exhaustivas.

La animación de sprites, que el llama animación de fotogramas, es tema que más me gustó del libro, ya que es la que a los programadores de juegos nos sirve más directamante. Aquí encontramos todo tipos de trucos y programas sobre la animación arcade, gestión de colisiones y rebotes, con ejemplos de sprites moviéndose por la pantalla y bolas que rebotan en los bordes de la misma. Estos temas los podemos más tarde acoplar en nuestros propios programas de forma sencilla.

Ficha técnica

Título: Gráficos en C y Ensamblad
Editorial: Anaya Multime
Pag: 592
Autor: Lee Adams

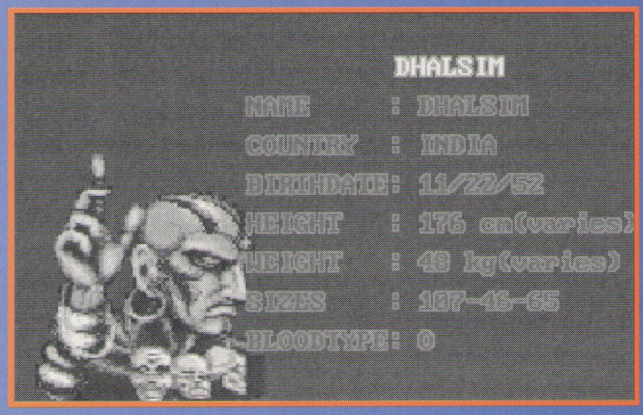
Por Raul Reales

Street Fighter Turbo

version 2.0

Aunque este juego no se encuentra disponible comercialmente, vamos a citar lo que es la versión 2.0 de este clásico en las recreativas en su conversión para el PC: Street Fighter II Turbo.





LA MONUMENTAL SAGA DE STREET FIGHTER II

En las recreativas, hace ya tiempo, apareció un video juego que ocasionó un boom en los arcades de lucha. Se llamaba Street Fighter II y hacía que no parasen de depositar monedas para poder jugar.

El toque maestro era dado por CAPCOM, que ya era una compañía con una notable experiencia en juegos arcade de todo tipo.

Había un total de ocho luchadores seleccionables, sumados a otros cuatro que aparecían cuando habías derrotado a los que figuraban al principio.

Cada personaje tenía una historia, unos motivos, una preparación física y unos golpes designados como especiales, donde se conseguía el acceso a ellos mediante una habilidosa combinación de movimientos con el joystick.



Las versiones del juego en los salones recreativos fueron las siguientes:

- Street Fighter II.- La primera "entrega", donde sólo estaban disponibles ocho luchadores. La velocidad era media-alta. Cada uno de ellos disponían de dos o más golpes especiales. El argumento era mas bien simple: derrotar a todos los luchadores usando las mejores técnicas y los mejores golpes para llegar el conseguir el título de campeón del mundo. Cada uno de los guerreros tenía un final distinto, lo cual hacía más interesante terminar el juego con varios luchadores.

- Street Fighter II Champion Edition.- Debido al gran éxito conseguido con el primer Street Fighter II, CAPCOM había conseguido la recreativa del año, así que se lanzó manos a la obra y trabajó duro para sacar la continuación. A esta la llamó Champion Editon. Aunque el argumento no cambiaba, se habían incluido mejoras, tales como incremento de la velocidad, cambio de los decorados, mayor definición, poder elegir ambos al mismo luchador -lo cual hacía posible más combinaciones de combates-, etc. Como punto culminante, se incluyen los cuatro guerreros que antes no aparecían, pasando así a tener doce para elegir. Una pasada.

- Street Fighter II Champion Edition Turbo.- La versión realizada para los verdaderos maestros que manejaban a la perfección las anteriores continuaciones. Se habían realizado modificaciones de los golpes especiales -más variedad-, habían duplicado la velocidad, se podía cambiar de luchador en cualquier momento de la partida. En fin, poco se puede decir de esta versión que no haya dicho, lo cual quería decir que la revolución de Street Fighter parecía ir acabando para quedar como historia en el mundo de las recreativas, pero... ¡Sorpresa! Aparece...:

- SUPER Street Fighter II.- Lo más brutal, lo mejor de Street Fighter II, aparece con la inclusión de ¡cuatro! guerreros nuevos, unos decorados reformados con una calidad tremenda, una intro que quita el aliento, vuelta al esquema clásico de Street Fighter II Champion Edition, efectos de sonido desarrollados exclusivamente para este desenlace -por ahora-, e innumerables mejoras, que seguramente harían este artículo muy extenso. (Dejemos para eso a las revistas de consolas y máquinas recreativas).

LAS VERSIONES PC

Pasó mucho tiempo para que apareciese una versión oficial, producida por CAPCOM, que finalmente nos dejó un poco desilusionados. No es que el juego para PC fuese malo, sino simplemente versiones de Super Nintendo o Mega Drive les superaban en algunos aspectos.

Como eso no bastó para satisfacer a los numerosos seguidores de este mega-juego de lucha, personas ajenas a la compañía CAPCOM crearon versiones no-oficiales, que llegaron a ser mejores en aspectos como el sonido o rapidez de movimiento.

Una de estas versiones japonesas la pude ver en casa de un amigo, con sus melodías, sonidos y personajes, que eran bastante aceptables. Por supuesto, al hablar con él, me dijo que era una de las mencionadas versiones no-oficiales.

Esta versión es la 2.0 realizada por un grupo de personas que ha puesto todo su empeño en realizar por medios propios la mejor versión,

para dejar satisfechos a aquellos que pedían más y más.

Antes de pasar al siguiente apartado, decir que el juego comentado no está destinado a ser realizado con fines profesionales. Tan sólo es comentado para que veáis lo que podéis llegar a hacer vosotros mismos si seguís los cursos de programación y le añadís toda vuestra voluntad. Todo esto es avisado, para que no os pongáis a buscar como locos entre catálogos de revistas para conseguirlo.

CONFIGURACIÓN

Una vez en el Disco Duro, ejecutamos el programa de configuración para ver sus opciones.

Vamos a ver: Teclas del jugador uno, del dos, configuración automática y manual con respecto a velocidad en MHz -frames, tarjeta de sonido- cuyos efectos curiosamente se reproducen por ficheros .VOC pudiendo ser modificados y editados-, grabar...En resumidas cuentas, lo fundamental para este tipo de juegos se encuentra aquí.

EL JUEGO

Esta versión no-oficial dispone de más de seis tipos de configuraciones, donde aparece una sorpresa: se permiten elegir dos luchadores que no pertenecen a Street Fighter II. Estos son de otra conocida recreativa. El enigma lo podéis resolver -si os apetece- echándole un vistazo a las pantallas que acompañan a este artículo. Después del logotipo de presentación -un pelín cutre-, aparecen las tres alternativas: modo un jugador, dos jugadores o pasar al menú de opciones donde se pueden configurar tiempo, número de combates, hacer un test de los efectos de sonido, y poco más. Si tu elección es una de las dos primeras, pasarás a entrar en el archiconocido menú de selección de personajes -versión turbo-, que puede variar dependiendo del tipo de configuración (aparte de esa, se puede elegir la clásica de ocho luchadores).

Una vez en el combate, se pueden usar los seis tipos de golpes -tres tipos de patadas y tres de puñetazos-, con lo que dispones de una amplia gama a elegir para poder así machacar a tu oponente. Algo que quiero señalar es que algunos dirán que la recreativa que ellos juegan de Street Fighter II Turbo, tan solo consta de tres botones y el mando. Lo que ocurre es que los propietarios, para ahorrarse el coste adicional de la máquina original, sólo compran la placa y la montan en una máquina "de las corrientes" -si la podemos llamar así- con los tres botones, desaprovechando de esta manera, cinco golpes. Si a esto le añadimos que sólo conectasen los golpes menos potentes, significa que nunca podremos aprovechar al cien por cien las posibilidades que nos ofrece el juego. Una lástima.

Volviendo de nuevo al combate, decir que podréis realizar todo lo que habeis visto en la recreativa de vuestro salón.

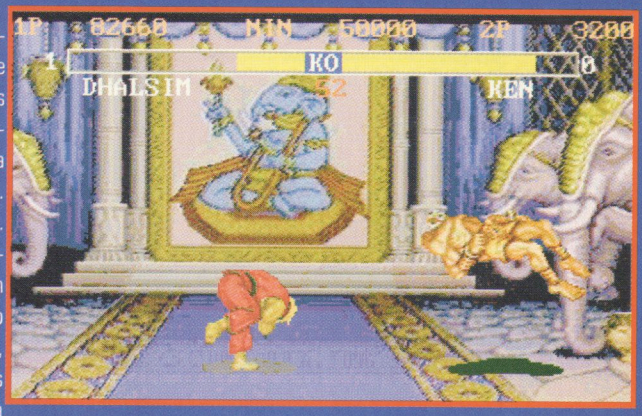
LA PROGRAMACIÓN

Teniendo en cuenta que está realizado por un pequeño grupo de personas, y no por un gran equipo de programación profesional, el resultado es bastante satisfactorio -lo podéis apreciar en las pantallas-. Se aprecian unos movimientos bien conseguidos, unos efectos de sonido satisfactorios, y un scroll de pantalla que no presenta ninguna incomodidad para la vista.

El tirón de orejas es en el aspecto de la música. ¡No hay!, y eso es algo necesario para dar vida a el software de entretenimiento que sale al mercado. Y si no estáis de acuerdo, pensad en las bandas sonoras de Alone in the Dark I y II, Oscar en su versión CD, y otras muchas.

De nuevo el Lenguaje Ensamblador -imprescindible en este tipo de juegos- está presente. Como ya se ha dicho, tened en cuenta que si llegáis a alcanzar unos conocimientos elevados, podréis mejorar incluso el juego que se comenta en éstas páginas.

LAS DIFERENTES VERSIONES DE SF2IBM







En las pantallas aparece muchas veces la especificación de la versión. Aclarémoslo. En un principio, la versión fue -como todas- la 1.0, pero después de que una gran cantidad de personas informase sobre los errores -bugs- y lo que le hacía falta para ser un juego con mayúsculas, los programadores siguieron manos a la obra para perfeccionar lo que habían realizado. Mejoras y más mejoras, tras varios lanzamientos más, hicieron aparecer la versión 2.0, comentada en estas páginas. Pero debido a los pequeños bugs y a la demanda de personajes,

los programadores seguirán en el empeño de conseguir llegar al SUPER Street Fighter II, que llegará con el tiempo. Si sabéis ser pacientes, quizás logremos ver comentada la última versión, a ver que tal les ha salido.

PALABRAS FINALES

No vamos a dar una recomendación de si el juego es bueno o no. El artículo se ha elaborado con la intención de dar a conocer los juegos que se pueden realizar en plan un poco casero, dando una calidad que puede superar incluso a la versión oficial, que está en el mercado. Si de verdad queréis llegar a ser auténticos programadores, intentadlo con juegos originales, de ideas propias, que sean graciosos, divertidos, o auténticos simuladores, o cualquier cosa que se denomine Software de Entretenimiento Original.

Como solución al enigma propuesto, deciros que los personajes son Andy Bogard y Terry Bogard -fácil, ¿verdad?-.

Aprendemos la manera de usar las órdenes WRITE y WRITELN y muchas más cosas

Por:
Alberto García

Lenguaje PASCAL

3ª PARTE

Procedimientos de entrada/salida

El mes pasado estuvimos viendo algunas de las funciones básicas de TPascal. También vimos de pasada el procedimiento READLN, el cual usábamos para leer el valor de una variable desde el teclado. Analizaremos en este capítulo del curso de Pascal los cuatro procedimientos básicos de entrada/salida (E/S) en modo texto, de los cuales ya conocemos a dos.

WRITE, WRITELN, READ, READLN

Comenzaremos hablando de WRITE y WRITELN. A WRITELN ya lo conocemos, su función es la de mostrar en pantalla una línea de texto, el valor de una variable, o cualquier cosa que queramos. Teclea el programa1 y ejecútalo:

```
programa1;
begin
    writeln ('HOLA');
    writeln ('SOY');
    writeln ('UN');
    writeln ('ORDENADOR');
end.
```

(ya deberías saber lo que hace). Verás aparecer en pantalla las palabras HOLA SOY UN ORDENADOR, cada una en una línea, pero seguramente algún avispado lector se estará preguntando de que forma podemos hacer que aparezcan las palabras en la misma línea. Una solución muy sencilla sería simplemente escribir todas las palabras en la misma instrucción WRITELN, de ésta forma:

```
WRITELN ('HOLA SOY UN ORDENADOR');
```

Sin embargo no todos los problemas con los que nos encontraremos serán tan fáciles de resolver, así que buscaremos otra forma. Como seguramente estás imaginando la respuesta está en el otro procedimiento de que disponemos para la escritura; hablamos de WRITE. En efecto, WRITE realiza exactamente la misma función que WRITELN, con la salvedad de que no salta a la línea siguiente una vez que ha escrito en pantalla. Ahora modifica el programa1, sustituyendo WRITELN por WRITE, y ejecútalo, verás aparecer en pantalla (no olvides pulsar ALT+F5 para verlo) algo como esto:

```
HOLASOYUNORDENADOR
```

¿Donde están los espacios? Bien, este error intencionado, lo he cometido para que veas que incluso un lenguaje potente como TPascal, es incapaz de leerte el pensamiento, TPascal ha echo LITERALMENTE lo que le decíamos en el programa, y en el no decíamos nada sobre espacios. Para arreglarlo, simplemente añade un espacio después de cada palabra a imprimir (siempre dentro de las comillas) en las instrucciones WRITE. Para finalizar con WRITE, indicar sólo que posee la misma sintaxis que WRITELN (incluidos los modificadores de justificación que ya vimos), Así que es decisión tuya el usar uno u otro dependiendo de lo que quieras hacer.

Veamos ahora las instrucciones que nos permitirán la entrada de datos desde el teclado. Estos datos se almacenan en variables, que hemos declarado previamente, y que ni decir tiene que el tipo del dato introducido por teclado, TIENE que coincidir con el tipo de variable declarado. Los procedimientos encargados de la entrada de datos son READ y READLN. Ambas son virtualmente iguales, sólo hay una pequeña diferencia que veremos ahora, aunque seguramente podrás imaginártela. En el capítulo anterior vimos como introducir por teclado un dato, pues bien, esta es la sintaxis más general de este procedimiento, sin embargo, estas instrucciones nos permiten introducir más de un dato, es decir,

```
READLN v1;
READLN v2;
READLN v3;
```

es igual que:

```
READLN (v1,v2,v3);
```

Como ves el procedimiento READ/READLN nos permite la entrada múltiple de datos, y he aquí la diferencia básica entre READ y READLN, con READ, la entrada de varios datos se realiza introduciendo un espacio entre cada dato y pulsando ENTER al final (tras introducir todos los datos), sin embargo, READLN, nos permite introducir los datos uno detrás de otro pulsando ENTER des-

pués de introducir cada uno de ellos por separado. Para aclarar conceptos, introduce el programa2 y ejecútalo.

Verás más claro como funcionan estos procedimientos. Nota que al imprimir los resultados, no hemos hecho como siempre, es decir almacenar el resultado en una variable y luego mostrarla. A cambio, hemos hecho que el procedimiento WRITE/WRITELN calcule y muestre el resultado. El efecto es el mismo, y ahorras una variable, aunque posiblemente el programa pierde en claridad.

```
PROGRAMA NÚMERO 2
program programa2;
var
    v1,v2,v3 : integer;
begin
writeln ('Entra tres números, pulsando ENTER despues de cada uno:');
readln (v1,v2,v3);
writeln ('La suma es:',v1+v2+v3);
write ('Entra tres números separados por espacios y ENTER al final:');
read (v1,v2,v3);
writeln ('La suma es:',v1+v2+v3);
end.
```

LA PANTALLA EN MODO TEXTO

Hemos visto como funcionan los procedimientos básicos para la Entrada/Salida (E/S) en modo texto, pero no nos basta esto para poder hacer programas con una presentación agradable, ya que aunque se nos permite algo de control (saltar a la siguiente línea o no saltar) este no es suficiente para crear programas agradables. Cuando usamos un programa, este ha de darnos la posibilidad de interactuar con él, dándonos la posibilidad de introducir datos y consultarlos, entre otras muchas cosas. La parte del programa que nos permite hacer esto se llama INTERFAZ CON EL USUARIO, y aunque no lo creas es una de las partes mas importantes del programa, incluso en ocasiones, mas importante que el núcleo de este. El diseño de interfases de usuario no es una ciencia exacta, es mas un arte que una técnica. EL interfaz puede ser gráfico y en modo texto. Un ejemplo de interfaz gráfico es Windows (mira CJ3 para mas información) y como ejemplo de interfaz en modo texto, te pongo como ejemplo al mismo TPascal. No es necesario que un interfaz de usuario sea tan complejo como estos que he citado, de hecho veremos un ejemplo mucho más simple en el programa3, pero no lo teclees aun, ya que en él usamos instrucciones de TPascal que aun no hemos visto.

LO PRIMERO QUE HA DE HACER UN INTERFAZ DEL USUARIO ES LIMPIAR EL CONTENIDO PREVIO DE LA PANTALLA

Lo primero que ha de hacer un interfaz de usuario es limpiar o borrar la pantalla de otras cosas que hubieran en ella. Como vimos en el primer capitulo, el procedimiento que se encargaba de borrar la pantalla en modo texto era CLRSCR (que proviene de CLeaR SCReen). Sin embargo, Este procedimiento está incluido en una UNIDAD (mira el capitulo1), concretamente la unidad CRT, por lo tanto hemos de declararla, y esto se hace con la palabra reservada USES, mira el programa3 para comprenderlo mejor, y recuerda que no se puede hacer esta declaración en cualquier parte del programa, sino que ha de hacerse en el lugar que tiene asignado para ello, si no lo recuerdas, mira el capitulo1 en el que se describe la estructura de un programa en Pascal. Una vez que hemos declarado la unidad CRT, ya podemos usar el procedimiento CLRSCR, pero no sólo este, sino un montón de procedimientos y funciones que están almacenados en ésta unidad, pero los iremos viendo más adelante.

```
PROGRAMA 3
program programa3;
uses crt;
var
base,altura,resultado : real;
begin
    clrscr;
    gotoxy(20,10);
    write('BASE:');
    gotoxy(20,12);
    write('ALTURA:');
    gotoxy(26,10);
    read(base);
    gotoxy(28,12);
    read(altura);
   resultado := (base * altura) / 2;
    gotoxy(20,15);
  writeln('El resultado es: ',resultado:1:4);
end.
```

UN PASO MAS

Con esto estamos en condiciones de hacer un programa capaz de interactuar con nosotros de una manera apropiada, pero aún podemos mejorarlo algo. Cuando usamos algún procedimiento de E/S de los que vimos antes, estos actúan siempre al principio de la línea. Pero ¿a que quedaría mejor si pudiéramos centrar el texto en la pantalla? Bien, pues esto podemos hacerlo gracias a un procedimiento llagado GOTOXY, y cuya sintaxis es:

GOTOXY (X,Y);

Este procedimiento coloca el cursor en la posición X,Y de la pantalla que nosotros queramos, con lo cual, la siguiente instrucción de E/S que hagamos tendrá lugar en la posición que hemos indicado con GOTOXY. En la figura1 aparece el formato que tiene la pantalla en modo texto, que es de 80x25, o sea, con 25 líneas y 80 columnas. Ahora estamos en condiciones de comprender el programa3, así que introdúcelo y ejecutalo. Es el mismo programa que vimos en el primer capitulo para calcular áreas de triángulos, pero como ves ahora queda mas bonito.

¡MAS DIFICIL TODAVIA!

Ya podemos hacer programas con un aspecto más decente, pero seguramente echaras de menos algo. Es el color, pero antes de ver los procedimientos que son capaces de manejarlos, veremos algo más sobre la pantalla en modo texto del PC. La pantalla como dijimos antes tiene una dimensión de 80x25, y como tal es guardada en memoria. Si recuerdas el primer capitulo en el que comparábamos la memoria con casilleros o buzones, podrás hacerte una idea. Imagina un casillero con 80 casillas por línea y con 25 líneas. Ahora cada casillero representa una posición en

la pantalla (como el juego de los barquitos), es decir, si ponemos una A en la primera casilla, es como si la pusiéramos en la primera posición de la pantalla. Pero en cada casilla sólo cabe una letra (carácter), por lo tanto no caben los colores de la letra, ¿como lo solucionamos? Bien, la manera más fácil es meter en otro casillero exactamente igual los colores. De ésta forma, si metemos el color verde en la primera casilla del segundo casillero, la letra A aparecerá de color verde. En éstas casillas no sólo podemos introducir los colores, sino que además podemos indicar si la letra está el alta intensidad, en baja intensidad, parpadeando, en video inverso, etc... (seguramente éstos términos no te sonaran, pero no desesperes, lo veremos seguidamente). Los datos que introducimos en éste segundo casillero son genéricamente llamados atributos de los caracteres, o simplemente atributos. Lo que hemos visto, es a grosso modo la forma en que el PC maneja éstos datos internamente, pero por ahora, nosotros no lo haremos así, ya que TPascal posee procedimientos que nos harán la vida mas fácil en éste sentido. Los primeros que vamos a ver se encargan del atributo de intensidad, estos procedimientos son HIGHVIDEO y LOWVIDEO. Estos procedimientos no llevan parámetros, y son los encargados de controlar la luminosidad de los caracteres. Como adivinaras HIGHVIDEO hace que los caracteres aparezcan en alta intensidad y LOWVIDEO hace exactamente lo contrario. Introduce el programa4 para comprobar como lo hacen. Esto es útil para resaltar datos o frases dentro de la pantalla. Toma nota del procedimiento que aparece en el programa4 al final, hablamos de NORMVIDEO, cuyo cometido es el de restablecer el modo de video a sus valores iniciales, es decir a los que estaba al principio de la ejecución del programa.

Y AHORA EN COLORES

También podemos colocar el texto en colores, esto lo hacemos mediante dos procedimientos: TEXTBACKGROUND y TEXTCOLOR. Sus respectivos formatos son:

TEXTBACKGROUND (color); TEXTCOLOR (color);

o sea colocando el color en el interior de los paréntesis. El color ha de introducirse mediante un número, que representa a un color. He aquí la tabla de colores y sus valores:

```
Negro (Black)                -> 0
Azul (Blue)                  -> 1
Verde (Green)                -> 2
Celeste (Cyan)               -> 3
Rojo (Red)                   -> 4
Magenta (Magenta)            -> 5
Marron (Brown)               -> 6
Gris Claro (LightGray)       -> 7
Gris Oscuro (DarkGray)       -> 8
Azul Claro (LightBlue)       -> 9
Verde Claro (LightGreen)     -> 10
Celeste Claro (LightCyan)    -> 11
Rojo Claro (LightRed)        -> 12
Magenta Claro                -> 13
Amarillo (Yellow)            -> 14
Blanco (White)               -> 15
```

Ha de hacerse notar, que el color puede especificarse mediante su nombre en inglés o mediante su correspondiente número. Esto es debido a que TPascal ya tiene definidas las palabras en inglés para los colores como constantes que contienen el número de color. El primero de los procedimientos (TEXTBACKGROUND) señala el color del fondo para los caracteres, y TEXTCOLOR señala el color del carácter en sí. Lo mejor es que introduzcas el programa5 y lo compruebes por ti mismo. Ahora ya podemos hacer un interfaz con el usuario más decente, pero podríamos mejorarlo aún más. Seguramente has manejado otros programas en los que el ordenador avisa con una señal acústica, por ejemplo cuando ha terminado la ejecución del programa o cuando el ordenador pide algún dato por teclado. Las posibilidades y usos de estas señales son muchos. Por suerte, TPascal es capaz de generar estas señales de una manera muy simple gracias al procedimiento SOUND. El formato de éste procedimiento es:

SOUND (Hz);

Donde Hz es la frecuencia del sonido en Hercios. De hecho, es posible hacer sonar música por el SPEAKER del PC gracias a ésta instrucción. He aquí la lista de las notas principales y su correspondiente frecuencia:

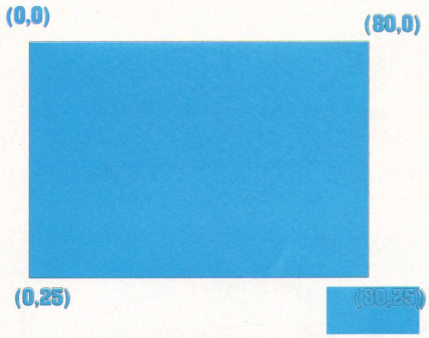
[INSERTAR TEXTO 3 AQUI]

Para moverse una octava arriba o abajo, ha de tenerse en cuenta que la frecuencia es el doble de la misma nota de la octava anterior, es decir, si queremos el DO (C) de la octava siguiente, solo tenemos que multiplicar por dos su frecuencia, y si queremos bajar de octava, sólo dividir entre dos. Para desconectar el sonido, se usa el procedimiento NOSOUND, que calla al SPEAKER del PC. Más adelante, cuando tengamos algo más de conocimientos, haremos un programa que simulará a un órgano muy sencillo para poder tocar música con nuestro PC, pero eso será mas adelante y nos servirá para ilustrar algunas de las estructuras de control del TPascal.

Otra técnica que vamos a ver es la de la lectura de teclas. Seguramente has visto mas de una vez el mensaje PULSE UNA TECLA PARA CONTINUAR, o algo parecido. Bien nosotros, con nuestros conocimientos actuales, podríamos usar el procedimiento READ y esperar a que se pulsara ENTER, pero esto no es demasiado elegante, así que echaremos mano de una nueva función llamada READKEY, cuyo formato es:

tecla := READKEY;

Como observaras, READKEY no es un procedimiento, sino una función, ya que devuelve un valor que es almacenado en la variable a la que queramos asignarselo, en este caso TECLA. Exactamente, esta función, detiene la ejecución del programa hasta que una tecla es pulsada, y la tecla pulsada se almacena en la variable. Ahora mira el programa6, que como observaras es algo mas completo que el anterior. Observa que asignamos la tecla pulsada a una variable, sin embargo no hacemos uso de ella, es decir, malgastamos una variable. Por ahora lo haremos así, y más adelante veremos una forma más elegante de hacerlo. Encontramos también una instrucción que no conocemos, se trata de DELAY, que está descrita más adelante en éste mismo capitulo. Por último, observa en la declaración de variables, de que tipo es la variable TECLA. La hemos declarado como tipo CHAR, ¿y que es eso? Bien, eso lo dejaremos para el siguiente numero, pero te adelanto que se trata de una variable de tipo CARACTER, es decir, que es capaz de albergar un carácter alfanumérico, como una letra o cualquier otra cosa, pero atención, solo un carácter a la vez.



Formato de la pantalla en modo texto

OTRAS ORDENES UTILES

Como ves las construcción de interfases de usuario no es una ciencia exacta. De hecho nosotros aún poseemos recursos muy limitados. Sin embargo ya somos capaces de hacer que nuestro programa tenga una correcta comunicación con el usuario. Siente libre para diseñar tus propios interfases a tu gusto y estilo.

Pasamos ahora a ver algunas instrucciones útiles contenidas también en la unidad CRT. La primera que vamos a ver es DELLINE, cuya misión es borrar la línea en la que está actualmente el cursor. Recuerda que este se posiciona mediante el procedimiento GOTOXY. Es decir, si quieres borrar la línea 5, te sitúas primero en ella con GOTOXY(1,5) y seguidamente la borras con DELLINE. Este procedimiento tiene múltiples funciones, pero las mas clásica es la de borrar mensajes de error o de advertencia, por ejemplo, imagina que haces un programa que te pregunta el sexo de una persona, y solo tiene como respuestas validas masculino o femenino, sin embargo el que introduce los datos escribe otra cosa por error. En este momento nuestro programa detectaría el error (ya veremos como) e informaría de él por pantalla durante un período de tiempo. Tras esto, con la instrucción DELLINE borraríamos la línea y volveríamos a preguntar de nuevo. Un derivado de este procedimiento es CLREOF, que es otro procedimiento similar a DELLINE, pero con la diferencia de que sólo borra el trozo de línea que va desde donde está el cursor hasta el final de la línea. Observa que ninguno de éstos procedimientos tienen parámetros. Otro procedimiento que puede ser de utilidad es TEXTMODE, el cual conmuta el modo texto. Para más información sobre esto consulta el manual del DOS y en especial la orden MODE del DOS (el sistema operativo). Las opciones válidas son:

[INSERTAR TEXTO 4 AQUI]

De esta forma si quisiéramos poner la pantalla en modo texto de 40 columnas solo habríamos de hacer TEXTMODE(co40);. Veremos ahora otro par de funciones útiles, estas son WEREX y WEREY, las cuales devuelven en una variable de tipo INTEGER la posición X e Y del cursor respectivamente. Es decir, si hacemos GOTOXY(5,10); y usamos estas funciones, WEREX nos devolverá 5 y WEREY nos devolverá 10. el formato general de éstas funciones es:

x := WEREX; y := WEREY;

Otro procedimiento útil es DELAY, el cual para la ejecución del programa durante un tiempo determinado. Su sintaxis es:

DELAY (tiempo);

Teniendo en cuenta que el tiempo va expresado en milisegundos, por ejemplo DELAY(100); detiene la ejecución durante 100 milisegundos para seguir después por donde iba. Se usa éste procedimiento para dar tiempo, por ejemplo, a que el usuario del programa pueda leer algún mensaje, o simplemente para hacer que un programa haga algo más lentamente (sobre todo en ordenadores muy rápidos). Recuerda el ejemplo que poníamos antes del mensaje de error. Con DELAY podríamos hacer que nos diera tiempo a leerlo antes de borrarlo con DELLINE. Otra interesante instrucción es INSLINE, la cual inserta una línea en pantalla. ¿que qué es insertar? bien, supón que queremos meter una frase o línea de advertencia en la línea 5 de la pantalla, pero esa línea ya está ocupada, y además no podemos borrarla porque nos es útil. INSLINE, lo que hace es mover todas las líneas a partir de la 5ª una posición hacia abajo (observa que la última se pierde) dejando así libre la línea 5. Este procedimiento no tiene parámetros, así que inserta la línea en la actual posición del cursor. Otro procedimiento útil, es el procedimiento WINDOW, que significa ventana en español. pero ¿que es una ventana? Bien, seguramente has visto Windows, y sabes que es un entorno de trabajo basado en ventanas. En una ventana se puede mostrar información, pedir datos, y cualquier cosa que se te ocurra. El procedimiento WINDOW de TPascal, sirve para crear ventanas en modo texto. Si te has fijado, al compilar TPascal un programa, aparece un recuadro en la parte central de la pantalla mostrando información sobre el proceso de compilación. pues bien, esto es una ventana en modo texto. La sintaxis del procedimiento WINDOW, es:

WINDOW (x1,y1,x2,y2);

Siendo: X1-Y1: coordenadas (x,y) de la esquina superior izquierda. X2,Y2: coordenadas (x,y) de la esquina inferior derecha.

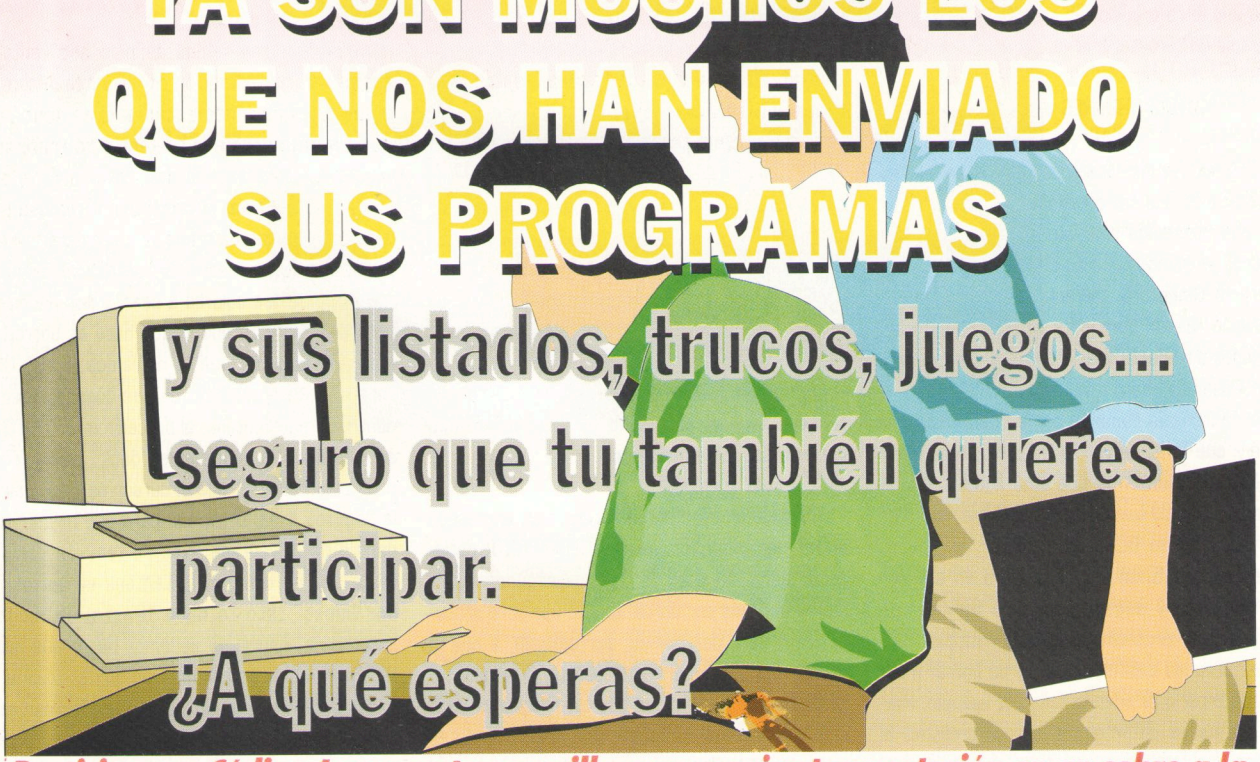
De esta forma hemos definido una ventana. Ahora cualquier operación de las antes descritas, o sea, las pertenecientes a la unidad CRT, tendrán efecto sobre la ventana y no sobre toda la pantalla. Para tenerlo más claro, mira el programa7. Si observas el programa, veras que para hacer que la ventana tenga un color determinado, primero hay que crearla con WINDOW, seguidamente definir los colores a usar, y por fin inicializarla con CLRSCR, que lo que hace es borrarla y poner los colores que le habíamos dicho. Otra cosa a tener en cuenta, es que cuando estemos dentro de una ventana, el procedimiento GOTOXY, se referirá a la ventana, es decir, que si en el programa hacemos que el cursor se posicione en la segunda línea, el cursor hará eso mismo pero dentro de la ventana. Nótese también como el texto que no cabe en la ventana es partido en dos líneas. Esto to hace TPascal de manera automática. Ahora, ya podemos hacer que nuestros programas se comuniquen con el usuario de forma decente en modo texto. Cuando veamos la forma en que TPascal trabaja con gráficos, podremos hacer interfaces más eficaces en modo gráfico, al estilo OS/2 y Windows. Mientras tanto, practica todo lo que puedas... y hasta el próximo número.****

EN

CODIGO JUEGO

TE ESTAMOS ESPERANDO

YA SON MUCHOS LOS QUE NOS HAN ENVIADO SUS PROGRAMAS



y sus listados, trucos, juegos...
seguro que tu también quieres participar.
¿A qué esperas?

Participar en Código Juego es tan sencillo como enviar tu aportación en un sobre a la dirección que detallamos abajo. Si tu programa es elegido contactaremos contigo.

Código Juego
Ajatel S.L.
Apdo. 114, 11520, Rota
Cádiz

Ajatel S.L.

2ª PARTE

CURSO DE PROGRAMACION WINDOWS

Windows de red

Por:
Leopoldo Fernández

Windows para Trabajo en Grupo, es un sistema operativo en el que se han combinado dos elementos:

1.- Plenas posibilidades y funcionamiento de trabajo en red.
2.- El sistema operativo Windows.

Estos dos elementos facilitan el uso compartido de la información y de los recursos informáticos, a la vez que agiliza y mejora la productividad del trabajo en grupo. Tanto como si se dispone de una red local como si no, el sistema operativo Windows para Trabajo en Grupo ofrece las siguientes características:

1.- Trabajar en grupo mediante una red fácil de poner en marcha, fácil eprender y de usar.
2.- Es la primera plataforma Windows para trabajar en grupo.
3.- Si ya se dispone en la actualidad de una red local, Windows para Trabajo en Grupo se añade a dicha red, mejorando notablemente sus posibilidades y aminorando los problemas que se pudieran presentar en la actualidad.
4.- Permite aprovechar la potencia del trabajo en red de 32 Bits.

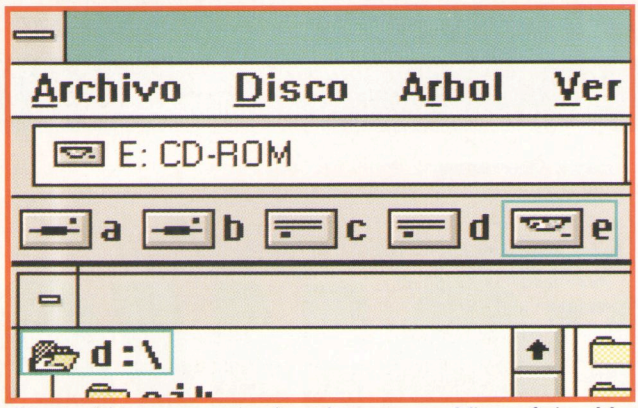
Windows para trabajo en grupo, juega un papel clave con sus capacidades de comunicación incorporadas. Cualquier grupo de trabajo que disponga de varios PCs puede rápida y fácilmente beneficiarse de las ventajas de la conexión en red. La posibilidad de compartir archivos e impresoras, junto con las ventajas del correo electrónico y el planificador de grupo, conforman una plataforma completa para esta modalidad de trabajo. Toda la familia de sistemas operativos Windows comparte la misma interface de usuario y el mismo entorno de programación, lo que permite mezclar distintas plataformas formando una solución global perfectamente adecuada a las necesidades del proceso.

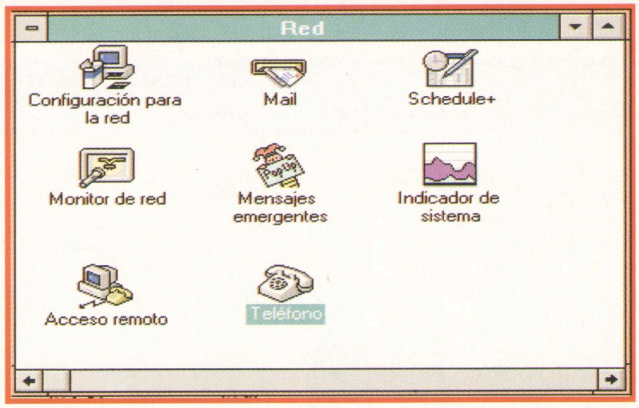
EN EL WINDOWS 3.11 SE HAN COMBINADO DOS ELEMENTOS: LAS POSIBILIDADES DE LA RED LOCAL Y LA POTENCIA DEL ENTORNO OPERATIVO WINDOWS.

EL PORQUE DE WINDOWS PARA TRABAJO EN GRUPO

Se observan distintos aspectos relevantes en el panorama actual de la informática en red:

- La mayor parte de las redes actuales son difíciles de instalar, de aprender y de usar.
- El traslado de aplicaciones propias de los ordenadores mayores hacia el sistema en red está experimentando un gran auge, pero los clientes no estan seguros de cómo hacer este cambio protegiendo sus actuales inversiones en redes.
- La informática de grupos está en sus





albores, si bien se esta desplazando un gran interés hacia esta modalidad, interés que se acrecienta según los usuarios empiezan a percibir sus enormes posibilidades.

Un pequeño grupo de trabajo puede ser una persona (como tú) que tiene dos o más PCs para desarrollar sus actividades y juegos, o bien una pequeña empresa con unos cuantos PCs, o un departamento de una gran empresa en la que no hay interacción con otros departamentos o redes compartidas. Los planteamientos en este caso serían los siguientes:

Primero, plantearse una forma asequible de compartir la información y de trabajar en grupo más productivamente.

Segundo, darse cuenta que las redes son caras, el S.O. del sistema operativo en red llega a costar cientos de miles de pesetas, resulta caro contratar a un experto que ponga la red en marcha y que la administre y mantenga.

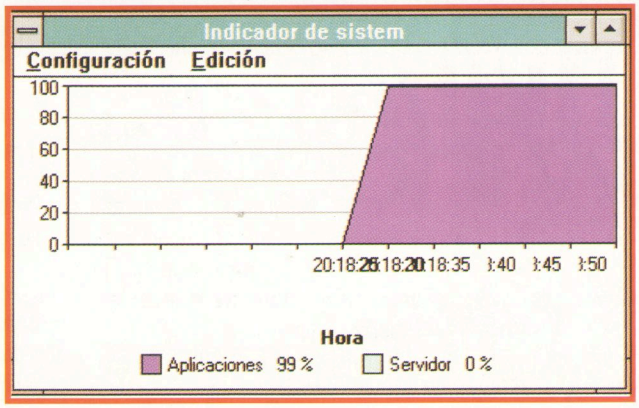
Y tercero, es difícil de configurar, de comprender y de usar, por ello los usuarios no llegan a aprovechar todas sus ventajas.

Pero, amigo, Windows para Trabajo en Grupo resuelve perfectamente estas necesidades de los pequeños grupos de trabajo: es una red económica, no necesita expertos ni de una persona dedicada a su administración y mantenimiento, no necesita de un PC dedicado a los labores de servidor y es muy fácil de usar, puesto que bastan con los conocimientos de Windows Sistema Operativo.

CONECTIVIDAD EN RED WINDOWS TRABAJO EN GRUPO 3.11

A la par de la presentación de windows para trabajo en grupo 3.1, apareció Windows NT, esto fue una estrategia Microsoft, por la pérdida de interés de Microsoft Lan Manager. Windows NT, también es conectable en red punto a punto, pero con un sistema de capas mucho más perfecto; con lo cual actua de soporte para la red integrada, es mas aùn, es un sistema de red mas que un sistema operativo, con lo cual es capaz de soportar otras redes locales.

Windows para trabajo en grupo es el mejor Software para cliente bajo MS-DOS. Funciona en modo protegido con un direccionamiento en red de 32 bits (cosa que la versión 3.1. nolo tenía), por supuesto soporta los mismos protocolos y la misma arquitectura en red. Y como siempre comento lo que ya sabéis, la ventaja de trabajar en un entorno igual.



Otras de la novedades de Windows 3.11 a nivel de prestaciones para el usuario, es poder compartir el FAX/MODEM, más capacidad en seguridad, control de red, y otros. En el campo de seguridad en red aparecen novedades pero con la sencillez de siempre, fácil y didáctico; con la posibilidad de conectar Windows NT y NT Advanced Server.

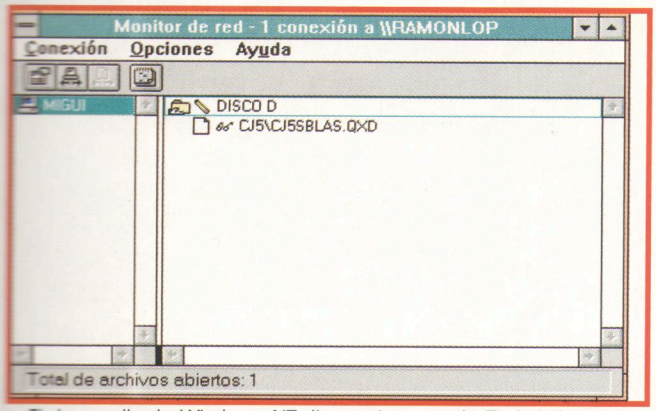
CONECTIVIDAD MS-DOS

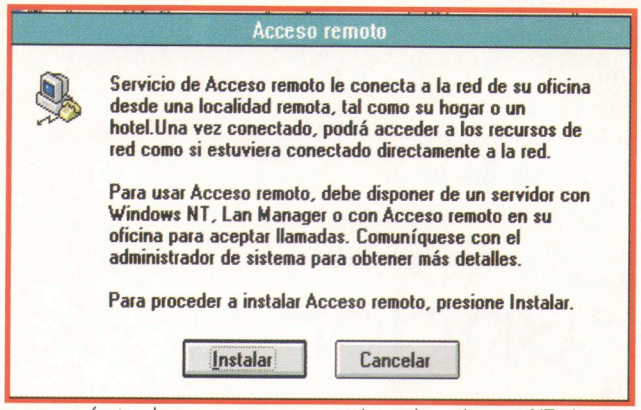
Una de las mejoras en la conectividad en red son unos controladores en red con 32 Bits. Esto permite duplicar la capacidad de trabajo en red. Por otro lado el uso de fichero de 32 Bits nos proporciona un medio más de eficacia sobre las operaciones en ellos. Pero tenemos que analizar bien las tarjetas controladoras IDE, ya que nos pueden dar "la vuelta a la tortilla ", en función de que no soporte trabajar con 32 Bits y esto nos influiría, fuertemente, en las prestaciones. Por ello estas tarjetas controladoras pueden ser perjudiciales o no. Una mejora destacable de NOVELL es el aumento de soporte en red punto a punto para el Sistema Operativo NOVELL-DOS 7.0, pero no para versiones anteriores, y sí gracias a nuestro apreciado Windows Trabajo en Grupo que nos facilita todo esto de forma que se puede trabajar sin problema alguno. Pero además, Windows Trabajo en Grupo nos permite compartir impresoras y las unidades en disco, como si todo fuera uno mismo. Es como un mundillo que rodea a Windows y no se separa de él, como a tí y mi nos gusta ; fácil, directo y didáctico.

El MS-DOS nos proporciona lo que ya conocemos (cosa que no me pararé a comentaros), pero Windows nos proporciona la posibilidad de controlar llamadas, almacenarlas, leerlas, e imprimirlas; es decir todo un proceso evolutivo de la información, con gran seguridad y mas sencillo de lo que parece. Por otro lado el programa soporte NET, se carga como residente y tan sòlo ocupa 5 KB en memoria convencional; y 55 KB en extendida. Esto es una gran mejora ya que se ha conseguido reducir el espacio considerablemnte. Los parámetros son muy variados, y no nos vamos a entretener en ello pero te mencionaré alguno de ellos: todo tipo de contraseñas, diversos protocolos, conexiones..., y union a Windows NT...etc.

BREVE HISTORIA DE WINDOWS NT

las imágenes corresponden a diversos aspectos de la red para trabajo en grupo





El desarrollo de Windows NT dio comienzo hace aproximadamente 5 años. En 1989 David Cluter estaba al frente del grupo DEC WEST, vio como esta empresa cancela su proyecto de un Sistema Operativo avanzado de 32 Bits; y éste, al sentirse traicionado, decidió irse de la empresa con sus 16 desarrolladores; pasándose todos a Microsoft, y presentando sus planes para NT (New Technology).

A los dos años, en 1991, mientras se estaba en la fase media de la creación de NT, Microsoft ya estaba cormecializándolo. Microsoft empezó a atraerse a los fabricante independientes de Software Unix; dedicándose a esta labor 100 miembros del grupo de Sistema, ya que por lo general UNIX mostraba una aptitud receptiva hacia NT. A principios de 1992, Microsoft había cogido impulso de los fabricantes independientes. En Abril de este mismo año, se celebró la muestra Spring/Condex Windows, en la cual se presenta oficialmente Windows NT. En Octubre de dicho año, Microsoft hizo publico su producto "Windows para Grupos de Trabajo", y unas semanas más tarde, en la muestra de Condex/Fall, se hicieron demostraciones con más de 200 aplicaciones de 32 Bits, declarando un portavoz de Microsoft que: "Antes de que termine el año se sacaran más de 50.000 juegos de NT". Tú bien lo sabes que ha sido cumplido.

LA COMPETENCIA

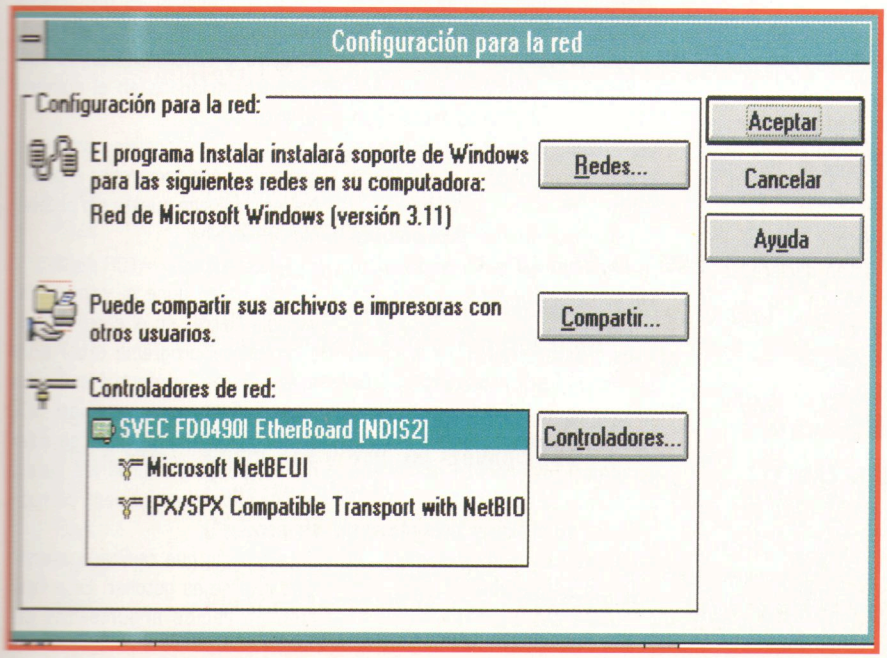
Se diga lo que se diga acerca de la tecnología de NT, Microsoft, tiene una gran ventaja respecto a la industria Unix , y es en todo lo referido a la comercialización. Esto se debe a la gran cantidad que posee tanto de expertos como de dinero y que supera en cualquiera de los casos a cualquier fabricante de Unix y cualquier otro.

"Nadie más, aparte de Madonna, sabe venderse mejor que Bill Gates. Microsoft puede echar del mercado al universo Unix entero ". Los partidarios de Unix se apresuran a señalar que esperan que la actitud de Microsoft hacia los estándares abiertos y otros posibles defectos alejarán a los clientes corporativos de unirse a NT. La nueva vorsión Unix de Univel, UnixWare, y demás entraran en contienda por captar muchos de los clientes que pretende captar NT. APPELBAUM aduce que "Microsoft entiende de software de productividad personal y de PC. pero no de de los sistemas que depende toda una empresa".

JOAL APPELBAUM cree que: Unix tiene ventaja en la expertización de los VAR. de los integradores de Sistemas y de consultores con años de experiencia en la resolución de problemas referentes a Unix, así como referente a programadores que conocen bien el Sistema. JIM BILLMATER, vicepresidente de comercialización de productos de SunSoft, cree que una cuestión clave inicial será la trayectoria que tiene Unix: La arquitectura está aquí en NT, pero llevará años hacerla consistente, afirma. Hemos tenido versiones multiprocesador de Solaris durante bastante tiempo y la gente esta utilizándolas para trabajar en ellas. A pesar de todo esto BILLMAIER también reconoce que :Microsoft será el único rival de SunSoft a largo plazo. En contraposición a todas estas declaraciones hay personas que también alaban a NT. (seguro que tú, eres uno de ellos) . Así LOU GIGLIO (Vicepresidente de investigación de la empresa de valores Bear Stearns & Company) declara que: Creo que NT va a ser un producto con éxito. No creo que de manera inmediata sea un éxito apoteósico. El ciclo de adopción será un poco diferente del de Windows.

Hecha esta breve introducción a el Windows para trabajo en Grupo, a su historia y desarrollo a lo largo de los años, materia ésta que todo programador debe conocer para saber en el entorno en el que va a moverse, os dejo por este mes.

En el número siguiente nos dejaremos de preámbulos y entraremos de lleno en lo que de verdad nos interesa; La programación bajo el entorno de Windows, una plataforma que resulta excepcional para juegos. Adiós.

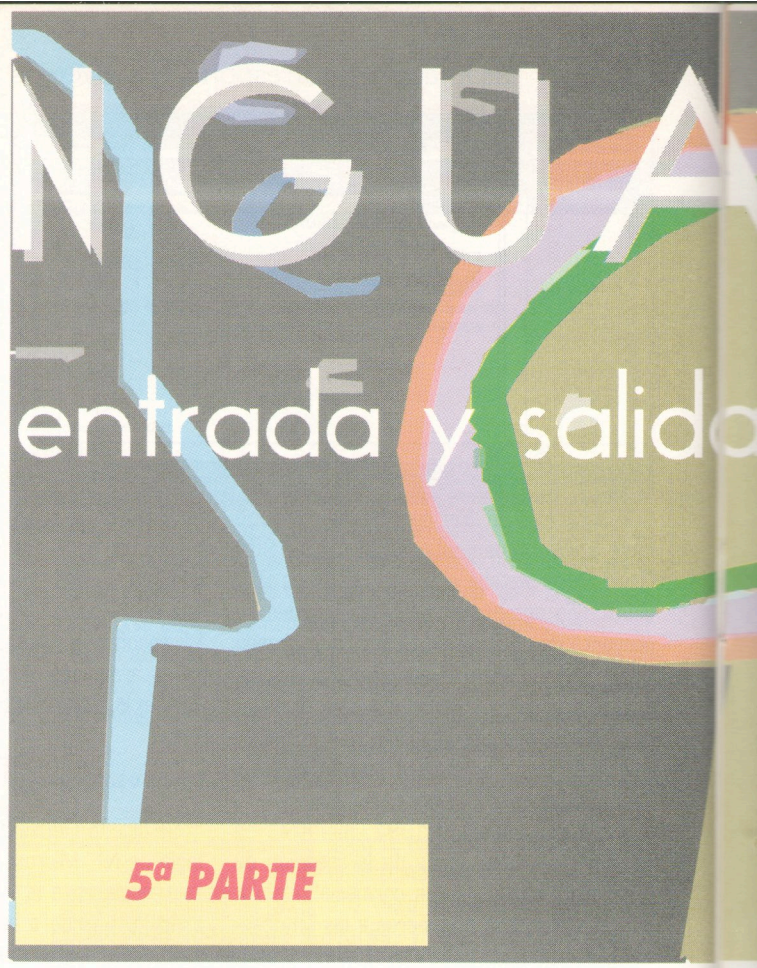


CURSO

LENGUAJE C

Por: José Antonio Brihuega

Funciones de entrada y salida



5ª PARTE

Las funciones de la bibliote ca de E/S estándar son, con diferencia, las más usadas por los programadores de C. Permiten una fácil interacción entre los programas y los canales (streams) estándar que proporciona el sistema operativo.

En el número anterior se introducía el concepto de "canales estándar", como una forma sencilla de tomar datos del teclado y ponerlos en la pantalla. Pero su función no termina ahí, pues el sistema operativo permite "redireccionar" estos canales hacia cualquier dispositivo o fichero externamente al programa, sin necesidad de modificación alguna. El programa sigue tomando o poniendo datos en el canal, sin saber que éste ha sido reconducido a otro dispositivo.

El único canal que no se puede redireccionar es el de errores (stderr). Esto es así en el entorno MS-DOS, aunque en otros sistemas puede hacerse, pero no es habitual.

El redireccionamiento se hace directamente sobre la línea de órdenes del MS- DOS, al arrancar el progra ma.

Tomando como ejemplo nuestro programa del núme ro anterior "lista.c", que escribía en pantalla la lista de caracteres ASCII, lo compilamos, produciendo el ejecutable "lista.exe". Para ejecutarlo escribimos en la línea de órdenes del DOS:

```
lista
```

Esto producirá en pantalla la lista ASCII, porque el canal de salida estándar (stdout), que utiliza el programa, se dirige "por defecto" a la pantalla.

Ahora probemos a escribir lo siguiente:

```
lista >PRN
```

El resultado aparece en la impresora, o se produce un error, en caso que ésta no esté conectada. El símbolo ">" indica al DOS que redireccione la salida estándar a la impresora (disposit ivo PRN).

Si queremos guardar el resultado en el archivo "prueba.txt":

```
lista >prueba.txt
```

También se puede redireccionar la entrada (con el símbolo "<"), o interconectar dos programas mediante una barra vertical (|), que hace de "tubería" (pipeline), conectando la salida del primero con la entrada del segundo (tal y como aparece en el dibujo de la página siguiente).

Por ejemplo, podemos pasar la salida de lista.c a nuestro filtro mayus.c, para que elimine las letras mayúsculas:

```
lista | mayus
```

Aparecerá en pantalla la lista ASCII, con las mayús culas convertidas en puntos.

Todo ello ha sido posible porque en estos programas hemos usado funciones de E/S que usan los canales estándar.

Las funciones que se explican a continuación están declaradas en la cabecera stdio.h, por lo que pondre mos la orden

```
#include <stdio.h>
```

...al principio de los programas que usen de ellas.

UN CARACTER CADA VEZ

Las funciones más sencillas son las que hemos estado usando en los números anteriores.

Estas funciones manejan un sólo carácter cada vez, produciendo valores de tipo int. Si usamos valores char, hay que tener en cuenta que se convertirán a int al ejecutarse la función. Habrá que tener cuidado con ello, pues podría hacer que el resultado no sea el que esperamos.

La función putchar pone en el canal de salida estándar (stdout) el carácter cuyo código ASCII le pasemos como parámetro entre los paréntesis.

```
putchar('100);
```

Esta orden pondrá en pantalla (o en el dispositivo direccionado por stdout) la letra d minúscula, a la que corresponde el código ASCII 100.

Como ya hemos explicado, si usamos un valor char, éste se convertirá automáticamente a int:

```
putchar('d');
```

Esto es equivalente a la orden anterior. La letra en cuestión debe estar, como es habitual, entre comillas simples, para que el compilador la interprete como una constante de tipo char.

Como en cualquier otra función, se puede usar como argumento el valor de una variable.

```
putchar(letra);
```

Si usamos una variable char, se regirá por las mismas normas, para la conversión a int.

Muchos piensan que putchar es una función void, es decir, que no devuelve resultado. En realidad sí lo hace, aunque normalmente se desecha. La función devuelve el mismo valor que se le ha pasado como parámetro. En el caso (harto improbable) de que ocurra un error en el canal, putchar devuelve la constante EOF, la cual ya fue mencionada en el número anterior.

Así pues, es posible usar este resultado para comprobar si nuestra letra ha sido puesta sin problemas en el canal:

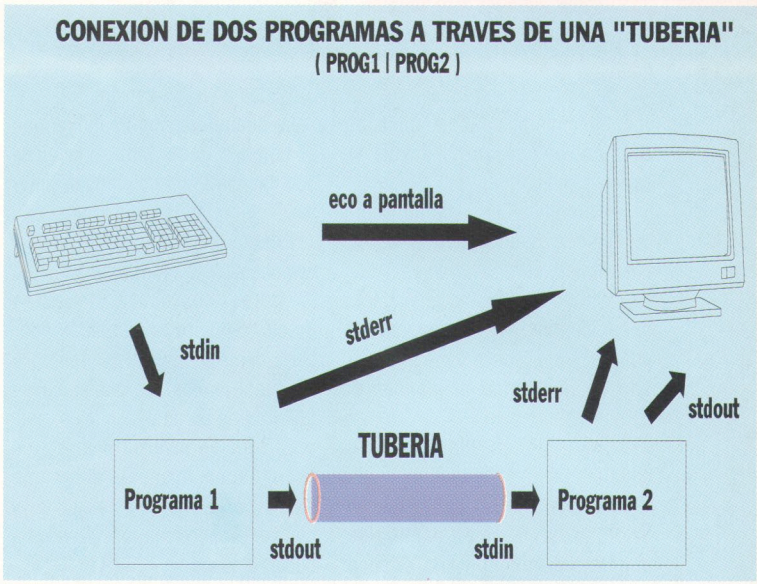
```
if (putchar(letra)==EOF) exit(1);
```

Esta orden pone el valor de la variable "letra" en el canal stdout y termina el programa (exit) si la operación no se efectuó correcta mente. Esta comprobación, sin embargo, no suele hacerse casi nunca, pues son raros los casos en que el canal no funciona correcta mente.

La función que realiza la operación inversa es getchar. Esta función no recibe argumentos de

JE

da estándar (II)



entrada (no se pone nada entre los paréntesis). Tras cada ejecución, devuelve el si guiente carácter que encuentre en el canal de entrada estándar (stdin), sin importar de qué dispositivo proviene. Si encuentra el carácter de final de fichero (control-Z), devuelve la constante EOF (End Of File). También devuelve EOF en el caso de que ocurra algún error en el canal.

Por ejemplo:

car = getchar();

lee un carácter de stdin, y lo almacena en la variable "car". Si dicha variable está declarada como char, el valor se "truncará" antes de ser almacenado, por lo puede perderse información. Esto es así porque el tipo char tiene un rango de valores menor que el int. Siempre que el valor leído esté dentro del rango char, la conversión se realizará sin problemas.

Este no es el caso de la constante EOF, cuyo valor está fuera del rango de char. Por tanto, si no tenemos cuidado de usar variables de tipo int, encontraremos problemas para detectar el final de fichero, tal y como se explicó en el número anterior. La secuencia si guiente:

letra = getchar(); if (letra == EOF) exit(0);

o su equivalente:

if((letra=getchar)==EOF) exit(0);

no funcionarán correctamente, a no ser que tengamos cuidado de declarar "letra" de tipo int.

Como ya se adelantó en el número anterior, el canal stdin funciona a través de un "buffer" (memoria intermedia). Los caracteres teclea dos van acumulándose en el buffer hasta que está lleno o se pulsa Intro.

Así pues, la función getchar se mantiene en espera hasta que el buffer es liberado, y entonces va leyendo los caracteres uno a uno.

En el caso que la entrada no esté siendo tomada del tecla do, el funcionamiento es idéntico.

PRINTF Y SCANF: EN TRADA Y SALIDA FOR MATEADA

Tomar y poner datos en los canales estándar a razón de un carácter cada vez puede ser útil en muchos casos; sin embargo, hay momentos en los que necesitamos presentarlos con un formato más elaborado, que incluya cantidades numéricas, cadenas de caracteres, etc. Lo mismo podemos decir de la entrada de datos, sea por teclado o no.

La función printf (print format) se encarga de poner en la salida estándar la información de acuerdo con el "formato" que le indiquemos.

Printf acepta un número variable de argumentos entre los paréntesis. El primero de ellos es el formato que indica exactamente la manera en que queremos los datos. El resto de parámetros, que son opcionales, son los propios datos (constantes, variables, expresiones, etc.) a los que nos referimos dentro del forma to.

printf("Formato", dato,...);

El formato debe ir entre comillas dobles, porque es una cadena de caracteres (las comillas simples sólo se usan con caracteres indivi duales). Los datos pueden ser de cualquier tipo válido en C.

Cualquier carácter dentro del formato se imprimirá tal cual. Así pues, una de las formas de usar printf es con un formato totalmente literal:

printf("Hola mundo");
printf("Buenos días\n");

Están permitidas las secuencias de escape (como \n y \r). Printf no da saltos de línea automáticamente, así que, si los queremos, tendremos que ponerlos directamente, como en el segundo ejemplo.

Si ejecutamos seguidas las dos

FORMATOS DE CAMPO PARA PRINTF

%d y %i	Entero decimal con signo
%o	Entero octal sin signo
%u	Entero decimal sin signo
%x	Hexadecimal sin signo (minúsculas)
%X	Hexadecimal sin signo (mayúsculas)
%f	Punto flotante (notación decimal)
%e	Punto flotante (notación científica)
%g	%f o %e según precisión necesaria
%E y %G	Como %e y %g, en mayúsculas
%c	Carácter simple
%s	Cadena de caracteres
%%	Imprime el caracter '%'
%p	Puntero (near XXXX, far XXXX:XXXX)

órdenes anteriores, el resultado sería:

Hola MundoBuenos días

El segundo printf sale "pegado" al primero, porque no incluía el salto de línea.

La verdadera potencia de printf la obtenemos al intercalar datos no literales. Esto lo haremos usando un signo de porcentaje, seguido de un identificador de tipo, que es una letra indicadora de la manera en que queremos presentar el dato:

```
printf("Tienes %d años.\n", edad);
```

La secuencia %d indica que el valor de la variable "edad" sea impreso como número entero (decimal).

Suponiendo que la variable edad tenga el valor 19, el resultado sería:

Tienes 19 años. No es obligatorio que el identificador se separe con espacios:

```
printf("Tienes%daños.\n", edad);
```

Dará como resultado:

Tienes19años.

En el cuadro adjunto tienes todos los formatos disponibles, aunque algunos se refieren a tipos de datos que todavía no se han explicado aquí. Como siempre, pueden existir pequeñas diferencias, según el compilador que usemos (esta tabla se refiere a Turbo C).

El número de datos que se pueden poner no está limitado, pero es obligatorio que cada uno tenga su correspondiente expresión al final, y que estén en el mismo orden. Esto es importante, pues si no el resultado será imprevisible.

```
printf("Tienes %d años y mides %f\n", edad, altura);
```

Si edad contiene 21, y como altura 1.81, el resultado sería:

Tienes 21 años y mides 1.810000

La razón por la que salen los ceros a la derecha del 1.81, es porque se está usando la precisión por defecto. Si deseamos un número determinado de cifras, ya sea en formato entero como flotante, lo pondremos entre el porcentaje y la letra:

```
printf("Tienes %d años y mides %1.2f\n", edad, altura);
```

Da como resultado:

Tienes 21 años y mides 1.81

La secuencia %1.2f indica que el número float debe escribirse con una cifra entera y dos decimales.

Existen muchos modificadores y opciones como esta, que permiten formatear los datos a nuestro antojo. Su estudio detallado ocuparía demasiado espacio. Todo ello se puede encontrar publicado en multitud de libros y manuales.

La función inversa a printf es scanf (scan format), que toma datos de la entrada estándar, y nos los proporciona formateados a nuestro gusto.

Scanf se usa igual que printf; incluso las secuencias de formato son iguales.

```
scanf("%d", &edad);
```

Este ejemplo toma de la entrada un número entero, y lo almacena en la variable edad.

El porqué del símbolo & se escapa a los conceptos hasta ahora explicados. Por ahora, sólo decir que si no lo ponemos, no obtendremos el resultado deseado.

Scanf proporciona como valor de retorno el número de valores leídos correcta mente (en el ejemplo, 1). Esto es útil para comprobar si hubo algún error en la lectura:

```
if (scanf("%d",&edad)!= 1) printf("Error.\n");
```

Al igual que en printf, el argumento alfanumérico puede contener reservas de espacio y textos normales.

Sin embargo, scanf es mucho más selectivo acerca de lo que se hace con el texto del argumento alfanumérico.

Es importante distinguir entre "espacios en blanco" (blancos, tabuladores, saltos de línea...) y caracteres que no sean espacios, a los que trata de forma diferente.

Así, la cadena de caracteres puede tener tres clases de información diferentes:

- Reservas de espacio (%) - Espacios en blanco - El resto de caracteres

Si scanf ve un carácter de espacio en blanco, leerá (no almacenará) cualquier espa cio hasta encontrar el primer carácter no blanco. Así, para la siguiente orden:

```
scanf("%d %c %f", &a,&c, &x);
```

podría ser una entrada válida la siguiente:

```
17    c    35.6
```

Sin embargo, si hubiéramos separado los campos con comas en vez de con espacios, sólo se permitiría una entrada como la que sigue:

```
17,c,35.6
```

Si scanf no encuentra el carácter que busca, la función se acaba inmediatamente, lo cual no suele ser muy deseable.

A causa de ello, scanf no es la mejor forma de capturar datos del teclado. Otras funciones, que se verán más adelante, realizan esa función con mejor soltura.

En la cadena que le pasamos a scanf, los caracteres que no son espacios serán buscados en la entrada, pero no almacenados. Simplemente los reconoce y se los salta. Esto será un efecto útil cuando sólo nos interese determinados trozos de la información que leemos. Scanf los saltará, y quedará en disposición de leer la siguiente información. Este efecto es especialmente útil cuando la entrada está siendo direccionada desde un archivo de base de datos, o algo parecido.

Al igual que en printf, es posible indicar los tamaños de los campos en sus reservas de espacio. Estos son interpretados como el número máximo de posiciones a leer para una variable dada.

Scanf se quedará "atascado" si se especifica una anchura de campo más pequeña que el dato que se va a leer. Esto se muestra en el pequeño programa SCANF.C que podéis ver aquí arriba.

Scanf.c

```
/* Programa para ilustrar el uso de la
 anchura de campo en scanf */

#include <stdio.h>

int main(void)
{
 int i1, i2, i3;
 char c1;

 printf("Por favor, introduzca un entero, un caracter y otro entero: ");
 i3 = scanf("%1d %c %d", %i1, &c1, %i2);
 printf("i1=%d\nc1=%c\ni2=%d\n", i1, c1, i2);
 printf("Se han procesado %d argumentos.\n", i3);

 return 0;
}
```

Si se le da como entrada, por ejemplo:

```
123 c 45
```

el resultado, totalmente inesperado, será:

i1=1 c1=2 i2=3 Se han procesado 3 argu mentos.

La primera reserva de espacio en el argumento alfanumérico de scanf indica que se lea un dígito como máximo. La función lo hace y busca el siguiente carácter. Toma al 2 como un carácter y continúa procesando, lee el 3 y, como ya ha leído todo lo que se le mandó leer, termina, afirmando que ha proce sado satisfactoriamente los tres datos que se le pidió.

Por tanto, ni siquiera podremos fiarnos del valor de vuelto por scanf.

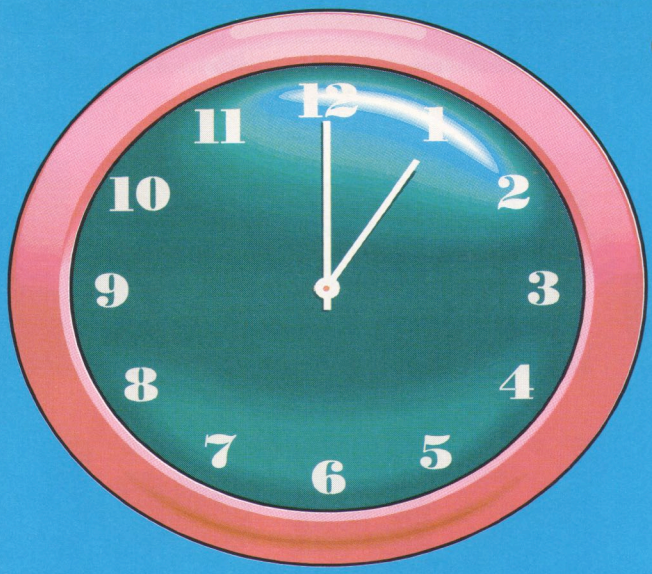
Como conclusión, podemos decir que scanf es mucho más útil cuando la entrada no se toma del teclado, y tiene un formato predecible. Al contrario, cuando se toman datos directamente del teclado es casi un "milagro" que el usuario llegue a introducir exactamente lo que nosotros queríamos.

Un buen programa deberá ser consistente frente los errores al teclear, para lo cual scanf suele ser de poca ayuda.

En el próximo número continuaré haciéndo el C más amigable para todos.****

Ajatel S.L.

CODIGO JUEGO



Ya sólo queda 1 mes

para el final

EL GRAN CONCURSO DEL ROMPECABEZAS INFORMATICO

Las bases del concurso son muy fáciles. Cada mes, y desde el número 1 de Código Juego se publica una pregunta cuya contestación es siempre un número de una única cifra. Cuando llegemos a la sexta pregunta, los seis numeros resultantes de éstas formarán un teléfono secreto.

El primer concursante que llame a este número en el día indicado en las bases que se explicaron en nuestra primera revista, recibirá en su domicilio un 486 DX2, totalmente gratis.

La pregunta de este mes:

En un artículo del mes pasado se hacía mención al formato interno de los archivos .MOD, pero ¿sabrías decirnos cuántos bytes ocupa el campo FINETUNE dentro de este tipo de archivos musicales?

El dato que obtengas lo debes multiplicar por el número de veces que se dice la palabra "AMIGA" en la portada el número anterior y sumárselo a la cifra que teníamos del mes anterior. Fácil, ¿verdad?

Por:

David Brioso

Analizando y programando la tarjeta

Sound

La detección

La programación de videojuegos es al mismo tiempo un arte y una técnica. Como técnica, existen una serie de fórmulas, de recursos, que, aplicados con inteligencia, pueden producir excelentes resultados. Como arte, sin embargo, no existen normas o cortapisas a la imaginación. Tanto el planteamiento de un juego como el diseño de los gráficos son fundamentales; pero tan importantes como ellos son la música y los efectos de sonido, que pueden convertir unas simples imágenes en algo que, de repente, cobra vida propia. En cierto sentido se podría comparar con la diferencia existente entre el cine mudo y el sonoro.

Durante bastante tiempo los ordenadores PC han estado enmudecidos por la falta de un hardware adecuado para la generación de sonidos o música. Hace unos años apareció una tarjeta de sonido, la Adlib, que, sin ser una maravilla, puso las primeras notas de color a aquellos frios beeps de la máquina. Un poco más tarde salió al mercado una nueva tarjeta de ampliación, la Sound Blaster, que hizo las delicias de más de un aficionado. No llegaba a alcanzar un nivel excepcional de calidad, pero nos tuvo vibrando durante una buena temporada. Y a partir de aquella placa tan primitiva apareció una nueva camada de tarjetas de ampliación de sonido que no tienen nada que envidiar a los mejores aparatos de música.

La tarjeta por excelencia en el reino de los videojuegos sigue siendo la Sound Blaster, que ha sentado un estándar aceptado por casi todos los fabricantes de hardware de sonido. Por tanto, la programación de ésta es una cuestión importantísima en el desarrollo del software de entretenimiento. Esta tarjeta, como casi todas en el mundo del PC, ha venido acompañada, mal acompañada, por una sequía absoluta de información sobre su hardware y su manejo. Creative Labs, la compañía responsable del diseño, fabricación y distribución de la Sound Blaster, no podía haber sido más parca a la hora de publicar información interna sobre la tarjeta. Durante años tras su aparición, la única fuente de información fiable fue el paquete de programación de la propia compañía. El esfuerzo de muchos programadores, tanto aficionados como profesionales, tuvo su recompensa poco a poco. Ahora mismo resulta factible programar cualquier aspecto de la tarjeta accediendo directamente a su hardware, sin necesidad de utilizar las librerías del kit de desarrollo de la Sound Blaster.

La primera parte se esta serie de artículos sobre la programación de esta tarjeta cubrirá el aspecto más elemental de la misma: su propia detección. Cada programa ha adoptado un sistema diferente para averiguar si una tarjeta de sonido está instalada en el ordenador en cuestión, desde interrogar al usuario hasta la detección directa del hardware, pasando por la lectura de las variables del Sistema Operativo. Preguntar al usuario representa una molestia añadida a las habituales, calibración del joystick, configuración del juego, password, etc. Leer las variables del Sistema puede ser un sistema válido siempre que el usuario se haya molestado en configurarlas previamente. En caso contrario, resulta imposible detectar la presencia de la tarjeta. Por lo tanto, parece que el método más fiable es la investigación por cuenta propia.

La aparición de, literalmente, centenares de clónicos de Sound Blaster no ha hecho sino empeorar las cosas. Lo que en principio podía ser una cuestión circunstancial se ha convertido en una enfermedad endémica del PC: la incompatibilidad. Cualquier fabricante de hardware toma una patente, la modifica ligeramente y la explota. Por supuesto, las tarjetas de sonido no iban a ser ninguna excepción, así que nos encontramos con múltiples variantes de los mismos chips. Para evitar los problemas que acarrearía el tener que comprobar tarjeta por tarjeta, vamos a limitarnos a presentar un algoritmo de detección de la Sound Blaster, que debería funcionar con todas las demás tarjetas "compatibles".

La Sound Blaster tiene un determinado número de posibilidades de configuración que se seleccionan en la propia placa. Podemos elegir en qué IRQ (interrupción generada externamente), canal de DMA ("Direct Memory Access", Acceso Directo a Memoria) o dirección I/O queremos situar la tarjeta. Estos detalles suelen escaparse de nuestra atención hasta que un juego nos recuerda que la configuración de la tarjeta está mal elegida. Entonces aparecen los problemas.

Para detectar la presencia de una tarjeta podemos comprobar directamente si las direcciones I/O responden a nuestras llamadas. Este puede ser el método más sencillo y efectivo, pero es conveniente adornarlo con una serie de comprobaciones accesorias para evitar malentendidos. Con este sistema averiguaremos, además, qué dirección I/O base está configurada en la tarjeta. Acostumbra a ser 0220h, que es la dirección por defecto u omisión, pero no debemos contar con ella de antemano, ya que algunas

dblaster

tarjetas están configuradas en la dirección 0240h, por ejemplo.

En este programa de detección hemos incluido las dos direcciones más habituales de la tarjeta Sound Blaster. Si fuera necesario detectar todo el rango de direcciones de que disponen los modelos más antiguos, bastaría con repetir el procedimiento detecta_blaster para las otras direcciones, 0210h, 0230h, 0250h y 0260h.

Los pasos fundamentales de la detección de la tarjeta de sonido son los que siguen: desconexión de la salida de sonido; test del chip DSP ("Digital Signal Processor", Procesador Digital de Señal, el corazón de la Sound Blaster) y la lectura del registro de estado, que da mayor fiabilidad a las pruebas.

En la tarjeta, todos los accesos al hardware se realizan a través de una serie de puertos o direcciones I/O que es conveniente conocer. Si tomásemos 0220h como dirección de partida para la tarjeta, la lista sería así:

0226h puerto de reset ("DSP Reset").
022Ah puerto de lectura de datos del DSP ("DSP Read data")
022Ch puerto de escritura de comandos/datos ("DSP Write Data or Command")
022Ch puerto de estado del buffer de escritura ("DSP Write Buffer Status")
022Eh puerto de estado del buffer de lectura ("DSP Data Available Status")

La Sound Blaster tiene más direcciones de lectura y escritura, pero las únicas necesarias para la generación de sonido o música digitales son estas cinco.

El acceso a la primera dirección resulta en un reset del chip DSP, tras el cual se debe comprobar si éste devuelve el byte 0AAh en su puerto de datos. Este byte es la prueba genérica de que se trata de una tarjeta Sound Blaster o compatible con ella. El procedimiento test_sblaster se encarga de leer dicho byte. Si no aparece tras varios cientos de lecturas consecutivas podemos descartar el resto del proceso de detección, ya que, o bien la tarjeta no está instalada, o bien ha sido configurada en otra dirección I/O.

El siguiente paso en la detección de la tarjeta es la lectura del puerto de datos del DSP para comprobar si devuelve el byte 55h, que es el inverso de 0AAh. Si este byte aparece, podemos considerar que la tarjeta de sonido es una Sound Blaster sin temor a equivocarnos.

Quien haya observado de cerca los procedimientos de lectura o escritura de datos en los puertos de entrada y salida de la tarjeta, se habrá percatado de la complicación que tienen estas operaciones. Al ser tan lento el acceso al hardware, hay que comprobar si éste se encuentra en disposición de recibir un nuevo dato antes de enviarlo para evitar que éste se pierda o hacer una lectura errónea. Esto no representa ninguna dificultad siempre que se respeten las prioridades de acceso y no se escriban o lean valores antes de que el DSP nos dé su conformidad.

LA DETECCIÓN DE LA TARJETA A NIVEL DE HARDWARE ES EL PRIMER PASO QUE TODO PROGRAMADOR DE SONIDO DEBERÁ TOMAR. DE ESTA FORMA SE ASEGURA QUE EL PROGRAMA SERÁ PLENAMENTE COMPATIBLE CON LA SOUNDBLASTER.

El test de presencia de la tarjeta empleado por el programa CTVOICE.DRV, por ejemplo, comprueba también la versión del chip DSP instalado para detectar el hardware obsoleto, en el que algunos programas no funcionarían. Nosotros no hemos considerado necesaria esta última comprobación porque la inmensa mayoría de los usuarios, por no decir todos, tienen tarjetas Sound Blaster o compatibles adquiridas de hace cinco o seis años a esta parte.

El chip DSP recibe nuestras órdenes a través del puerto de escritura de datos y comandos, de la forma utilizada en el procedimiento escribe_DSPCOM1. En primer lugar espera a que aparezca un cero en el bit 7 del registro, y después escribe el dato en sí. El DSP tiene varias decenas de comandos, de los que ahora mismo nos pueden interesar unos pocos. Entre ellos están los siguientes:

DSPCOM 0D1h Desconecta la salida del altavoz
DSPCOM 0D3h Conecta la salida del altavoz
DSPCOM 0E0h Devuelve el ID (código identificativo) del DSP
DSPCOM 0E1h Devuelve el número de versión del DSP

La desconexión de la salida del sonido tiene como objetivo evitar ruidos espúreos que pudieran aparecer al manipular los registros internos de la tarjeta. Aunque pueda parecer que la desconectamos dos veces, ésto no ocurre porque en uno de los dos intentos la dirección I/O no es la correcta y, por lo tanto, no tendrá ningún efecto sobre la configuración del chip DSP.

Esperamos que estas rutinas de detección del hardware de sonido os sean de utilidad, y que no olvidéis mezclar las dosis exactas de programación técnica y de arte. Bueno, ¡menos sermones! ¡Hasta pronto!

David Brioso Santos

Detección de la Soundblaster: DBLAST.ASM

```
; ================================
;
; Programa:  DETBLAST.COM
;  Version:  1.00
;    Autor:  David Brioso Santos
;    Fecha:  19.08.94 / 20.08.94
; Copyright  (c) 1994 David Brioso
;
; ================================

segmento_codigo  SEGMENT

                 ASSUME   CS:segmento_codigo,DS:segmento_codigo,SS:segmento_codigo

                 ORG      0100h                    ; *01*

inicio_codigo:   call     copyright                ; Impresión del copyright
                 call     detecta_blaster          ; Detecta la tarjeta SBlaster
                 call     imprime_config           ; Imprime la configuración
                 mov      ax,4C00h        ; Función DOS para regresar
                 int      21h                      ; al Sistema Operativo

ioa_base dw      0                        ; *02*

; =-=-=-=-=-=-= DETECCION HARDWARE =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

detecta_blaster  PROC

                 mov      dx,0220h        ; Dirección I/O base para el
                 call     test_presencia           ; test; intenta detectar una
                 jnc      detectado                ; tarjeta en esa dirección
                                                   ;
                 mov      dx,0240h        ; Dirección I/O base para el
                 call     test_presencia           ; test; intenta detectar una
                 jnc      detectado                ; tarjeta en esa dirección
                 ret                               ; Vuelve a la rutina principal
                                                   ;
detectado:       mov      cs:[ioa_base],dx ; Guarda la dirección correcta
                 ret                               ; Vuelve a la rutina principal

detecta_blaster  ENDP

; =-=-=-=-=-=-= DETECCION DSP =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

test_presencia   PROC

                 call     desconecta_out           ; Desconecta salida de altavoz
                 call     retardo_largo            ; Espera varios milisegundos
                 call     test_sblaster            ; Comprueba la presencia
                 jc       no_detectada             ; Si hay error, salta fuera
                 call     lee_sblaster             ; Lee el status de la tarjeta
no_detectada:    ret

test_presencia   ENDP



retardo_largo    PROC

                 mov      cx,0FFFFh                ; Repite un bucle de retardo
                 loop     $                        ; 65535 veces
                 ret                               ; Vuelve a la rutina principal

retardo_largo    ENDP
```

```
desconecta_out   PROC

                 push     dx                       ; Guarda la dirección de prueba
                 add      dl,0Ch                   ; 02xCh = "DSP Write Command"
                 mov      al,0D3h                  ; DSPCOM: Desconecta Salida
                 call     escribe_DSPCOM1          ; Envía el comando al DSP
                 pop      dx                       ; Recupera la dirección base
                 ret

desconecta_out   ENDP


escribe_DSPCOM1  PROC

                 mov      cx,512                   ; Repite la lectura 512 veces
                 xchg     al,ah                    ; Guarda el valor de escritura
lee_DSPCOM_10:   in       al,dx                    ; Lee el status del DSP y
                 or       al,al                    ; espera hasta que el bit 7
                 jns      lee_DSPCOM_11            ; se encuentre a 0
                 loop     lee_DSPCOM_10            ; Si está a 1, sigue intentando
                 stc                               ; Devuelve el flag Carry
                 ret                               ; encendido para indicar error

lee_DSPCOM_11:   xchg     al,ah                    ; Retoma el valor de escritura
                 out      dx,al                    ; y lo envía al DSP
                 clc                               ; Borra el flag Carry para
                 ret                               ; indicar que no hubo error

escribe_DSPCOM1  ENDP


test_sblaster    PROC

                 push     dx                       ; Guarda la dirección base
                 add      dl,06h                   ; 02x6h = Reset del DSP
                 mov      al,01h                   ; Escribe un 1 en el puerto
                 out      dx,al                    ; del DSP para hacer un reset
                 in       al,dx                    ; Lee cinco veces el mismo
                 in       al,dx                    ; puerto para asegurarse de
                 in       al,dx                    ; que transcurren al menos
                 in       al,dx                    ; tres microsegundos entre
                 in       al,dx                    ; una escritura y la siguiente
                 mov      al,00h                   ; Escribe un 0 en el puerto
                 out      dx,al                    ; del DSP para hacer un reset
                                                   ;
                 mov      bl,16                    ; Repetiremos el proceso 16
lee_status_1:    mov      cx,512                   ; veces para mayor seguridad
                 add      dl,08h                   ; 02xEh = Status del DSP
lee_status_2:    in       al,dx                    ; Lee el puerto de status
                 or       al,al                    ; Comprueba si el bit 7 está
                 js       lee_status_3             ; encendido; si lo está, salta
                 loop     lee_status_2             ; y, si no, vuelve a intentarlo
                 pop      dx                       ; Recupera la dirección base
                 stc                               ; y devuelve el flag Carry
                 ret                               ; encendido para indicar error

lee_status_3:    sub      dl,04h                   ; 02xAh = Lectura de datos DSP
                 in       al,dx                    ; Lee un byte del DSP
                 cmp      al,10101010b             ; Comprueba si AL = 0AAh
                 je       lee_status_4             ; Si son iguales, correcto
                 dec      bl                       ; Vuelve a intentarlo otra
```

```
                jnz     lee_status_1            ; vez hasta 16
                pop     dx                      ; Recupera la dirección base
                stc                             ; y devuelve el flag Carry
                ret                             ; encendido para indicar error

lee_status_4:   pop     dx                      ; Recupera la dirección base
                clc                             ; y devuelve el flag Carry
                ret                             ; apagado si no hubo errores

test_sblaster   ENDP


lee_sblaster    PROC

                push    dx                      ; Guarda la dirección base
                add     dl,0Ch                  ; 02xCh = "DSP Write Command"
                mov     al,0E0h                 ; DSPCOM: Lee DSP_ID
                call    escribe_DSPCOM1         ; Escribe el byte en el DSP
                jc      error_sblaster          ; Tarjeta no identificada
                mov     al,0AAh                 ; Escribe 0AAh en el DSP
                call    escribe_DSPCOM1         ; y, si devuelve un error,
                jc      error_sblaster          ; salta fuera
                                                ;
                add     dl,02h                  ; 02xEh = Status del DSP
                mov     cx,512                  ; Repetimos la lectura 512 veces
lee_DSP_status: in      al,dx                   ; Lee el byte de status del DSP
                or      al,al                   ; hasta que el bit más alto se
                js      DSP_preparado           ; encuentre encendido
                loop    lee_DSP_status          ; Si no, vuelve al bucle
error_sblaster: pop     dx                      ; Recupera la dirección base
                stc                             ; y devuelve el flag Carry
                ret                             ; encendido para indicar error

DSP_preparado:  sub     dl,04h                  ; 02xAh = Lectura datos del DSP
                in      al,dx                   ; Lee un byte del DSP
                cmp     al,01010101b            ; Comprueba si es igual a 55h
                je      DSP_correcto            ; Si lo es, SBlaster detectada
                pop     dx                      ; Recupera la dirección base
                stc                             ; y devuelve el flag Carry
                ret                             ; encendido para indicar error

DSP_correcto:   pop     dx                      ; Recupera la dirección base y
                clc                             ; devuelve el flag Carry a 0
                ret                             ; para indicar que no hay error

lee_sblaster    ENDP

; =-=-=-=-=-=-= MUESTRA CONFIGURACION =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

imprime_config  PROC

                mov     ax,cs:[ioa_base] ; Dirección I/O base de SBlaster
                or      ax,ax                   ; Comprueba si es igual a cero
                jz      no_hay_tarjeta          ; Si lo es, salta a error

                push    cs                      ; Carga DS con el mismo valor
                pop     ds                      ; que el segmento de código
                mov     ah,9                    ; Función DOS para imprimir
                mov     dx,offset _mensaje_1    ; una cadena de texto ASCII
                int     21h                     ; en la pantalla
                clc                             ; Borra el flag Carry y
                ret                             ; regresa a la rutina principal

no_hay_tarjeta: push    cs                      ; Carga DS con el mismo valor
                pop     ds                      ; que el segmento de código
                mov     ah,9                    ; Función DOS para imprimir
                mov     dx,offset _mensaje_1    ; una cadena de texto ASCII
                int     21h                     ; en la pantalla
                stc                             ; Enciende el flag Carry y
                ret                             ; regresa a la rutina principal

imprime_config  ENDP

_mensaje_1      db      'Una tarjeta Sound Blaster o compatible ha sido '
                db      'detectada en el ordenador.',10,13,36

_mensaje_2      db      'La tarjeta Sound Blaster no ha podido ser detectada '
                db      'en este ordenador.',10,13,36

; =-=-=-=-=-=-= COPYRIGHT =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

copyright:      push    cs                      ; Carga DS con el mismo valor
                pop     ds                      ; que CS (segmento_codigo)
                mov     dx,offset cadena_texto  ; Apunta DX a la cadena de texto
                mov     ah,9                    ; Función del MSDOS para impri-
                int     21h                     ; mir cadenas de texto ASCII
                ret                             ; en la pantalla en modo texto

cadena_texto    db      'DETBLAST Versión 1.00.  Detecta la presencia '
                db      'de una tarjeta Sound Blaster.',10,13
                db      'Copyright (c) 1994 David Brioso Santos.',10,13,10,36

segmento_codigo ENDS

                END     inicio_codigo
```

```
; ****************************************************************************
;
; Notas:
;
; (*)   Los números de nota se indican entre asteriscos para que puedan distin-
;       guirse de valores enteros (por ejemplo: *9* ).
;
; (**)  Las palabras clave o directivas del ensamblador (TASM, MASM, etc, no el
;       lenguage) están en mayúsculas.
;
; (1)   Todos los ejecutables .COM comienzan su ejecución en la dirección 0100h
;       del segmento de código (CS:0100h).  Los ejecutables de tipo .EXE pueden
;       comenzar en cualquier dirección del segmento de código principal.
;
; (2)   En esta variable se almacena la dirección I/O de la tarjeta de sonido o
;       un 0000h si no pudo ser detectada. Las tarjetas Sound Blaster 1.0 ó 1.5
;       pueden ser configuradas para funcionar con el siguiente rango de direc-
;       ciones I/O:
;
;       0210h, 0220h, 0230h, 0240h, 0250h y 0260h
;
;       En cambio,  la última versión de la tarjeta Sound Blaster estándar (2.0)
;       y las tarjetas Sound Blaster Pro  solamente permiten la configuración en
;       las direcciones 0220h y 0240h.
;
; ****************************************************************************
```

Detección de la Soundblaster: DBLAST.ASM

Aprende a poner música en tus juegos

5ª PARTE

Music Instrument MIDI Digital Interface

Por:

José Manuel Ginés

El asesino entra en escena. La puerta del dormitorio chirría y sus pasos se oyen fuertes y estruendosos haciendo crujir la vieja madera del parqué. De repente una música tétrica hace que la sangre del jugador se hiele en sus venas. No hay duda. Los juegos de ordenador ganan mucho cuando la acción viene acompañada de una melodía. Por ello, es necesario que si queréis que vuestros juegos sean buenos y con calidad, tenéis que aprender como poner el toque musical adecuado. Continúa el estudio del interfaz MIDI y pronto sabrás como hacerlo.

TABLA DE IMPLEMENTACION MIDI

La tabla de implementación Midi es usada para una referencia rápida, saber qué funciones puede recibir y transmitir los instrumentos con Midi. No todos los sintetizadores y de más dispositivos con Midi transmiten o reciben los mismos mensajes Midi, es decir, por ejemplo si tenemos dos sintetizadores interconectados vía Midi, uno llamado A y el otro B. El sintetizador A puede transmitir presión y velocidad de teclado (tan fuerte se opriman las teclas) pero el sintetizador B que está recibiendo la información enviada por A, únicamente puede reconocer los mensajes de velocidad de teclado y no de presión de teclado, así que sólo responderá a la fuerza en que las teclas se oprimen (velocidad). En otras palabras, el sintetizador B no entiende lo que es el mensaje Midi, la presión de teclado, porque no fue incorporado durante el diseño del sintetizador B.

Hay varias razones por las cuales diferentes dispositivos con Midi entienden diferentes mensajes. Una de las razones es simplemente que el equipo diseñador de un aparato con Midi pensó que no era necesario implementar algunos de los mensajes. Otra razón es que la manufacturera de sintetizadores quiere mantener sus productos al alcance del consumidor. Para implementar la función de presión de teclado en un sintetizador es necesario aumentar el número de componentes electrónicos y otros materiales como sensores de presión para el teclado.

Para poder saber con facilidad que mensajes Midi son transmitidos y recibidos por los aparatos que posee el usuario, existen lo que se le da el nombre de Tabla de Implementación Midi. Esta tabla viene al final de cada manual de instrucciones de los aparatos con Midi. Por un acuerdo de la MMA y JMSC, cada compañía de sintetizadores debe incluir esta tabla haciendo saber al público qué mensaje transmiten y aceptan sus aparatos Midi. Esta es muy útil y lo digo por experiencia. Es muy molesto el no entender por qué el sintetizador B no produce ningún efecto cuando se hace más presión en el teclado del A, especialmente durante una sesión de grabación donde el tiempo es dinero, pero si se lee la tabla de implementación Midi en el manual de instrucciones del sintetizador, sabremos que mensajes mensajes se pueden recibir y transmitir.

La tabla de implementación Midi es recomendable adquirir o usar en diferentes situaciones. La manera de leer esta tabla es, en primer lugar, comparando qué funciones o mensajes son transmitidos por el controlador maestro que se desea

utilizar y cuáles funciones son reconocidas por los aparatos esclavos.

A continuación se explica cómo se lee esta tabla y como ejemplo tomaré la tabla de implementación Midi del sintetizador D-50 de la compañía Roland. (tambien se ha incluido la del sintitizador M1 de Korg).

La tabla consiste en cuatro columnas. La columna de la izquierda es la columna de las funciones o mensajes Midi ya especificados por la MMA, es decir, la nota activada, cambio de controladores, etc... La segunda columna indica qué funciones son transmitidas cuando el instrumento actúa como el controlador maestro. La tercera columna es la recepción de funciones, cuando el instrumento actúa como esclavo, en esta columna es donde se dará cuenta qué mensajes o funciones Midi pueden entender y responder el sintetizador. La última columna es para comentarios, es decir, cualquier instrucción específica que el fabricante del instrumento quiera dar a conocer. Cuando lean estas tablas se encontrarán varios símbolos. Un círculo "O" significa que el instrumento Si recibe y transmite el mensaje de que se está hablando y una "X" significa que el instrumento no puede responder a los mensajes enviados por el controlador maestro. También, en la parte inferior de cada tabla se encuentra el código de los mensajes de MODO que es la manera como los instrumentos esclavos responderán a los mensajes enviados por el controlador maestro, es decir, si la información es recibida por los 16 canales Midi o si se tiene que asignar un canal específico para recibir la información. La descripción de las funciones es la siguiente:

CANAL BASICO (basic channel): la función del canal básico se divide en dos y son:

a)Designación de fábrica (default). Este es el número de canal Midi en que el instrumento ha sido asignado desde la fábrica. Por lo general es el canal número 1. Tambien indica si el usuario puede cambiar esa asignación por medio del panel frontal. En el D-50 los 16 canales pueden ser transmitidos y recibidos. Estos se pueden asignar al canal deseado. Tambien, los cambios se pueden almacenar en la memoria como parte del sonido.

b)Cambio de canales (change). Esto quiere decir quñe el instrumento puede cambiarse a cualquier canal (1 al 16) para transmitir y recibir los mensajes Midi.

En el D-50 todos los canales se pueden usar para transmitir y recibir información Midi.

MODO, (Mode): Esta función es la manera en que los instrumentos esclavos recibirán la información Midi. El código de los diferentes modos se encuentran en la parte inferior de la tabla de implementación Midi de cada instrumento.

a) Designación de fábrica (default). Cuando el aparato se enciende por vez primera, generalmente viene asignado en el modo número 1 (OMNI ACTIVADO/POLIFONICO). Este modo se puede cambiar al modo deseado dependiendo de la aplicación. La razón por la cuál el Modo 1 es generalmente asignado de fábrica, es debido a que si el comprador es novato, que pueda conectar dos instrumentos vía Midi y pueda tocar sin preocuparse en asignar los canales Midi.

El D-50 transmite la información Midi en el modo 3 (OMNI/DESACTIVADO/POLIFONICO) y recibe la información en cuanto se enciende en los modos 1, 3, y 4. Estos modos se pueden almacenar en la memoria del instrumento para no tener que cambiar cada vez que se encienda el aparato.

b) Mensajes (Messages). Son la descripción de cada Modo. La abreviaciones de estos se encontrarán en las tablas de la siguiente manera: el Modo 1=omni on/poly; El modo 2=omni on/mono, el Modo 3=omni off/poly y el Modo 4=omni off/mono. El D-50 transmite en el Modo 3 que es OMNI OFF/POLY y recibe la información en los Modos 1, 3, y 4, es decir, en OMNI ON/POLY, OMNI OFF/POLY y OMNI OFF/MONO, respectivamente.

c)Alteración de modo (altered). Esta función ha sido incorporada para que en caso de que el instrumento esclavo no responda a un Modo en particular, éste sea sustituido por otro automáticamente para poder recibir la información Midi. Por ejemplo, digamos que el instrumento controlador maestro es una guitarra controladora de sintetizador que envía los mensajes Midi en el Modo 4 (Omni off/mono), es decir, cada cuerda será asignada a un canal diferente. Ahora bien, si el sintetizador que se está controlando con la guitarra no puede recibir en el modo 4, éste se cambiará a otro modo automáticamente para que pueda recibir la información enviada por la guitarra controladora de sintetizadores. En las tablas notarán que en la columna de información, recibida, estará escrito el siguiente: Mode 4 > Mode 3. Esto significa que si el esclavo recibe el byte de estado desde el maestro que le indica qule los mensajes vienen en el modo 4, éste, el esclavo, recibirá los mensajes en el modo 3. El modo al que se cambia es decidido por el equipo diseñador del instrumento. Por supuesto, el sintetizador esclavo no responderá correctamente a la acción de la guitarra porque no es el modo apropiado.

En el D-50, si el Modo en que la transmisión de información se lleva a cabo es el modo 2, este no lo entenderá y consecuentemente lo remplazará del 2 al 1. En otras palabras, recibirá la información en el Modo (OMNI ON/POLY).

- NUMERO DE NOTA: (note number). El número de nota es el número ya estandarizado para cada tecla del sintetizado. Una manera de usar el Número de Nota es cuando se quiere controlar una caja de ritmos por medio del teclado maestro, es decir, en lugar de tocar los botones en la caja de ritmos para producir los sonidos de batería y de percusiones, estos son controlados por medio del teclado del sintetizador. El usuario deberá asignar los números de cada tecla a cada botón de la caja de ritmos según se desee. La asignación se lleva a cabo por medio de los controles del panel frontal de la caja de ritmos. Otra aplicación es cuando se tiene un sintetizador multi-timbral y el usuario desea dividir el teclado en diferentes partes o zonas para poder asignar más de un sonido diferente en el mismo teclado del sintetizador multi-timbral.

En la tabla de implementación Midi se encuentra el rango de notas que puede ser transmitido. El rango de notas que el sintetizador transmite, es usualmente el mismo número de teclas que posee. En un teclado de cinco octava como el DX-7 de la compañía de sintetizadores Yamaha, el rango de notas es del 36 al 96 siendo el número 36 la primera tecla del DX-7 o sea el primer DO y el 96 es el último DO del teclado, la última tecla.

En la columna de información "recibida" por lo general el rango es de 0 al 127, es decir que ese sintetizador en particular es capaz de recibir todo el rango de la numeración de un teclado de diez octavas, la mayoría de los sintetizadores cuentan con una función con la cual se puede transportar el tono varias octavas arriba o abajo del tono original.

La función llamada Numeración real (true voice) que es parte de la función del Número de nota, es en sí el número de nota que un instrumento puede recibir. En otras palabras, son los tonos que pueden reproducir. Para que pueda reproducir el sonido de las notas fuera del número de teclas, se tendrá que transportar en octavas al teclado, ya sea hacia arriba o hacia abajo dependiendo de las notas transmitidas.

El D-50, transmite el rango de número de notas del 12 al 108, y puede recibir los números de nota del 0 al 127. En lo que concierne en "numeración real", el D-50 puede recibir notas del número 12 al 108. El teclado de un "grand" piano es de ocho octavas y su numeración va desde 21 a 108.

VELOCIDAD (velocity): Esta función hace que un sintetizador responda a la variación dinámica de un teclado. La nota activada (note on) y la nota desactivada (note off) son partes de la función de la velocidad.

Nota activada (note on). Si en la columna de información "transmitida" aparece un círculo, esto quiere decir que la velocidad de teclado puede ser transmitida, si tiene una "x" significa que el teclado no envía la función Velocidad de teclado. Cuando los sintetizadores no cuentan con velocidad de teclado, estos envían a los esclavos conectados a él el valor numérico 64 que significa que todas las notas son enviadas al mismo volumen (sin dinámica). La representación del mensaje en forma binaria es la siguiente:

1001nnnn

9nH

00111100
60
01000000
64

El primer byte es el código de la nota activada y la "n" es es número de canal Midi.
El segundo byte es la tecla que corresponde al número 60 (DO central)
El tercer byte es el número 64 que singnifica que todas las notas oprimidas son del mismo volumen.
Ahora bien, si la columna de información "recibida" de la tabla contiene una "X", quiere decir que ese sintetizador no recibe velocidad de teclado y que reproducirá al mismo volumen todas las notas recibidas, por el contrario, si tienen un círculo, este sintetizador o módulo (sintetizador sin teclado) recibirá el mensaje de velocidad de teclado. El rango de velocidad de Teclado por lo general es del 0 al 127. El 127 significa que el sintetizador será muy sensible a los cambios de velocidad de teclado y creará efectos dinámicos, es decir, pianisimo, piano, forte, mezoforte, etc. Si la recepción de velocidad en el sintetizador esclavo es cero, entonces, no reproducirá sonido alguno.
Nota desactivada (note off). Una nota activada con velocidad de cero (0) es equivalente a una nota desactivada y por lo general se muestra en la tabla de implimentación Midi de la siguiente manera: 9n v=0. El 9H es la representación del mensaje de la nota activada en el sistema hexadecimal, la "n" es el número del canal Midi en que esa nota se transmitirá. Si la columna de información "transmitida" contiene una "x", quiere decir que el sintetizador no envía velocidad cuando una tecla se desactiva (la velocidad en que se levanta el dedo de la tecla). Si la columna de información "recibida" tiene una "x", el sintetizador no recibirá velocidad de desactivación del teclado. La velocidad de desactivación de teclado no es frecuentemente utilizada. Unos de los sintetizadores que poseen esta función, es el sintetizador Xpander y Matrix 12 de Oberheim.
El D-50 transmite y recibe velocidad de teclado del 0 al 127. Pero no transmite ni recibe velocidad de desactivación del teclado.
PRESION DE TECLADO (after touch): La presión de teclado se divide en dos formas y son:
a) Presión de teclado polifónico (key). Donde cada tecla contiene su propio sensor de presión para transmitir diferentes niveles de presión. Esta función se utiliza muy poco por lo costoso de las partes. Uno de los sintetizadores que los utilizan es el EPS de la compañía Ensoniq. Por esta razón, en la mayoría de las tablas de implementación "transmitida" y "recibida" están marcadas con una "x" significando que no reciben ni transmiten presión de teclado polifónico.
b) Presión de teclado monofónico (chanel pressure). Este tipo de pressión de teclado es el más utilizado por la compañías fabricantes. Como su nombre lo dice únicamente transmite un nivel de presión, es decir, si al tocar un acorde un el teclado, el dedo que esta aplicando más presión en el acorde va a establecer la cantidad de presión enviada a los instrumentos esclavos.
Existen sintetizadores que no pueden transmitir el mensaje de presión de teclado y únicamente pueden recibirlo, especialmente los módulos generadores de sonido, los sintetizadores sin teclado. En algunas tablas de implementación puede notarse que en las columnas de información "recibida" y "transmitida" en lugar de utilizar los símbolos para indicar si el sintetizador responde o no a los mensajes descritos (0 y "x"), ellas se muestran con un asterisco (*) El significado de éste siempre se encuentra en la parte inferior de la tabla donde se describe el símbolo. En este caso significa que la presión de teclado puede activarse y desactivarse en forma manual por medio de los controles en el panel frontal del instrumento.
El D-50 no transmite ni recibe presión de teclado polifónico, pero si pude enviar y recibir presión de teclado monofónico que puede ser activado y desactivado. Este se puede programar y almacenar en memoria.
VARIACION DE TONO (pitch bend): La mayoría de los sintetizadores transmiten y responden a esta función. Esta función es con frecuencia activada y desactivada manualmente por medio de un botón en el panel frontal de los aparatos. Por eso notarán el asterisco en las columnas de información "transmitida" y "recibida" de la tabla de implementación. En la columna de información "recibida" algunas veces se encontrará el rango de variación de tono. Estas variaciones pueden ser programadas para que se transmitan y reciban en semitonos, cuartas, octavas, etc. Por lo general se especifican en semitonos del 0 al 24.
En la cuarta columna, que es la de los comentarios o instrucciones especiales, se encuentra la cantidad de precisión en la variación de tono. Cuando la precisión no es buena, entonces el cambio del sonido no se escuchará continuamente, sino que se escuchará como una clase de sonido escalonado. La mayoría de sintetizadores utilizan una precisión de 7 a 9 bits. La forma en que se describe en las tablas de los productos en el mercado será de "9 bit resolution" o "7 bit resolution" y en otras ocasiones será en forma de rango del 1 al 64 en lugar de forma de bits.
En el D-50, la función de variación de tono puede ser activada y desactivada manualmente al igual que puede ser almacenada en su memoria. El rango de variación de tono es de 12 semitonos (0-12), y la precisión es de 9 bits.
CAMBIO DE CONTROL (control change): En la primera columna se encuentran los números de los controladores ya anteriormente asignados por la MMA y el JMSC que pueden ser transmitidos y recibidos. Estos controladores se refieren a la rueda de modulación, contolador número 1, el pedal controlador de volumen, controlador número 7, etc. En algunos instrumentos estos controladores se pueden asignar al número que el usuario desee, dependiendo de la aplicación, es decir, que la rueda de modulación no siempre tiene que ser el controlador número 1, este puede ser el controlador número 4, 25, 46 o el número 100. El procesador de señales de audio de la compañía Alesis llamado Quadra Verb puede asignar cualquier número de controlador para modular la cantidad de reberveración con el pedal controlador para modular la cantidad de reberberación con el pedal controlador de volumen. En la columna de información "transmitida" de la tabla de implementación, se indica que controladores se transmiten, cuáles no se transmiten y cuales se pueden activar y desactivar manualmente. Lo mismo es en la de información "recibida", en la de comentarios por lo general se especifica el nombre del controlador que corresponde al número de la primera.
El D-50 puede transmitir los siguientes números de controladores: 1, 5, 64 y 65 puede ser activados y desactivados de forma manual; de 0 al 31 se pueden transmitir, al igual que del 64 al 95; el 6, 38, 100 y 101 no se transmiten. Por otro lado el D-50 puede recibir los controladores 1, 5, 7, 64 y 65 que se pueden activar y desactivar manualmente; los controladores números 0,2,3,4 del 8 al 31 y del 66 al 95 se pueden recibir. La recepción de los controladores número 6,38,0 y 1 son casos especiales del D50 para una explicación de ésto en el manual de instrucciones correspondiente.
CAMBIO DE PROGRAMA (program change): El mensaje de cambio de programa es un poco complicado por la razón de que no todos los fabricantes de sintetizadores utilizan el mismo sistema de numeración de los programas o sonidos. Cuando esto sucede es necesario organizar los programas de tal manera de que al cambiar del programa del 1 al 40. El DX-7 únicamente cuenta con 32 programas en su memoria así que debe utilizarse un dispositivo que pueda organizar el orden de los programas. A la organización de programas se le conoce con el nombre de "mapping".
En la columna de información "transmitida" se encuentra el número de programas que el sintetizador en particular puede transmitir. Generalmente se pueden transmitir 128 progra-

mas del 0 al 127. Lo mismo es en la columna de información "recibida".
El numero real de programas que pueden ser transmitidos y recibidos por un instrumento están en esta subdivisión de la función. En otras palabras, son los verdaderos números de programas que se pueden transmitir y recibir. En algunas ocasiones se podrá observar que los números varían de instrumento a instrumento, por ejemplo, del 0 al 99, del 11 al 88, etc. El D-50 puede recibir y transmitir los programas del 0 al 127 al igual que el número real.

Continuaremos en la próxima revista con la tabla de implementación MIDI, completándola y revisando los distintos aspectos que reflejados en la tabla gráfica que podéis observar y que serán completados para un mayor entendimiento de la misma.

TABLA DE IMPLEMENTACION MIDI

FUNCION	TRANSMITIDA	RECIBIDA	COMENTARIOS
Canal Básico Designación Cambiado	1-16 1-16	1-16 1-16	Fijado en memoria
Modo Designación Mensajes Alteración de Modo	Modo 3 Poly, Omni desactivado	Modo 1,3,4 Mono, Poly, Omni ON/OFF Modo 2>1	Fijado en memoria
Número de Nota Numeración Real	12-108 ---------	0-127 12-108	
Velocidad Nota Activada Nota Desactivada	0 x9n V=0	0V=1-127 X	
Presión de teclado Polifónico Monofónico	X	X *	
Variacion de Tono	*	*0-12 Semi	Resolución 9 bits
Cambio de control 1 5 7 0-131 6,38 64 65 64-95 100-101	* * * 0 X * 0 X	* * * 0(0,2-4,8-31) ** * * 0(66-95) ** (0-1)	Modulación Tiempo de Porta. Volumen Control Externo Datos de entrada Detección 1 Portamento SW Interruptor pedal
Cambio de Programa Número Real	0-127 ******	0-127 0-127	
Sistema Exclusivo	X	X	
Sistema común .Marcador Posición .Selección Canción . Afinación	X X X	X X X	
Sistema de tiempo real ;Reloj ;Comandos	X X	X X	
Mensajes auxiliares ;Local ON OFF ;Desactivación total ;Sensor Activo ;Restablecimiento	X 0(123) X X	0 0(123-127) X X	Fijado en en memoria
Apuntes	X		

Modo 1: OMNI ACTIVADO, POLIF/MODO 2: OMNI ACTIVADO, MONOF/MODO 3:OMNI DESACT,POLIF/MOD 4:OMNI DESACT, MONOF 0=SI, X=NO

Microsoft© Macro Assembler 6.0

Hemos llegado a un punto donde vamos a realizar nuestro primer programa en ensamblador. En el número anterior e nuestra serie de artículos, vimos la estructura básica de un procedimiento, al cual definíamos como "unidad funcional", es decir un trozo de programa que es capaz de hacer algo por si sólo. Dado que un programa realiza múltiples funciones diferentes, cada una de ellas será, o deberá ser, un procedimiento diferente y a su vez estos últimos, se subdividirán en más y más pequeños procedimientos.

Pero pasemos a la acción con el programa que podéis ver en la página adjunta.

Como podréis comprobar, el programa no es gran cosa, pero nos va a servir para demostrar y enseñar muchos de los aspectos de la estructura de un programa en ensamblador y como se crea y los pasos que hay que seguir para compilarlo y corregir los posibles errores que se hayan producido en la creación del mismo.

Lo que podéis observar aquí no es sino un programa normal y corriente de tipo .EXE. En este formato, y a diferencia de los .COM, el segmento de pila está presente y el comienzo del código puede estar situado en cualquier lugar de nuestro programa, no así en el otro formato en el que el comienzo del mismo ha de estar necesariamente localizado en la posición 0100h, ya que esos 256 primeros bytes están ocupados por el PSP (ver el apartado sobre el MSDOS de este mismo número)

Claramente pueden verse los tres segmentos que siempre váis a utilizar con vuestros superprogramas. En primer lugar el de código que es donde se sitúa nuestro pequeño procedimiento al que hemos llamado NADA, en honor a lo poco que es capaz de hacer, ya que simplemente nos va a sacar el siguiente "display" de pantalla:

```
AHORA EL CONTADOR VALE 0
AHORA EL CONTADOR VALE 1
AHORA EL CONTADOR VALE 2
AHORA EL CONTADOR VALE 3
AHORA EL CONTADOR VALE 4
```

No es gran cosa, ¿verdad?. Pues bien, al arrancar el entorno operativo del MASM 6.0, escribiendo PWB y pulsando ENTER, se nos abre una ventana muy parecida al EDIT del MSDOS, no en vano se trata de la misma compañía creadora de ambas aplicaciones. Damos nombre a nuestro archivo y lo grabamos sin nada, con lo cual nos aseguramos que cada vez que mandemos un SAVE siempre lo hará sobre este archivo y no sobre otro. A continuación ya podemos empezar a teclear nuestro programa. Se pueden elegir la distancia entre tabuladores para hacer cada una de las columnas mayores o menores en tamaño, con lo que nuestro listado tendrá el aspecto y tamaño que nosotros deseemos. Por lo general una distancia de tabulación de 8 caracteres será más que suficiente.

LOS DISTINTOS CAMPOS

Un listado en ensamblador nunca deberá ser una amalgama de líneas sin sentido puesto que ello nos llevará, tarde o temprano, al mas completo de los desastres. Por ello, la utilización de los tabuladores es primordial, no debiéndonos importar el aspecto bonito del listado sino su comprensión y organización.

En la primera columna siempre pondremos las etiquetas. Estas son los nombres propios que nosotros le daremos a cada uno de los "niños" que componen nuestro programa y gracias a las cuales no tendremos que referirnos a posiciones de memoria, que como números que son, siempre se prestan más a la confusión. La etiqueta puede tener hasta 32 caracteres, aunque difícilmente necesitaremos esa longuitud cuando queramos expresar lo que tal o cual parte del programa o variable va a representar. Además, etiquetas tan grandes harán de nuestro programa un listado excesivamente ancho en columnas y con lo que su impresión será tarea que nos reportará más de un problema.

El ancho, a mi gusto, no debe exceder las 120 columnas, permitiendo que se pueda imprimir en una impresora de carro ancho sin que por ello se rompan las columnas, lo cual trae consigo la ilegibilidad del listado.

En el próximo número realizaremos la compilación paso a paso para que no se os escape nada de tan importante proceso.

Por:



Ramón López

aLa mejor herramienta para el programador en ensamblador es su compilador. En esta sección estamos analizando uno de los mejores del mercado. Aprende a manejarlo con nosotros.

```
CODIGO      SEGMENT PARA PUBLIC 'CODE'
            ASSUME CS:CODIGO, DS:DATOS, ES:DATOS, SS:PILA

NADA        PROC FAR
            MOV AX, DATOS         ;Cargamos AX con el comienzo del Seg. de datos.
            MOV DS,AX                    ;Y lo pasamos a DS
OTRO:              INC CONTADOR          ;el contador se incrementa de 0 a 4
            CMP CONTADOR, 5              ;¿Es el contador = 5?
            JZ SALIDA                    ;Sí es así sal del programa
            MOV AX, CONTADOR             ;Carga el contador en AX para cálculos
            SHL AX,1                     ;Multiplícalo por 2
            MOV BX,AX                    ;Ponlo en BX
            ADD BX, OFFSET MENSAJES      ;Suma CONTADOR*2+OFFSET MeNSAJES
            MOV DX, [BX]                 ;Pon en DX el contenido de donde apunta BX
            MOV AH,09H
            INT 21H              ;escribe mensaje en la pantalla
            JMP SHORT OTRO       ; ve a por el siguiente mensaje


SALIDA:     MOV AX,04C00H        ;Salida al MSDOS
            INT 21H
NADA        ENDP
CODIGO      ENDS

DATOS              SEGMENT PARA PUBLIC 'DATOS'
LF          EQU    0AH
CR          EQU    0DH
FIN         EQU    $
ENTER1      EQU     LF,CR,FIN

CONTADOR    DB     0

MEN1        "AHORA EL CONTADOR VALE 0", ENTER1
MEN2        "AHORA EL CONTADOR VALE 1", ENTER1
MEN3        "AHORA EL CONTADOR VALE 2", ENTER1
MEN4        "AHORA EL CONTADOR VALE 3", ENTER1
MEN5        "AHORA EL CONTADOR VALE 4", ENTER1

MENSAJES    LABEL WORD
M1          OFFSET MEN1
M2          OFFSET MEN2
M3          OFFSET MEN3
M4          OFFSET MEN4
M5          OFFSET MEN5
DATOS              ENDS
PILA        SEMENT STACK 'PILA'
            DB     32  DUP ("PILAPROGRAMA")
PILA        ENDS

            END    NADA
```



archivos .M

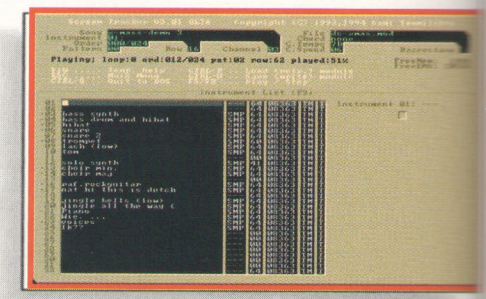
Un estándar de

Después de ver la estructura de cabecera de un archivo .MOD en el número anterior de Código Juego y antes de pasar a una explicación más profunda sobre cómo son y como funcionan internamente este tipo de archivos músicales y sonoros, de gran aplicación en nuestros juegos debido a su enorme calidad, vamos a hacer un alto en el camino para elegir, de entre todas las existentes en el mercado, una herramienta que nos permita editar, visualizar y hacer sonar los MODS.

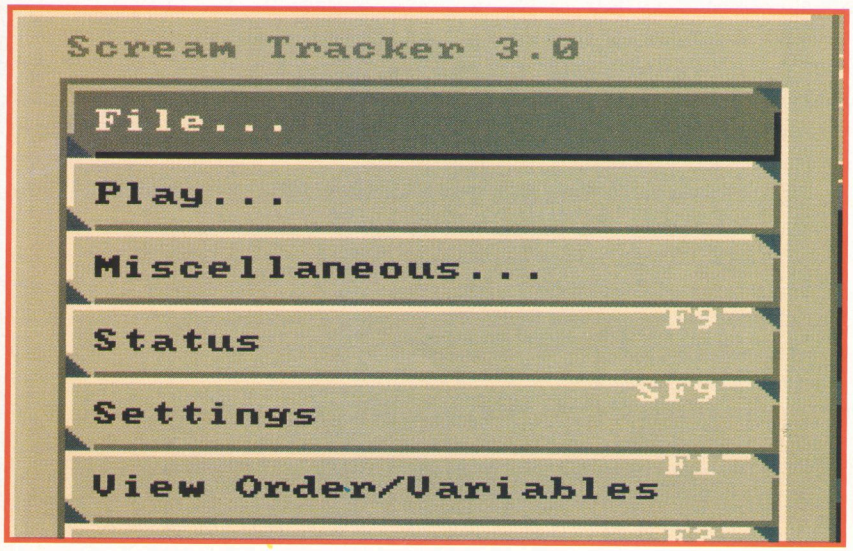
Hay gran cantidad de programas que simplemente "tocan" estos archivos, pero pocos de ellos son capaces de servirnos para crear y editar los patrones, permitiéndonos realizar nuestras propias melodías originales. Por ello, cuando nos encontramos con un programa como el que hoy comento y encima es de dominio público, uno no puede sino quitarse el sombrero y aplaudir, dando las gracias a su programador, Sami Tamiletho, por habernos regalado tan fantástica herramienta.

El grupo finlandés FUTURE CREW, muy conocido por la enorme calidad de sus jóvenes programadores, gracias a las demos que hay distribuido gratuitamente por el mundo como "Unreal" y la más reciente "Second Real", ganadora del certamen ASM´93, ha utilizado la estructura de archivos MOD para todas sus realizaciones, en las cuales el aspecto sonoro siempre ha sido digno de resaltar, conjuntamente con la calidad gráfica de las mismas.

En un pequeño archivo de texto, que va incluido en el paquete de la segunda de las demos que he comentado, ellos mismos hacían saber que el programa de creación musical que habían utilizado para desarrollar la música de "Second Real" había sido, precisamente, Scream Tracker 3.0. Este programa ya era conocido en sus versiones anteriores, siendo utilizado masivamente por propios y extraños. Lo que



Scream Tracker 3.0 versión Beta es el mejor programa de edición y creación de archivos MOD que hemos visto. Es potente, rápido y lleno de prestaciones que nos permiten escuchar, corregir, cambiar nombres de samplers, canciones, cortar, copiar, pegar, incluir nuevos instrumentos, etc. Su manejable sistema de menús lo hacen ideal para trabajar con la música y los sonidos que después vamos a incluir en nuestros juegos. Apúntate a la moda del MOD con STracker.

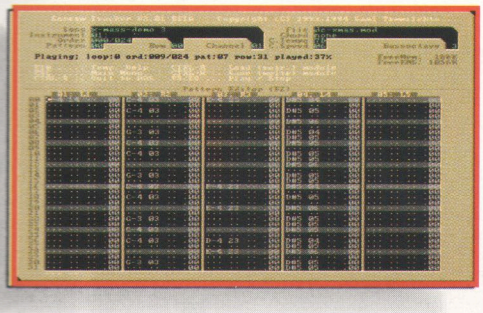


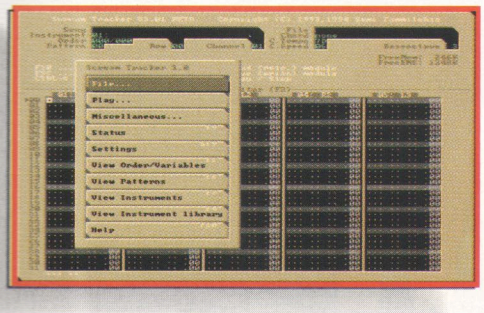
en formato MOD

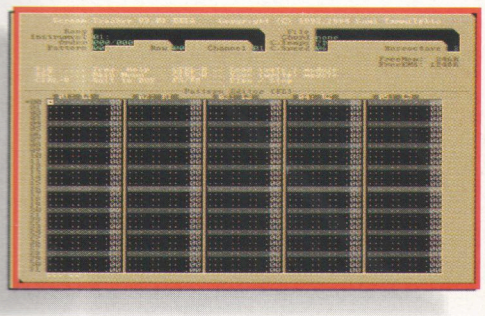
de calidad

Por: Ramón López

El formato musical y sonoro .MOD se está convirtiendo en un estándar para la inclusión de música en los juegos actuales. Por ello, debes aprender cual es el formato de los datos que conforman este tipo de archivos.







ocurría era que las limitaciones eran grandes. A un interface difícil de entender se unía un sistema de menús críptico y enrevesado. Pero todo ha cambiado con esta nueva versión.

SCREAM TRACKER 3.0

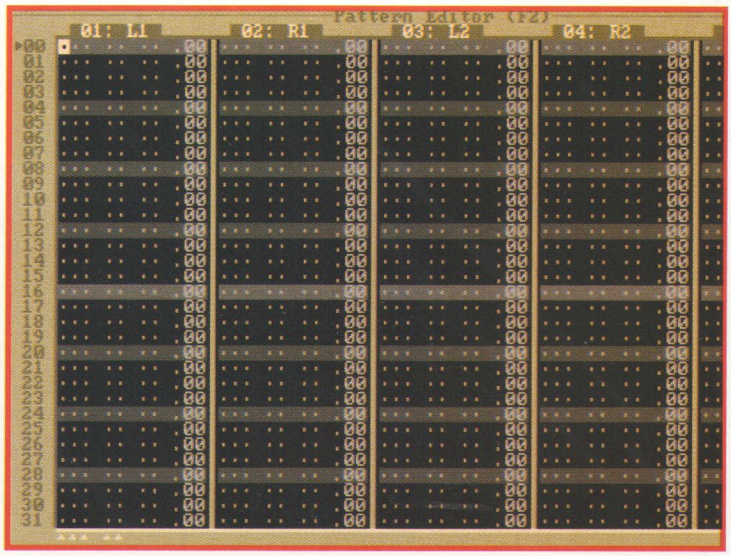
Aunque se trata de una versión beta del producto, el programador hace incapié en que no dispone de tiempo para mejorarla, por lo que, por ahora, la version final no se otéa en el horizonte. De todas formas, salvo un pequeño error que le hemos detectado, el resto del programa funciona a las mil maravillas. En las pantallas podéis ver sus posibilidades y muchas palabras sobran.

Sí os diré que podemos editar absolutamente todos los aspectos de un MOD ya creado con anterioridad, así como añadir lo que nos venga en gana a uno que estemos haciendo nosotros mismos.

Se puede variar el volumen general de la canción si deseamos que suene más fuerte en nuestros altavoces, ya que no todos tenemos la SoundBlaster conectada a un amplificador de sonido. También, en uno de los apartados podemos variar el nombre de la canción, autor, nombres de los samplers, etc. Los samplers mismos pueden ser cambiados, haciendo que una trompeta cambie a un saxofón, por ejemplo. Situándonos en el lugar correcto incluso podemos hacer que el teclado suene al pulsar las correspondientes teclas como el sampler que elijamos. Se accede fácilmente a las bibliotecas de sonidos y arrancar una canción es tan fácil como la tabla del dos.

En definitiva, merece la pena incluirlo en nuestro disco duro y disfrutar de él, principamente por que admite varios formatos musicales e incluye uno nuevo con el que podremos disponer de hasta hasta cuatro canales estéreo. La calidad en este formato es grandiosa, aunque es necesario tener una máquina veloz, 486, para que no se produzcan los consabidos chispazos sonoros que a nadie le gusta. De todas maneras, los archivos MOD los hace sonar como un verdadero virtuoso.

Este programa se incluye en el superdisco número cuatro, para aquellos que estéis interesados.



Nuevo CÓDIGO JUEGO

La primera revista del mundo que enseña a programar videojuegos

Eres el último soldado

Código Juego te ofrece la oportunidad

y tienes que escapar de un mundo lleno

de conseguir el primer Kit completo

de enemigos. Un infierno plagado de

de programación del juego que más

seres que sólo tienen un objetivo...

ha dado que hablar

...acabar contigo

DOOM

Desde este número y en los sucesivos, Código Juego enseñará a sus lectores como programar los niveles de un juego apasionante; DOOM, de la compañía ID Software. Gracias a la utilización de unos editores especiales se pueden confeccionar todos los niveles del juego, añadiendo los objetos, enemigos, habitaciones, escaleras mecánicas, puertas, paredes, armas, vidas, etc.

En este paquete especial de cinco disquetes de alta densidad, no sólo encontrarás los editores más usados para tal fin, sino muchos niveles creados por otros programadores y que son inéditos en España, con lo cual te vas a divertir de lindo creando con tu imaginación al tiempo que te diviertes todavía más con DOOM. Pídelo al número de teléfono 956-840246 al increible precio de 2500 pesetas mas 750 pesetas de gastos de envío y podrás seguir el curso de Código Juego. No te lo pierdas, o te arrepentirás. Nota: el juego DOOM no está incluido en estos disquetes y es necesario tener una copia del mismo para que los editores funcionen. Los envios se realizan por agencia de transportes.

El contestador



Todos los meses, en este apartado, queremos echarte una mano para que los problemas con tu ordenador no te quiten el sueño. Envíanos tus dudas a: Código Juego. Apdo 114. 11520 Rota (Cádiz)

Por:

Joan Esquerra

Solucionamos tus dudas

De todas las peticiones de contestación a dudas que hemos ido recibiendo en nuestra redacción, hemos seleccionado las siguientes por su contenido educativo. Son las 7 magníficas. Rogamos nos disculpen aquellos que enviaron más de una pregunta, pero en la mayoría de las ocasiones éstas eran excesivamente complicadas para los primeros pasos que estamos dando en el contestador. De todas formas, quedan archivadas para más adelante. Ya sabéis que la principal intención de esta sección es, al igual que el resto de la revista, enseñar cosas utiles y no tanto como el hecho de solucionar varios problemas a una persona en particular. Aún así, esperamos todas las cartas con todas la preguntas a las que queráis encontrar respuesta.



1) No entiendo muy bien que son los "puertos" de comunicación del ordenador, ¿Podríais explicármelo? Julio Escafoz, Mallorca

2) Tengo un compilador de Microsoft para ensamblador, pero no encuentro el PWB ni el BROWSER del que habláis en la revista, ¿tengo el compilador erróneo? José Joaquín Gonzalez, Lérida

3) ¿Cómo puedo utilizar el ratón en mis propios juegos? Juan Bernal, Puerto de Sta. María

4) ¿Puede una sola persona realizar un juego íntegramente? Fernando Amor, Leon

5) Me gustaría contactar con gente que supiese de juegos y que esté dispuesta a enseñar, ¿sabéis de lugares donde puedan existir grupos de programación de este estilo? Domingo Arial, Burgos

6) Yo creo que Código Juego necesitaría de un soporte o "línea caliente" para todos los que compramos la revista, de esta forma siempre tendríamos un lugar a donde acudir en caso de atasco... Ramón Fenedes, Barcelona

7) Como puedo hacer para detectar la colisión entre un "muñeco" y otro en la pantalla, por ejemplo, cuando la una bala o proyectil toca a un enemigo. Teresa Martinez, Burgos



1) El ordenador que tienes en tu habitación o en el la oficina de tu lugar de trabajo, está compuesto de muchas pequeñas partes, todas ellas desempeñando una función lógica dentro del conjunto del ordenador. Pero tan sólo una de ellas es capaz de llevar las riendas de todos estos caballos salvajes; el microprocesador o unidad central de proceso. Esta, como ya seguro todos sabéis, es una pastilla de silicio que reune la gran mayoría de las funciones de cálculo, ejecución y control de nuestros ordenadores. He dicho la gran mayoría, porque no todas la tareas son realizadas por él, hasta ahí podíamos llegar. Y es que si así fuese, ¿para que harían falta el resto de los chips de apoyo, como por ejemplo el coprocesador matemático? La CPU tendría entonces un tamaño evidentemente más grande un entonces si que sería ciertamente "un ordenador en un chip", que es como también se ha dado en llamarlo. Pero esto no es así. En primer lugar porque los microprocesadores costarían una fortuna, y en segundo lugar porque no se podrían incluir elementos nuevos al hardware del ordenador. Gracias al cielo, los desarrolladores técnicos del PC siempre han pensado en la posibilidad de incluir periféricos al ordenador de manera que éste gane en potencia y versatilidad. Pues bien, si sabemos que el microprocesador es el controlador, ¿cómo se comunica éste con los distintos elementos hardware del PC?, a través de los puertos de entrada y salida (E/S). En realidad son eso, puertos o muelles de carga, donde la CPU deja un dato y el periférico lo toma y viceversa. De esta forma puede haber una comunicación bidireccional entre la CPU y la Soundblaster, impresora, tarjeta de comunicaciones, puertos serie.

LOS PUERTOS DE ENTRADA Y SALIDA (E/S) SON COMO MUELLES DE CARGA DONDE SE DEPOSITA Y SE RECOGE INFORMACIÓN.

2) Vamos a ver si nos entendemos. De todos los ensambladores que existen en el

El contestador



mercado provenientes de la marca Microsoft Corporation, la gran mayoría de ellos, es decir de las versiones MASM 1.0 hasta la 5.1 inclusive, ninguno de ellos ha destacado por se "user friendly" como dicen los americanos, o sea que no han presentado un interfaz "amistoso con el usuario" o de fácil manejo. Por el contrario, todas la órdenes que queríamos pasar al compilador debian introducirse en una larga y difícil cadena de interruptores del tipo "/x". Entonces, la compañía norteamericana decidió dar un cambio a este olvidado lenguaje mediante el uso de un interfaz más agradable. Por ello, con la llegada de MASM 6.0, se añadieron novedades como PWB y BROWSE que nos mencionas en tu pregunta. Efectivamente tienes un compilador anterior. Fíjate cuando lo arranques desde la línea de comandos, ya que en el mensaje de entrada del programa te dice la versión que tienes. Los diferentes ensambladores ha ido incluyendo mejoras, pero siempre han mantenido la compatibilidad descendente.

3) El uso del ratón está limitado en los juegos de ordenador. Por su idiosincracia no constituye un buen elemento para el manejo de los personajes o elementos de la gran mayoría de los juegos de ordenador que salen al mercado hoy en día. Aún así, existe "otro" tipo de juegos en los que sí encuentra su rinconcito. Respondiendo a tu pregunta, te diré que para llegar a programar bien el ratón hay que experimentar un poco y no sólo una ayuda técnica puede valernos. Te remito al artículo que se va a publicar en Código Juego 6 sobre el soporte de este dispositivo desde lenguaje ensamblador. Toda la rapidez y potencia de este lenguaje al servicio de un periférico que se ha convertido en uno de los más usados en el entorno PC.

4) La respuesta es difícil y siempre va en relación al entorno de mercado en el que nos vemos inmersos. Si observas juegos de la categoría de MEGARACE o THE 7TH GUEST, te das cuenta inmediatamente que no hay posibilidad de llegar a hacer un proyecto de tal envergadura uno persona sola. Sin embargo, existen muchos juegos en la historia del PC, que se han realizado por una única persona. Pero pensad en la cantidad de requisitos que alguien así debería reunir: no tener novia o novio, no trabajar salvo en eso, saber programar muy bien en Ensamblador y C, saber dibujar bien, tener oido musical y una voluntad de hierro. Son muchas condiciones, ¿no os parece? Lo mejor es reunir un grupo de 4 ó 5 personas, no más, y entre todos realizar algo de calidad, sin que por ello la cabeza se te ponga tan cuadrada como la placa de un ordenador. Hace falta un músico, un dibujante y al menos dos programadores que hagan que todo el proyecto se pueda ejecutar en un ordenador. Ah, y eso sí, muchas ganas de trabajar sin esperar nada a cambio, al menos al principio.

✦✦✦✦✦✦✦✦✦

EXISTEN EN ESPAÑA MÁS GRUPOS DE PROGRAMACIÓN DE VIDEOJUEGOS DE LOS QUE TE PUEDES IMAGINAR...

✦✦✦✦✦✦✦✦✦

5) Para serte franco, y más ahora desde que hemos comenzado la publicación de Código Juego, te diré que existen en España más grupos de programación de videojuegos de los que tu te imaginas. Uno se cree que aquí sólo está Dinamic y Topo Soft, pero deberíais conocer la cantidad "mogollónica" de pequeños y grandes grupos de personas que se han unido con la esperanza de ver su juego en todas las ludotiendas españolas. Por el contrario, es difícil acceder a ellos. Están muy ocupados en sus proyectos particulares y no pueden perder tiempo enseñando a los demás. Lo entendemos. Recientemente, hemos intentado contactar con el grupo de maravillosos programadores FUTURE CREW para ver si querían contribuir con algún artículo en la revista. Nos han argumentado que tienen demasiado trabajo para podernos ayudar. Una salida torera, sí señor.

6) Estamos de acuerdo contigo. De hecho ya es algo que está sobre el tapete de la mesa de reuniones de Código Juego, pero no nos chafes la sorpresa, que en su momento vamos a ofrecer el sistema de ayuda para programadores más alucinante que jamás hayas visto, eso sí, vete buscando un módem, aunque sea debajo de las piedras.

7) Bueno, bueno. A un programador como yo, ya le estaba pareciendo muy extraño, no haber recibido una carta de este estilo. El tema al que tu haces referencia, se le suele llamar técnicamente control de colisiones. Es decir un objeto que circula por un mundo insustancial como es la RAM de vídeo del ordenador, y que para colmo de males es, o se supone que es, sólido. Por lo tanto, cuando colisiona con otro objeto, también sólido debe ocurrir algo; explosión, muerte, rebote, etc. Pero, ¿cómo controlar el momento en que un elemento móvil del mundo virtual creado por nosotros choca con otro? Este es un tema, que entre programadores profesionales ha tenido mucha literatura. La verdad es que hay muchos sistemas, más o menos imaginativos, para detectar la colisión, pero los más importantes son dos. Por un lado, la selección de colores y por otra el uso de mapas con mayor o menor resolución. Lo más importante es saber programar esto de forma que se detecten de manera rápida y que no por ello enlentezcan el resto del juego. En su momento dedicaremos una sección entera a los distintos métodos que existen.

El problema, sin embargo, es más difícil de solucionar de lo que puede parecer en un principio, ya que algunos métodos son rapidísimos, pero a costa de que el ojo humano se percate de que no existe una colisión real entre los objetos. ¿Cuántas veces no os habéis dado cuenta de que una bala os quitó una vida y sin embargo juraríais que ni tan siquiera os rozó un pelo?. Yo diría que muchísmas veces.

De todas formas, es un tema que requiere bastante trabajo de estudio y en el juego de MOOKIE, se verá en su momento.

Muy bien, esto es todo por este mes. Seguid enviando vuestras cartas.

Joan Esquerra.****

Ahora, los archivos de

CÓDIGO JUEGO

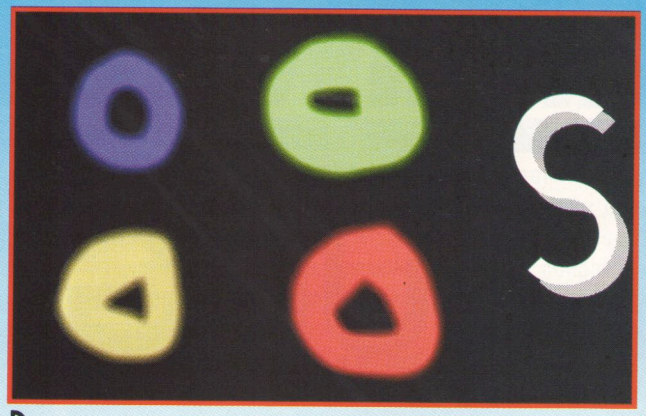
Ya están disponibles

en la BBS TARSIS

Si tienes un módem, ahora es muy sencillo tener todos los listados y programas que aparecen en la revista Código Juego. Tan sólo marca el 956-220035 configura tu transmisión a 8 bits, paridad ninguna y bits de paro 1 (8N1). La velocidad puede ser la que tu módem te permita. Tras contestar a las preguntas iniciales sitúate en el área "revista código juego" y podrás llevarte todos los archivos que desees de los listados y programas que mes a mes van apareciendo en la revista. El sysop se llama José Manuel, por si tienes algún problema.

Además, muy pronto estableceremos un Fórum de preguntas y respuestas, así como el acceso a la Fidonet y otras muchas sorpresas. Mantente en onda. horario de la BBS: todas la horas menos de 11 a 1 de la mañana y de 3 a 5 de la tarde.

Aprende a hacer un juego



SIMON

Programación Gráfica II

Por:

Eduard Sánchez

En el artículo del mes pasado vimos como acceder a la memoria de vídeo poniendo pixels de diferentes colores. Pero lo realmente imprescindible al programar un juego es el hecho de poder cargar ficheros gráficos. En el número anterior también os expliqué el tipo de ficheros gráficos que usa SIMON, así como su estructura, por lo tanto no voy a contarlo de nuevo. El siguiente listado en lenguaje 'C' es usado en SIMON para cargar los ficheros Jovain VI, veámoslo a continuación:

```
#include "stdlib.h"
#include "dos.h"
#include "stdio.h"
#include "conio.h"
#include "fcntl.h"
#include "mem.h"

#define BYTE unsigned char
#define WORD unsigned int
#define BOOL int

void set_video_mode(unsigned char num)
{
        asm {
                xor ah,ah
                mov al,num
                int 10h
            }
}

WORD EstableceSegmento(void)
{
WORD segment,err=0;

asm {
    mov ah,48h
    mov bx,4096
    int 21h
    jc mal
    mov word ptr segment,ax
    jmp salir
    }
mal:;
asm mov word ptr err,bx

salir:;

   if (err != 0)
     {
     set_video_mode(0x3);
     printf("Error asignando Segmento.\n");
     exit(3);
     }

   return segment;
}

void LiberaSegmento(WORD segment)
{
asm {
    mov ax,segment
    mov es,ax
    mov ah,49h
    int 21h
    jc mal
    jmp salir
    }
mal:;
    set_video_mode(0x3);
    printf("Error liberando segmento.\n");
    exit(3);
salir:;
}


void rgb(BYTE desde,WORD ct,WORD pal)
{

asm   {
      push ds
      pop es
      mov di,pal
```

Nota: El programa SIMON fue incluido en el disquete de CJ2

```
mov ax,3
      mul ct
      mov cx,ax
      }
resynch:;
asm   {
      mov dx,3dah
      in al,dx
      test al,8
      jz resynch
      mov al,desde
      xor ah,ah
      add di,ax
      add di,ax
      add di,ax
      cli
      mov dx,03c8h
      out dx,al
      inc dx
      }
otroreg:;
asm   {
      mov al,[es:di]
      out dx,al
      inc di
      loop otroreg
      sti
      }
}

void error()
{
        set_video_mode(0x3);
        printf("\nError abriendo archivo.");
        exit;
}

void graf(char nom[])
{

int fitxer;
unsigned char PALETA[768];
unsigned char SKIP[16];
BYTE far *ptr;
WORD Seg;
unsigned bytes_read;

        if ( _dos_open(nom,O_RDONLY,&fitxer) != 0
) error();

        _dos_read(fitxer,&SKIP,16,&bytes_read);

_dos_read(fitxer,&PALETA,768,&bytes_read);

        rgb(0,256,(WORD) &PALETA);
        Seg=EstableceSegmento();
        ptr=MK_FP(Seg,0);

_dos_read(fitxer,ptr,64000,&bytes_read);
        movedata(Seg,0, 0xa000, 0, 64000);
        _dos_close(fitxer);

        LiberaSegmento(Seg);
}

main()
{
        set_video_mode(0x13);
        graf("simon.000");
        getch();
        set_video_mode(0x3);
}
```

Ahora vamos a estudiarel listado detenidamente, rutina por rutina:

La rutina 'set_video_mode' nos permite cambiar el modo gráfico de la tarjeta VGA. En el modo 0x13 (13h) pasamos a 320x200 256 colores (para un examen más completo sobre las posibilidades de la tarjeta VGA, referiros a la serie de artículos que sobre este dispositivo se realiza en Código Juego). El funcionamiento de esta rutina ya fue explicado en los números anteriores de CJ, y su funcionamiento se basa en el uso de la interrupción 10h.

Recordemos que la imágen gráfica nos ocupará un total de 64000 bytes, por lo tanto necesitaremos un segmento enterito de la memoria para poder alojarla. De eso se encarga la rutina 'EstableceSegmento'. 'EstableceSegmento' hace uso de una de las funciones de la interrupción 21h para obtener un segmento de la memoria que no esté usado. Esta rutina nos devolverá, en un WORD, el número de segmento libre, que usaremos para almacenar la imágen. Si hubiera un error al intentar buscar un segmento libre, la rutina nos devolveria un mensaje de error.

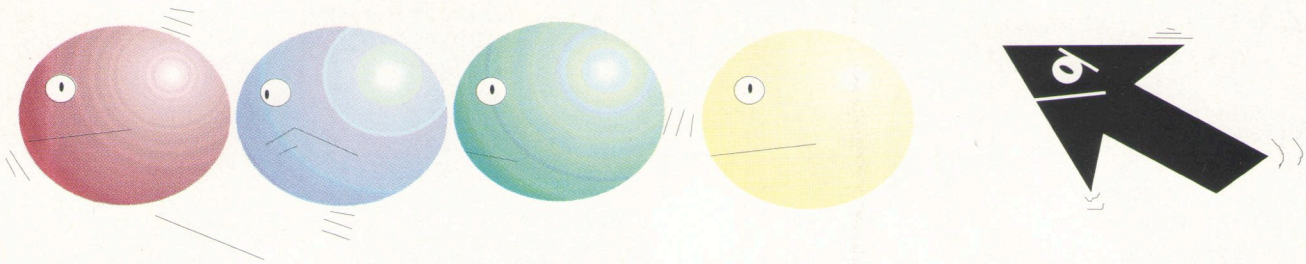
La operación inversa a la anterior la realiza 'LiberaSegmento'. Ésta lo que hace es desalojar el segmento que nosotros hayamos usado para guardar la imágen. A esta rutina se le envía un WORD que es el número del segmento a desalojar.

La rutina 'rgb' se encarga de establecer la paleta de colores contenida en el fichero gráfico. Para ello se usa el puerto 3C8h, al cual se le van enviando los distintos valores RGB de cada uno de los 256 colores de la paleta. El sistema RGB ya fué explicado en el número anterior de CJ, así como el funcionamiento del puerto 3C8h al cual se hace referencia en el listado de MOOKIE en el primer número de CJ, por lo tanto creo que queda suficientemente claro el funcionamiento de la rutina. También se podría definir toda la paleta mediante la interrupción 10h de la BIOS pero, como siempre os digo, es mejor hacerlo todo en ensamblador.

Pero la más importante es 'GRAF'. Que es la que se encarga de cargar el contenido del fichero y pasarlo a la memoria de vídeo. 'graf' abre el fichero que se le envia en la variable 'nom'. Si hay algún error al abrir el fichero se sale con un mensaje de error. Si no se procede a lo siguiente:

* Saltar la cabecera leyendo la variable SKIP (de 16 bytes)
* Leer los 768 bytes de la paleta RGB (256*3 = 768) y luego definirlos con el uso de la rutina 'rgb'.
* Liberar un segmento. El número del segmento liberado se almacena en la variable 'SEG'.

SIMON EL JUEGO DE LOS COLORES



* Luego se crea un puntero mediante el segmento liberado, de esa manera ya tenemos una variable (ptr) apuntando a un segmento libre.
* Leer los 64000 bytes (320x200) del fichero y guardarlos en 'ptr'.
* Mover 64000 bytes de la posición de memoria SEG:0 (donde hay la imágen almacenada) a A000:0000. Esto se hace mediante la instrucción 'movedata'. También se podría mover la imágen mediante la siguiente rutina en ensamblador:

```
asm  {
        mov ax,0a000h
        mov es,ax
        xor di,di
        mov ax,word ptr Seg
        mov ds,ax
        xor di,di
        mov cx,64000
        cld
        rep movsb              // De DS:[SI] a ES:[DI]
}
```

* Entonces se cierra el fichero y se libera el segmento almacenado, mediante la rutina 'LiberaSegmento'.

Eduard Sànchez Palazón

Nuevo CÓDIGO JUEGO

La primera revista del mundo que enseña a programar videojuegos

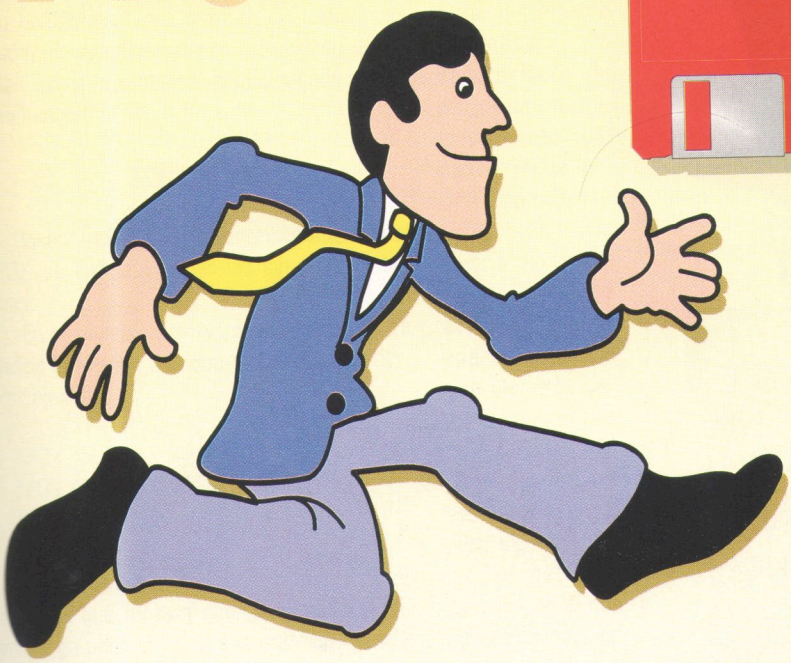
CON LA PEQUEÑA EMPRESA DE INFORMÁTICA



Si tienes un pequeño negocio de informática y estás interesado en anunciarte Código Juego, no lo dudes. Nuestros precios publicitarios son los más competitivos del mercado.
Nuestra distribución nacional nos coloca en una inmejorable situación para que tu anuncio llege a muchos hogares españoles.
Consulta gratuitamente nuestras tarifas especiales por módulos, verás como te asombrarán.
Pide información en el teléfono 956-813129, Joan Esquerra.

CUANDO ESTES DESESPERADO...
CUANDO NO LO ENCUENTRES...
CUANDO TUS AMIGOS NO LO TIENEN...
CUANDO TE GUSTE TENERLO...

NO LO DUDES



CORRE A PEDIR TU SUPERDISCO



956-840246

El Superdisco de Código Juego cuesta tan sólo 950 pts. gastos de envio no incluidos.
Se trata de una colección de programas Shareware y de Dominio Público seleccionados para ayudaros a programar videojuegos.

CODIGO JUEGO

Por:

Ramón Lopez

EL MS DOS

y el programador de videojuegos

5ª PARTE



Hemos sentado las bases de un conocimiento exhaustivo del PC y su sistema operativo. Tomemos ahora dos cuerpos de ventaja sobre nuestros adversarios conociendo a fondo los servicios de interrupción de nuestro hermano invisible,el MSDOS.

LAS FUNCIONES DOS

Para que todos nuestros programas, juegos o no, puedan utilizar las rutinas del kernel del DOS, que vimos en los números anteriores de donde proviene y donde se asienta en memoria, vamos a echar mano de la interrupción 21h, que, como estudiamos previamente, es la puerta de entrada al sistema operativo. Con la enorme cantidad de funciones que el sistema operativo puede ofrecernos, si se le asignase un número de interrupción a cada una de ellas, hubiésemos gastado gran parte de los posibles 256 números con los que contamos para acceder a ellas (recordad que la tabla de comienzo de interrupciones del PC, situada en la dirección 0000:0000 permite sólo 256 interrupciones.) Por eso, se decidió usar tan sólo una, la 21h, y a partir de ahí las demás serán funciones son subfunciones de ésta. Así tenemos Función 21h, subfunción 10h, con lo cual indicamos que accedemos a servicio 10h de la interrupción general del DOS. En ocasiones, esta subfunción todavía es capaz de ramificarse más, ofreciendo servicios extra, los cuales iremos viendo en su momento.

CONVENCIONES

Para llamar a cualquier interrupción del DOS se deben guardar las siguientes normas:

En el registro del microprocesador AH se indica la función a la que queremos acceder. Por ejemplo:

```
MOV AH, 4Ch ;salida al DOS
MOV AL, 00h ;ERRORLEVEL=0
INT 21h
```

En el ejemplo anterior, el registro AL no indica a que función queremos acceder sin embargo pasa un dato a la interrupción 4Ch que es la encargada de salir al sistema operativo. En el ejemplo siguiente, AL sí sirve como medio para indicar a que subfunción vamos acceder:

```
MOV AH, 44H; Control Entrada/Salida
MOV AL, 01h; Subfunción Datos dispositivo
MOV BX, Handle
MOV DX, Palabra de datos de dispositivo
INT 21h
```

Como podéis observar, en este caso en particular, se utiliza la subfunción 01h de la función 44h, cuyo cometido no viene a cuento por el momento y lo usamos como ejemplo simplemente.

Por lo general, salvo estos dos registros, los demás podrán contener información de entrada y datos útiles de salida, por ello lo mejor es mirar el listado de funciones del MSDOS y ver, exactamente, como funciona cada caso. Aquí vamos a comenzar una serie de sobre el uso de cada una de ellas. Vosotros deberéis ver la posible utilidad en vuestros programas. A fin de que todas las funciones del MSDOS se entiendan perfectamente, cada explicación irá acompañada de pequeños programas para la aplicación de esa función determinada.

También os pediría que prestaséis buena atención a este apartado. Ya se que lo hacéis, pero es ahora cuando se va a poner interesante. Es curioso, pero algunas funciones del DOS parecen no tener aplicación ninguna en un programa de juegos, y cuando menos te lo esperas, izas!, te salva la vida o muchas horas de duro "curro". Así que oidos abiertos.

CONVENCIONES

Para la explicación de las funciones del DOS vamos a seguir unas reglas que conviene aclarar antes de empezar.

Cada una de ellas irá precedida de un cuadro sinóptico en el que se mostrarán los datos necesarios que deberemos sumistrar a la función para que actúe de forma correcta. Para ello se explican las entradas y las salidas, su nombre oficial y posteriormente se entra ya en la explicación a fondo de la misma.

Las cruces indican que el valor no se usa.

FORMATO ESTÁNDAR PARA LLAMADAS AL SISTEMA OPERATIVO MSDOS DESDE LENGUAJE ENSAMBLADOR

ENTRADA

Registro AH=función requerida
Registro AL=Subfunción (normalmente)

Otros registros=Datos varios

KERNEL DEL MSDOS
INTERRUPCIÓN 21H

SALIDA

Todos los registros contienen información útil de salida o datosç basura. Todo depende de cada función en particular.

Interrupción 21h

Función: (En registro AH) 00h

Subfunción (En registro AL) XX

Otros registros:

Entrada

CS=Segmento del PSP del programa

Salida

Nada de importancia

Nombre de la función:

Terminar Proceso

DESCRIPCIÓN:

La función 00h se encarga de acabar con el funcionamiento de un programa que actualmente esta funcionando en memoria. Para ello, realiza cuatro operaciones principales.

1) Limpiar los buffers de memoria que el programa a utilizado
2) Restaurar los vectores de interrupción de INT 22h, INT 23h e INT 24h.
3) Libera la memoria utilizada por el programa
4) Devuelve el control al programa padre.

EXPLICACIÓN:

Aunque Microsoft recomienda la utilización de la interrupción 4Ch en versiones MSDOS 2.0 y posteriores, esta es la única que podemos usar si el ordenador en el que corre nuestro programa tiene la versión 1.0, cosa que resulta más que improbable.

Por ello, siempre es conveniente comprobar cual es la version DOS que funciona bajo nuestro programa, impidiendo la utilización del mismo, si hacemos usos de interrupciones que no estaban implementadas en versiones anteriores. Normalmente, la versión 3.0 ya puede servirnos como apoyo en nuestros juegos, y debemos desautorizar a los usuarios de la 1.0 y 2.0, con un mensaje como: "Por que no actualizas tu sistema operativo de una vez, amigo". Bromas a parte, si el ordenador tiene la versión 1.0 esta es la forma correcta de abandonar nuestro programa y volver al sistema operativo.

EL PSP

Comenzamos a meternos en berenjenales. A muchos de vosotros estas siglas no os dicen gran cosa, ¿verdad? Pues bien, detrás de el PROGRAM SEGMENT PREFIX (PSP) hay mucho, mucho que nos puede interesar, y aunque esta interrupción difícilmente la usemos alguna vez, si nos a servido de iniciativa para explicar qué es y cómo se puede sacar partido al PSP.

El MSDOS, cuando carga un programa en memoria para ser ejecutado, ya sea en formato .COM o .EXE siempre crea una estructura de 256 bytes que coloca justo antes del programa en cuestión. A diferencia de lo que mucha gente piensa, el PSP no es creado por la aplicación que va a funcionar, sino por el mismo sistema operativo.

¿Qué contiene?, pues datos que al MSDOS le conviene tener a mano antes, durante y depués de la ejecución de la aplicación. En la figura podéis ver la estructura y los datos que guardan cada uno de los campos. Para un estudio correcto se necesita casi un artículo como éste, por ello el mes que viene se lo vamos a dedicar a tan importante parte de las convenciones de ejecución de programas bajo entorno MSDOS.

También, veremos en el próximo artículo la siguiente de las funciones del sistema operativo, pero antes incluiremos un ejemplo de como utilizar la de hoy, así como una lección práctica de como sacar partido al PSP, leyendo los datos que el usuario pasa al programa que se va a ejecutar mediante la línea de comandos.

Hasta pronto, ¡feliz DOS!****

Offset: 00 longitud:2 INT 20h
Offset: 2 longitud:2 Dirección del último segmento que se ha apartado o asignado al programa.
Offset: 4 longitud:1 Reservado, valor=0
Offset: 5 longitud:5 Llamada larga (CALL FAR) a el despachador de funciones del Sistema Operativo
Offset: 10 longitud:4 INT 22h
Offset: 14 longitud:4 INT 23h
Offset: 18 longitud:4 INT 24h
Offset: 22 longitud:22 RESERVADO
Offset: 44 longitud:2 Segmento del entorno
Offset: 46 longitud:34 RESERVADO
Offset: 80 longitud:3 INT 21h, RET F
Offset: 83 longitud:9 RESERVADO
Offset: 92 longitud:16 FCB 1 por defecto.
Offset: 108 longitud:20 FCB 2, el cual queda sobreescrito cuando el FCB 1 está abierto.
Offset: 128 longitud:127 Cola de comandos

SUBSCRIBIRSE A

CODIGO JUEGO

ES UN BUEN BOCADO



Gracias a los descuentos especiales. Además, te aseguras de recibir en casa todos los meses la revista que más te gusta.

956-840246

HERRAMIENTAS DE PROGRAMACIÓN

DEBUGGERS

Depuradores del codigo

Por:

Barón Rojo Bytes

La buena y correcta utilización de DEBUG nos va ayudar en gran medida a corregir nuestros programas de forma rápida y sin las complicaciones de depuradores más complejos.

Poco a poco vamos conociendo a DEBUG.COM y nos vamos dando cuenta de su "potencia" y de sus limitaciones. Como ya os he explicado anteriormente, cuando más adelante, en este mismo apartado de Código Juego, empezemos a estudiar otras herramientas de depuración de código, como CodeView de Microsoft o Turbo Debugger de Borland, miraréis a DEBUG por encima del hombro, pero nunca lo dejaréis de lado.

Ya sabemos como mirar la memoria con el comando D(ump), hacer un programa en ensamblador, eso sí con limitaciones, con el comando A(ssemble), desensamblarlo con U(nassemble) y otras pequeñas cosillas. Vamos a continuar el estudio de su reducido juego de instrucciones para que veáis que todavía se pueden hacer muchas más cosas con él.

H, aritmética Hexadecimal

El comando H, Hexadecimal Arithmetic, es muy útil cuando se está trabajando con DEBUG de forma interactiva y para todos aquellos que no tienen una calculadora hexadecimal, ya que saca en pantalla el resultado de una suma y una resta hexadecimal de dos números dados, también hexadecimales (ya sabemos que DEBUG sólo funciona con este formato numérico).

El formato de la instrucción es el siguiente:

H número1 número2

Siendo número1 y número2 dos valores hexadecimales comprendidos entre 0000h y FFFFh (0 y 65535 decimal), o sea 16 bits como máximo. Por ejemplo:

H 5497 0100

dará como salida:

5597 5397

Si la suma de los dos números produce un OVERFLOW, palabra inglesa que viene a significar algo así como "sobrecarga") porque el resultado sea mayor de 16 bits, DEBUG mostrará los 4 dígitos menos significativos en pantalla.

Un pequeño gran hombre
DEBUG.COM

H FFFF 0004

dará como resultado:

0003 FFFB

y si el segundo número es mayor que el primero el resultado será de complemento a dos. Por ejemplo:

H 0001 0002

dará:

0003 FFFF

CARGA DE SECTORES

Una de las mejores posibilidades que tenemos gracias a DEBUG es la de cargar los sectores de disco de una forma sencilla y sin complicaciones, con lo cual podemos acceder a los rincones más recónditos de nuestro disco duro o disquete y hacer los cambios que queramos, eso sí, con mucho ojo de no dañar nada.

El formato de esta instrucción es:

L dirección drive comienzo número

donde

Dirección: Localización de memoria donde queremos colocar la información que traigamos desde el disco.

Drive: Número lógico de la unidad desde la que vamos a cargar (A=0, B=1, C=2, etc)

Comienzo: Número Hexadecimal del primer sector lógico a cargar

Número: Número de sectores consecutivos a cargar desde el que indicamos como inicio.

Un ejemplo:

L 100 1 0 10

✦✦✦✦✦✦✦✦✦

DEBUG TAMBIEN PERMITE CARGAR SECTORES Y GRABARLOS POSTERIORMENTE, UNA VEZ QUE HEMOS REALIZADO LOS CAMBIOS NECESARIOS EN NUESTRO PROGRAMA O ARCHIVO DE TEXTO.

✦✦✦✦✦✦✦✦✦

Cargará en la dirección 0100h del segmento de datos desde la unidad B, 16 sectores comenzando en el sector 0.

L, abreviatura de LOAD o cargar en inglés, permite trabajar con sectores sueltos o con archivos gracias a su interacción con otro comando llamado N(ame) o nombre. Este comando permite pre-nombrar el archivo que después vamos a cargar con L. La siguiente secuencia...

-N PROGRAMA.COM
-L

Cargará el archivo PROGRAMA.COM de la misma forma que si hiciésemos:

C:\>DEBUG PROGRAMA.COM

Lo cual ya hemos estudiado en artículos anteriores.

En el próximo número de este interesante artículo vamos a estudiar la forma de introducir datos en la memoria con la órden E(nter) y como podemos grabarlos en disco con la instrucción W(rite), hasta pronto CJmaniacos.

Nuevo CÓDIGO JUEGO
La primera revista del mundo que enseña a programar videojuegos

La Virguería

Ideal para los gráficos

¡Qué fácil resulta capturar tus pantallas!

Por: HUNTER

David Brioso

Este mes iniciamos una serie de artículos que cubrirán el desarrollo de diversas utilidades software con una aplicación directa en el campo de los videojuegos. En muchas ocasiones nos hemos lamentado de no disponer de cierto programa que nos salvaría la vida en un momento dado, y hemos tenido que recurrir a una BBS a altas horas de la madrugada, o a escribir nuestras propias herramientas. Por alguna oscura ley natural, cuando acabamos de retocar la última línea de nuestro programa, que hemos pulido durante semanas, aparece ante nuestros atónitos ojos uno similar, más rápido, más pequeño, con más funciones... en definitiva, acabamos de reinventar la rueda, pero con un pinchazo. Tras el desmayo inicial, un ligero detalle nos lleva a sonreir con maldad: el programa en cuestión tiene algunos fallos, mínimos, pero fallos al fin y al cabo. Y es así como nos animamos a continuar nuestra labor, perfeccionamos nuestro programa, y dejamos que sean los demás los que se irriten la próxima vez.

Esta es una situación que, aunque cómica, ocurre bastante a menudo. Nunca encontramos el programa adecuado en el momento justo. O viceversa. Para remediar en lo posible este problema o, al menos, hacerlo más llevadero, vamos a diseñar nuestras propias utilidades, a medida de nuestras necesidades. Por supuesto, siempre existirá una más compacta, más completa, pero todo es mejorable.

El objetivo primordial de estas herramientas va a ser su versatilidad. Constarán de un esqueleto lo más básico posible, alrededor del cual se estructura el programa. Esta forma de trabajar nos permitirá ampliar, remodelar, reducir, o incluso rehacer completamente el programa, si hubiera necesidad.

El segundo objetivo, no por ello menos importante, es el didáctico. De nada nos servirían estas utilidades si no aprendemos a aprovecharlas o adaptarlas a nuestros gustos. Hemos intentado encontrar unos programas que tengan el mayor nùmero de variaciones posible.

Por último, perseguimos la elaboración de unas herramientas realmente enfocadas a la creación de videojuegos. No vamos a diseñar complicadas bases de datos, ni nada remotamente parecido. La programación en PC soporta el lastre del origen del propio ordenador: la gestión. El PC ha sido, hasta hace cuatro dias, una máquina diseñada para ejecutar programas soporíferos, lo más alejado que se pueda imaginar de un juego. La falta de información técnica sobre los gráficos, sonido, etc, en el PC proviene de ese pasado tan tenebroso. Así que, sin más preámbulos, allá vamos...

La primera utilidad que presentamos es HUNTER, un capturador de pantallas para VGA, que funciona en "casi" cualquier programa (idesgraciadamente, el PC no es el ordenador más compatible!). Se trata de un capturador residente que programa el Controlador de Interrupciones ("Peripheral Interrupt Controller", PIC) para "colarse" en la mayoría de los juegos y aplicaciones gráficas.

El programa en sí tiene solamente dos opciones: grabar la pantalla como un fichero binario "crudo" ("raw", en inglés) o como un fichero LBM (IFF, para los usuarios de Commodore Amiga). El fichero "raw" contiene la información de la paleta de colores en formato RGB ("Red, Green, and Blue", Rojo, Verde y Azul) y el cuerpo de la imagen, los pixels, en este orden. Por lo tanto, el fichero ocupa 64768 bytes, y no tiene ningún tipo de compresión de datos. Quizás os pregunteis para qué sirve un formato tan arcaico, tan burdo. Buena pregunta. Para poder cargar o grabar pantallas sin la complicación que supone un algoritmo de codificación/decodificación. De acuerdo, el fichero ocupa más espacio en disco, pero es la simplicidad absoluta. De cualquier manera, al no ser un formato estándar, no puede ser cargado en un programa para recibir retoques, así que hemos incluido la otra opción, el formato LBM, para que sea intercambiable.

Este tipo de utilidad se emplea tanto para obtener una imagen de un juego y observar algún defecto que no se puede apreciar en movimiento, comprobar cuántos colores de la paleta están siendo usados en un momento dado o, simplemente, "coger prestados" unos cuantos gráficos de otro juego para hacer experimentos. En fin, depende de vosotros el encontrar algunas de las numerosas aplicaciones de este programa y... Aparte del objetivo práctico, es fundamental el didáctico. Con esta utilidad tendremos ocasión de ver de cerca el manejo de las interrupciones, cómo dejar un programa residente en memoria a través del Sistema Operativo, la lectura de los registros de la paleta de colores de la tarjeta VGA, la creación y escritura de ficheros secuenciales binarios a disco, la lectura de la línea de comandos del MSDOS, y unos rudimentos del formato LBM.

Resulta imposible tratar todos estos temas en profundidad en un solo artículo, así que los iremos tocando superficialmente en este número, y los desarrollaremos en profundidad en números sucesivos de la revista, al mismo tiempo que creamos nuevas herramientas complementarias que también nos sirvan para practicar.

Como sugerencias para los lectores más curtidos en el arte de la programación, partes del esqueleto del programa pueden ser reconvertidos en un salvapantallas, una ventana de ayuda, una calculadora, o similar, que aparezcan en pantalla al pulsar determinada combinación de teclas, un programa para buscar "pokes" en los juegos, un rudimentario desensamblador/debugger para espiar determinados pro-

En esta sección es donde publicamos vuestros artículos de gran tamaño. Ya lo sabes, ganarte un buen dinerito y ver tu artículo publicado en Código Juego es tan simple como que nos envíes tu colaboración en un disquete a la dirección:

CODIGO JUEGO
APDO. 114
11520 ROTA CADIZ
Anímate!!!!

gramas, etc. Como resulta evidente, este programa puede dar bastante "juego".

Una última advertencia: este programa residente no funciona correctamente si hay un gestor de Memoria Expandida (LIM/EMS) instalado. Esto sólo afecta en principio a EMM386.EXE, pero es posible que otros gestores también perturben el comportamiento de HUNTER.

Bueno, ya es hora de entrar en materia. El programa en sí se divide en dos partes claramente diferenciadas: por un lado el procedimiento de configuración e instalación, y por otro, el código que permanece residente cuando se devuelve el control al Sistema Operativo.

La primera parte es relativamente sencilla, hasta que la examinamos más de cerca. Si bien el proceso de instalación es muy básico, una simple llamada a una función del MSDOS (que, por cierto, es la misma que devuelve el control al Sistema), hay una parte más oscura. La configuración del PIC resulta, cuanto menos, críptica. Realmente no tiene mucho interés para nuestros objetivos más inmediatos, así que dejaremos su estudio en profundidad para más adelante. Por ahora debería ser suficiente con saber que existen ocho interrupciones generadas externamente (por dispositivos periféricos, como el teclado, la tarjeta controladora de la unidad de diskettes, el temporizador, el reloj de tiempo real, etc) que pueden ser configuradas mediante un procedimiento muy simple para que estén "solapadas" sobre otras ocho interrupciones software. Este grupo de interrupciones, que se encuentran siempre en interrupciones contiguas, está configurado por defecto en las interrupciones 08h-0Fh. Por lo tanto, cualquier programa que desee capturar una interrupción para sí (teclado, temporizador, etc) buscará directamente en dichos vectores. Este detalle tan simple hace que la mayoría de los capturadores de pantallas fallen estrepitosamente. No pueden evitar que el juego o programa en cuestión modifiquen la tabla de vectores de interrupción y desactiven el capturador. Se han probado diversas soluciones a este problema, pero ninguna ha llegado a funcionar correctamente. Aquí se puede ver con claridad cómo un enfoque inadecuado de un problema puede arruinar semanas, incluso meses de trabajo. Un planteamiento sencillo del problema hubiera demostrado que la raíz es que no existe ninguna manera fiable de garantizar que el vector de interrupción capturado para nuestro programa residente vaya a permanecer sin cambios. Nadie acertó durante mucho tiempo en una solución tan sencilla como la que hoy presentamos: si no podemos evitar que alguien modifique los vectores 08h-0Fh, que son conocidos por todo el mundo, cambiemos los vectores de interrupción externos a otro lugar. De esta manera tan elemental conseguimos que nadie los toque, porque aparentemente están solapados sobre unas interrupciones que no se utilizan normalmente.

Para explicar esto con más claridad, vamos a olvidar de momento los detalles técnicos más avanzados. Existen 256 vectores de interrupción en los 80x86, que forman una tabla de saltos a sus correspondientes rutinas. Una buena parte de estas interrupciones están ocupadas por la BIOS y el Sistema Operativo, y otra parte quedan libres para el usuario (al menos, en teoría). Las restantes se reparten entre interrupciones hardware, generadas por los dispositivos externos a los que antes hacíamos referencia, y una mezcla de software independiente, tarjetas de expansión de todas clases... En fin, ese es otro tema.

Como explicábamos antes, cada uno de estos vectores se dirige a una interrupción en concreto. De esta manera, si el procesador tiene que ejecutar la interrupción número 1Fh, buscará el segmento y el offset de la rutina en la dirección de la tabla correspondiente y, a continuación, salta. En el caso de las ocho interrupciones generadas externamente (IRQ, "Interrupt Request") se dirigen a los vectores de interrupción 08h-0Fh, en principio. Decimos esto porque el PIC nos permite elegir en qué bloque de ocho vectores (tiene que ser múltiplo de ocho también) vamos a dirigir estas IRQ. Si "engañamos" a los demás programas, que ciegamente van a buscar estas IRQ en los vectores por defecto, podemos permanecer controlando el teclado y los demás periféricos que nos interesen.

Ahora retomemos el tema principal. Ya sabemos en qué consiste básicamente el proceso de instalación del programa. Hay que configurar el PIC y dejar residente una parte del código que se encarge de gestionar las interrupciones que se produzcan de ese momento en adelante.

Pero hay algo más: el programa tiene un par de opciones, que tenemos que recoger de la línea de comandos del MSDOS. Sabemos, porque Microsoft se ha molestado en enseñárnoslo, que cualquier parámetro que siga al nombre del programa estará copiado en el bloque de memoria del PSP ("Program Segment Prefix", Prefijo del Segmento del Programa), en el offset 0080h. En nuestro caso, nos limitaremos a comprobar si en la dirección CS:0080h hay algo. Si este byte contiene un valor distinto de cero, indica cuántos bytes le siguen, en una cadena terminada con un cero. Esta cadena es transparente para el MSDOS, porque puede contener cualquier combinación alfanumérica, sin ningún sentido necesariamente.

La última parte de la configuración e instalación del programa residente puede resultar también bastante extraña. Consiste en pedir al Sistema la dirección (segmento:offset) de una variable, conocida como INDOS o "Critical Section Flag" (banderín o "flag" de la sección crítica), que es, cuanto menos, enigmática. Los propios programadores de Microsoft no pueden aclararse con este flag, lo cual da una idea de la magnitud del misterio. Este flag indica, ni más ni menos, si el MSDOS se encuentra ocupado con una tarea "sensible", es decir, peligrosa para la integridad de nuestros datos. Si es así, es conveniente dejarlo tranquilo hasta que acabe su cometido. De lo contrario, podríamos utilizar una misma función, acceder a un periférico cuando está grabando en disco... En fin, algo catastrófico.

La razón fundamental para pedir la dirección del flag en este momento, y no desde el propio programa residente, resulta obvia: si no podemos llamar a las funciones del Sistema mientras el flag se encuentre encendido (cualquier valor distinto de cero) ¿cómo vamos a pedirle la dirección del flag?

Si bien la comprobación de este flag resulta imprescindible, y en algunos casos, inevitable, existen algunas lagunas sobre su uso correcto. Microsoft ha variado en numerosas ocasiones la definición y tipo del flag, que ha pasado de ser una palabra a un solo byte, en algunas versiones menores del MSDOS ha desaparecido sin dejar huella, en otras ha cambiado de nombre, etc. Podemos suponer que alguna vez nos explicarán qué significan realmente los valores que puede adoptar el famoso flag. Quizás entonces ya no recordemos qué era el MSDOS...

Bueno, mientras pensáis en todo eso, os recordamos que en el próximo número continuaremos el examen a fondo del programa, y empezaremos a mostrar las aplicaciones y herramientas que pueden crearse a partir de él. ¡Hasta pronto!

David Brioso Santos

```
; ================================
;
; Programa:  HUNTER.COM
;  Version:  1.00
;    Autor:  David Brioso Santos
;    Fecha:  26.07.94 / 03.08.94
; Copyright  (c) 1994 David Brioso
;
; ================================

int_base EQU      0A0h                      ; *01*

segmento_codigo   SEGMENT

                  ASSUME   CS:segmento_codigo,DS:segmento_codigo,SS:segmento_codigo

                  ORG      0100h    ; *02*

inicio_codigo:    jmp      instalacion               ; *03*

buffer_de_disco   db       768 dup (?)               ; *04*

tipo_de_fichero   db       1                         ; Fichero .LBM por defecto

                                                     ; *11*
ruta_de_fichero   db       'PANTALLA.LBM',0  ; Nombre del fichero de destino

hand_de_fichero   dw       0                         ; "Handle" del fichero creado

estado_DOS_off    dw       0
estado_DOS_seg    dw       0

estado_programa   db       0

; =-=-=-=-=-=-= CABECERA DEL FORMATO LBM =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

cabecera_form     db       'FORM'            ; Nombre de la cabecera de IFF/LBM
                  db       0,0,0FDh,030h     ; Longitud total del fichero-8 (FD30h)
                  db       'PBM '            ; Tipo de fichero ("Packed BitMap")
                                             ; *10*
                  db       'BMHD'            ; Cabecera de datos ("BitMap HeaDer")
                  db       0,0,0,14h         ; Longitud de la cabecera de datos (20)

                  db       01,40h            ; Anchura del "brush" = 320 (0140h)
                  db       00,0C8h           ; Altura del "brush"  = 200 (00C8h)
                  db       0,0,0,0           ; Posición X, posición Y
                  db       8                 ; Número de bits por píxel = 8
                  db       0                 ; Máscara
                  db       0                 ; Tipo de compresión;  0 = ninguna
                  db       0                 ; Aspecto

                  db       0,0,5,6  ; Modo de video

                  db       01,40h            ; Anchura de la imagen = 320 (0140h)
                  db       00,0C8h           ; Altura de la imagen = 200 (00C8h)

                  db       'CMAP'            ; Cabecera de paleta (Colour MAP)
                  db       0,0,3,0           ; Longitud del bloque (256 * 3 = 0300h)

cabecera_body     db       'BODY'            ; Cabecera del cuerpo de la imagen
                  db       0,0,0FAh,0        ; Longitud del cuerpo (FA00h)

; =-=-=-=-=-=-= INTERRUPCIONES =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

nuestra_IRQ0:     int      08h                       ; IRQ0: Temporizador programable
                  iret

nuestra_IRQ1:     call     rutina_teclado            ; IRQ1: Gestor de teclado
                  int      09h
                  iret

nuestra_IRQ2:     int      0Ah                       ; IRQ2: Señal de video / esclavo
                  iret

nuestra_IRQ3:     int      0Bh                       ; IRQ3: COM1/3 - COM2/4
                  iret

nuestra_IRQ4:     int      0Ch                       ; IRQ4: COM2/4 - COM1/3
                  iret

nuestra_IRQ5:     int      0Dh                       ; IRQ5: Disco duro / LPT2
                  iret

nuestra_IRQ6:     int      0Eh                       ; IRQ6: Controladora de diskette
                  iret

nuestra_IRQ7:     int      0Fh                       ; IRQ7: LPT1
                  iret

rutina_teclado:   cli
                  cmp      cs:[estado_programa],0    ; ¿Ha sido llamado nuestro TSR
                  jne      proc_activado             ; y todavía está activo?

                  push     bx
                  push     ds
                  lds      bx,dword ptr cs:[estado_DOS_off]
                  mov      bl,ds:[bx]                ; Lee el byte de estado del
                  or       bl,bl                     ; MSDOS; si es distinto a
                  jnz      DOS_ocupado               ; cero, abortar el proceso.

                  xor      bx,bx                     ; Ahora comprobamos si el modo
                  mov      ds,bx                     ; de video es MCGA (320*200*256)
                  mov      bl,ds:[0449h]             ; Esta variable es usada por la
                  cmp      bl,13h                    ; int 10h para guardar el modo
                  jne      DOS_ocupado               ; de video actual (0000:0449h)

                  push     ax                        ; Preserva el registro AX
                  in       al,60h                    ; Lee el byte del teclado
                  cmp      al,58h                    ; ¿Es el código de pulsar F12?
                  jne      cancela_int               ; *07*
                  mov      cs:[estado_programa],1    ; Activa el flag del programa
                  mov      al,20h                    ; EOI: End Of Interrupt. Limpia
                  out      20h,al                    ; el chip de interrupciones
                  call     captura_imagen
cancela_int:      pop      ax
DOS_ocupado:      pop      ds
```

```
                pop     bx
                mov     cs:[estado_programa],0     ; Limpia nuestro flag de estado
proc_activado:  sti
                ret

captura_imagen: push    cx
                push    dx
                push    si
                push    di
                push    ds
                cmp     cs:[tipo_de_fichero],1
                je      pantalla_lbm
                call    graba_raw
                jmp     fin_de_int

pantalla_lbm:   call    graba_lbm
fin_de_int:     pop     ds
                pop     di
                pop     si
                pop     dx
                pop     cx
                ret

; =-=-=-=-=-=-= GRABACION DE RAW =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

graba_raw:      call    crear_fichero              ; Si no puede crear el fichero,
                jc      no_graba_raw               ; devuelve el Carry encendido
                call    graba_paleta               ; Graba 768 bytes al comienzo
                call    graba_cuerpo               ; del fichero. A continuación
                call    cerrar_fichero             ; graba los 64000 restantes
no_graba_raw:   ret

; =-=-=-=-=-=-= GRABACION DE LBM =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

graba_lbm:      call    crear_fichero              ; Si no puede crear el fichero,
                jc      no_graba_lbm               ; devuelve el Carry encendido
                call    graba_formato              ; Graba la cabecera del .LBM
                call    graba_paleta               ; Graba 768 bytes de la paleta
                call    graba_body                 ; Graba 8 bytes de datos "BODY"
                call    graba_cuerpo               ; Graba los otros 64000 bytes
                call    cerrar_fichero             ; Cierra nuestro fichero .LBM
no_graba_lbm:   ret

; =-=-=-=-=-=-= CREACION DEL FICHERO =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

crear_fichero:  mov     dl,0                       ; Unidad de disco por defecto
                mov     ah,36h                     ; Función del MSDOS que devuelve
                int     21h                        ; el espacio libre en el disco:
                                                   ;
                                                   ; AX = Número de sectors/cluster
                                                   ; BX = Número de clusters libres
                                                   ; CX = Número de bytes/sector
                                                   ; DX = Número total de clusters
                                                   ;
                mul     cx                         ; AX = Número de bytes/cluster
                mov     cl,10                      ; Ahora lo dividimos entre 1024
                shr     ax,cl                      ; AX = Número de Kbytes/cluster
                mul     bx                         ; DX:AX = Número de Kb's libres
                                                   ;
                cmp     ax,64                      ; Hay más de 64 Kbytes libres?
                jae     crea_fichero               ; *06*
                or      dx,dx                      ; Si no, comprueba el word alto
                jnz     crea_fichero               ; y, si no es cero, salta
                                                   ;
                push    cs                         ; Ahora hay que comprobar si el
                pop     ds                         ; fichero existe ya, porque no
                mov     ax,3D02h          ; necesitamos más espacio libre
                mov     dx,offset ruta_de_fichero
                int     21h                        ; Intentamos abrir el fichero
                jc      error_creacion             ; Salta si el fichero no existe
                mov     bx,ax                      ; Copiamos el "handle" en BX
                mov     ah,3Eh                     ; Función DOS para cerrar el
                int     21h                        ; fichero que acabamos de abrir
                                                   ;
                                                   ; Ahora borraremos dicho fichero
                                                   ;
crea_fichero:   push    cs                         ; Carga el registro de segmento
                pop     ds                         ; de datos con el de código
                mov     ah,3Ch                     ; Función DOS para crear fichero
                xor     cx,cx                      ; Atributos normales del fichero
                mov     dx,offset ruta_de_fichero
                int     21h                        ; Crear/truncar el fichero DS:DX
                jc      error_creacion             ; El fichero existe sólo-lectura
                mov     cs:[hand_de_fichero],ax    ; Guardamos nuestro "handle"
error_creacion: ret                                ; Devuelve flag de Carry ON/OFF

; =-=-=-=-=-=-= CERRAR FICHERO =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

cerrar_fichero: mov     bx,cs:[hand_de_fichero]    ; "Handle" de nuestro fichero
                mov     ah,3Eh                     ; Función del DOS para cerrar
                int     21h                        ; ficheros previamente abiertos
                ret

; =-=-=-=-=-=-= GRABACION DE LA CABECERA =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

graba_formato:  push    cs                         ; Carga el registro de segmento
                pop     ds                         ; de origen con el de código
                mov     bx,cs:[hand_de_fichero]    ; Carga en BX nuestro "handle"
                mov     ah,40h                     ; Función del DOS para grabar
                mov     cx,48                      ; n bytes en un fichero abierto
                mov     dx,offset cabecera_form    ; DX apunta al bloque de datos
                int     21h                        ; Graba 768 bytes desde DS:DX
                ret                                ;

; =-=-=-=-=-=-= GRABACION DE LA CABECERA DEL CUERPO =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

graba_body:     push    cs                         ; Carga el registro de segmento
                pop     ds                         ; de origen con el de código
                mov     bx,cs:[hand_de_fichero]    ; Carga en BX nuestro "handle"
                mov     ah,40h                     ; Función del DOS para grabar
                mov     cx,8                       ; n bytes en un fichero abierto
                mov     dx,offset cabecera_body    ; DX apunta al bloque de datos
                int     21h                        ; Graba 768 bytes desde DS:DX
                ret                                ;

; =-=-=-=-=-=-= GRABACION DE LA PALETA =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=
```

```
graba_paleta:   push    cs                      ; Carga el registro de segmento
                pop     es                      ; extra con el de código; ES=CS
                mov     dx,03C7h        ; Registro PEL Address Address
                xor     al,al                   ; de la tarjeta VGA. Apunta al
                out     dx,al                   ; número de color que queremos
                                                ; leer o escribir.
                mov     dl,0C9h                 ; Registro de datos de paleta
                mov     cx,768                  ; Número de colores * 3 (RGB)
                mov     di,offset buffer_de_disco
bucle_paleta:   in      al,dx                   ; Leemos un byte de paleta, y
                shl     al,1                    ; lo rotamos dos bits a la izq.
                shl     al,1                    ; para comformar un valor de
                stosb                           ; ocho bits, standard en ILBM,
                loop    bucle_paleta            ; y después lo almacenamos
                                                ;
                push    cs                      ; Carga el registro de segmento
                pop     ds                      ; de origen con el de código
                mov     bx,cs:[hand_de_fichero] ; Carga en BX nuestro "handle"
                mov     ah,40h                  ; Función del DOS para grabar
                mov     cx,768                  ; n bytes en un fichero abierto
                mov     dx,offset buffer_de_disco
                int     21h                     ; Graba 768 bytes desde DS:DX
                ret                             ; *08*

; =-=-=-=-=-=-= GRABACION DEL CUERPO =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

graba_cuerpo:   mov     ax,0A000h               ; *09*
                mov     ds,ax                   ; DS = A000h (Segmento de VGA)
                mov     bx,cs:[hand_de_fichero] ; "Handle" de nuestro fichero
                xor     dx,dx                   ; Origen: primer byte de VRAM
                mov     cx,64000        ; Número de bytes para grabar
                mov     ah,40h                  ; Función DOS para grabar un
                int     21h                     ; bloque de datos a un fichero
                ret

; =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

; =-=-=-=-=-=-= PARTE NO RESIDENTE =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

instalacion:    call    busca_opciones          ; Recogida de línea de comando
                call    instalar_TSR            ; Modifica las interrupciones
                call    copyright               ; Imprime mensaje en la panta-
lla
                mov     ax,3100h        ; AH = Función 31h del MSDOS:
                                                ; Terminar el programa en curso
                                                ; y dejar residente una parte.
                                                ; AL = ERRORLEVEL para el MSDOS
                mov     dx,005Ah        ; *12*
                                                ; Número de bloques de 16 bytes
                                                ; que queremos dejar residentes
                                                ; a partir de CS:0000h. (1.4Kb)
                int     21h                     ; Ejecuta la interrupción MSDOS
                                                ; que termina el proceso actual
                                                ; Fin del programa.

; =-=-=-=-=-=-= INSTALACION =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

instalar_TSR:   cli                             ; Deshabilita las interrupcio-
nes
                in      al,21h
                xchg    al,ah
                mov     al,00010001b
                out     20h,al                  ; ICW1
                call    micro_retardo
                mov     al,int_base
                out     21h,al                  ; ICW2
                call    micro_retardo
                mov     al,00000100b
                out     21h,al                  ; ICW3
                call    micro_retardo
                mov     al,00000001b
                out     21h,al                  ; ICW4
                call    micro_retardo
                mov     al,11111111b
                out     21h,al                  ; RDY
                call    micro_retardo
                xchg    al,ah
                out     21h,al                  ; OCW1

                push    cs                      ; Carga el registro DS con el
                pop     ds                      ; segmento_codigo
                xor     ax,ax                   ; Carga el registro ES con
cero,
                mov     es,ax                   ; donde está la tabla de int.
                mov     si,offset tabla_vector  ; Dirección de origen en DS
                mov     di,int_base*4           ; Dirección de destino en ES
                mov     cx,8                    ; 8 interrupciones
                push    cs                      ; Carga el segmento de código
                pop     ax                      ; AX para copiarlo a la tabla
bucle_interrup: movsw                           ; Copia el offset en destino
                stosw                           ; Copia el segmento en destino
                loop    bucle_interrup          ; Repite el proceso ocho veces
                sti                             ; Habilita las interrupciones

                mov     ah,34h                  ; *05*
                int     21h                     ; Función del MSDOS que devuel-
ve
                mov     cs:[estado_DOS_off],bx  ; la dirección del flag de
esta-
                mov     cs:[estado_DOS_seg],es  ; do de la parte crítica del
DOS
                ret                             ; en los registros ES:BX.

micro_retardo:  call    micro_retardo2
micro_retardo2: ret

tabla_vector    dw      offset nuestra_IRQ0
                dw      offset nuestra_IRQ1
                dw      offset nuestra_IRQ2
                dw      offset nuestra_IRQ3
                dw      offset nuestra_IRQ4
                dw      offset nuestra_IRQ5
                dw      offset nuestra_IRQ6
                dw      offset nuestra_IRQ7
```

```
; =-=-=-=-=-=-= CONFIGURACION =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

busca_opciones:   push   cs                     ; Estas dos instrucciones hacen
                  pop    ds                     ; DS=CS.
                  mov    si,0080h          ; Zona de parámetros del MSDOS

                  lodsb                         ; Lee el contador de bytes
                  or     al,al                  ; ¿Es igual a cero?
                  jz     sin_parametros         ; Si lo es, no continuar

                  mov    cl,al                  ; Utilizamos CX como contador
                  xor    ch,ch                  ; Carga AH con cero

bucle_opciones:   lodsb                         ; Lee el siguiente byte en AL
                  cmp    al,13                  ; ¿Es el código de fin de
texto?
                  je     sin_parametros         ; Fin del procedimiento
                  cmp    al,'/'                 ; Si no es la barra, puede que
                  jne    bucle_opciones         ; sea un espacio o un tabulador
                  lodsb                         ; Lee el byte detrás de la
barra
                  cmp    al,'l'                 ; Opcion "LBM", minúscula
                  je     opcion_primera
                  cmp    al,'L'                 ; Opcion "LBM", mayúscula
                  je     opcion_primera
                  cmp    al,'r'                 ; Opcion "RAW", minúscula
                  je     opcion_segunda
                  cmp    al,'R'                 ; Opcion "RAW", mayúscula
                  je     opcion_segunda
                                                ; Opción incorrecta, volver
sin_parametros:   ret                           ; Se toma la opción por defecto

opcion_primera:   mov    cs:[tipo_de_fichero],1     ; Tipo LBM
                  ret

opcion_segunda:   mov    cs:[tipo_de_fichero],2     ; Tipo RAW
                  mov    cs:[ruta_de_fichero+09],'R'
                  mov    cs:[ruta_de_fichero+10],'A'
                  mov    cs:[ruta_de_fichero+11],'W'
                  ret

; =-=-=-=-=-=-= COPYRIGHT =-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=-=

copyright:        push   cs                     ; Carga DS con el mismo valor
                  pop    ds                     ; que CS (segmento_codigo)
                  mov    dx,offset cadena_texto ; Apunta DX a la cadena de
texto
                  mov    ah,9                   ; Función del MSDOS para impri-
                  int    21h                    ; mir cadenas de texto ASCII
                  ret                           ; en la pantalla en modo texto

cadena_texto      db     'HUNTER Versión 1.00.  Pulsa la tecla F12 para '
                  db     'activar el programa.',10,13
                  db     'Copyright (c) 1994 David Brioso Santos.',10,13,36

segmento_codigo   ENDS

                  END    inicio_codigo

; ****************************************************************************
;
; Notas:
;
; (*)    Los números de nota se indican entre asteriscos para que puedan distin-
;        guirse de valores enteros (por ejemplo: *9* ).
;
; (**)   Las palabras clave o directivas del ensamblador (TASM, MASM, etc, no el
;        lenguage) están en mayúsculas.
;
; (1)    La interrupción base no se refiere a la numeración normal (0-255), sino
;        a las IRQs (Interrupt Request), las interrupciones hardware generadas a
;        través del Peripheral Interrupt Controller (PIC). Dichas interrupciones
;        son solicitadas por dispositivos periféricos, el teclado, la tarjeta de
;        control de la unidad de diskette, disco duro, el temporizador, etc.  El
;        PIC redirige estas solicitudes hacia la correspondiente IRQ (de 0 a 7),
;        que pueden estar situadas en cualquier vector de interrupción software,
;        pero siempre que sean múltiplos de ocho. Así, por ejemplo, las IRQs 0-7
;        podrían estar ubicadas en las interrupciones 00h-07h, 08h-0Fh, 10h-17h,
;        18h-1Fh, 40-47h, 68h-6Fh, D8h-DFh, o F0h-F7h.  Los vectores por defecto
;        son 08h-0Fh,  correspondiendo la IRQ0 al vector 08h,  la IRQ1 al vector
;        09h, y la IRQ7 al número 0Fh.  La mayoría de los juegos comerciales que
;        funcionan por teclado  instalan su propia interrupción  sustituyendo el
;        vector original por otro,  bien mediante la función dispuesta para ello
;        por el MSDOS, o bien modificando directamente el vector.  Cualquiera de
;        los dos  procedimientos desactivará la rutina de teclado que hubieramos
;        instalado previamente. Por el contrario, si programamos el PIC para que
;        dirija las interrupciones hardware directamente a nuestros vectores sin
;        posibles intromisiones de otro programa,  podemos detectar la pulsación
;        de la tecla deseada y, a continuación, saltar al vector de interrupción
;        que se supone que es el correcto (vector 0Ah,  en el caso del teclado).
;        Este procedimiento es el empleado por varios programas para capturar la
;        interrupción de teclado, detectar una determinada combinación de teclas
;        que dispara un proceso y,  al final,  saltar a la rutina de teclado del
;        programa principal para que continúe la ejecución normalmente.   El más
;        conocido de estos programas es Screen Thief (Nildram Software), similar
;        a HUNTER en su objetivo, pero radicalmente distinto en su concepción.
;        Para evitar conflictos con otros programas o hardware, se puede cambiar
;        el EQU "int_base" a 88h, 90h, 98h, y así sucesivamente, hasta encontrar
;        una zona de ocho interrupciones contiguas que se encuentren libres.
;
; (2)    Todos los ejecutables .COM comienzan su ejecución en la dirección 0100h
;        del segmento de código (CS:0100h).  Los ejecutables de tipo .EXE pueden
;        comenzar en cualquier dirección del segmento de código principal.
;
; (3)    El orden correcto de ubicación debe de ser:  datos y procedimientos que
;        van a permanecer residentes en memoria y, a continuación, los que van a
;        desaparecer tras su ejecución  (sería poco ahorrativo que permaneciesen
;        residentes también  los procedimientos de configuración  e instalación,
;        que no son necesarios una vez que han cumplido su objetivo).
;
; (4)    Realmente el buffer  podría tener una longitud mayor  para facilitar la
;        compresión de la pantalla,  pero no resultaría económico en terminos de
;        memoria. Hemos creido conveniente alcanzar un punto de equilibrio entre
;        la comodidad y el ahorro de espacio. Un factor primordial del diseño de
;        utilidades TSR  (Terminate and Stay Resident)  es que deben dejar libre
;        la mayor cantidad de memoria posible. Si es necesario, se puede habili-
```

```
;       tar una zona de trabajo en memoria expandida (EMS) o extendida (XMS).
;       El buffer de disco puede estar situado a partir de la dirección CS:0000
;       para ahorrar 256 bytes.  Estos primeros 256 bytes, el PSP (Program Seg-
;       ment Prefix, la cabecera del programa)  quedan inutilizados al dejar el
;       programa residente.  Para reubicar el buffer, basta con poner un EQU al
;       comienzo del listado y sustituir la linea de declaración del buffer por
;       " db 509 dup (0)". La linea del EQU sería: "buffer_de_disco EQU 0000h".
;
; (5)   La función 34h del MSDOS devuelve  la dirección del  flag de la sección
;       crítica del Sistema Operativo  en ES:BX.  Expresado  de una forma menos
;       críptica quiere decir que  nos devuelverá en estos registros un flag de
;       estado que nos indica en que estado  se encuentra el Sistema.  Si dicha
;       palabra (Critical Section Flag),  es igual a 0000h,  podremos acceder a
;       las funciones del propio Sistema sin riesgos. Si el flag contiene algún
;       otro valor, quiere decir que podemos tener problemas al acceder a estas
;       funciones.  Cuando el MSDOS se encuentra realizando alguna función crí-
;       tica con ficheros, puede estar en un estado imprevisible.
;       El uso de este flag de estado es bastante  aconsejable en cualquier TSR
;       o gestor de interrupción  que necesite acceder a una función del MSDOS.
;       Previamente se debe comprobar el estado del Sistema mirando el conteni-
;       do de esta palabra.
;       Un último detalle: no debe usarse esta función desde el gestor de inte-
;       rrupciones o programa residente,  sino que debe ser llamada en la parte
;       de instalación, y después únicamente comprobar el flag.
;
; (6)   En este caso nos limitaremos a dar por sentado que esa va a ser la lon-
;       gitud de nuestro fichero (64Kb).  No es una práctica recomendable hacer
;       esto en todos los casos,  ya que por regla general  no vamos a tener la
;       certeza de que esa es la longitud máxima. Si hubieramos utilizado una u
;       otra forma de compresión, tendriamos una longitud variable. Por lo tan-
;       to, o bien se realiza la comprobación una vez que conocemos la longitud
;       del fichero en sí, o bien utilizamos una cifra aproximativa del máximo.
;
; (7)   No utilizamos el byte de estado de las teclas modificadoras (ALT, CTRL,
;       SHIFT, etc),  situado en 0040:0017,  porque es bastante probable que la
;       rutina de teclado del MSDOS haya sido sustituida por una específica del
;       juego o programa. De hecho, ésta es la base de nuestra utilidad.
;       A propósito, un detalle bastante curioso:  en algunos manuales de BIOS,
;       la zona de variables aparece localizada en la dirección 0040:0000h y en
;       otros aparece en 0000:0040h. Ambas direcciones corresponden a una misma
;       dirección real. Son dos formas de direccionar lo mismo.
;
; (8)   Este procedimiento, como los demás, puede ser depurado para ahorrar me-
;       moria y ganar algo de velocidad de proceso. Se ha intentado mantener un
;       equilibrio entre estos objetivos y la claridad del programa, aunque sea
;       bastante difícil llegar a un término medio. No se puede llegar a satis-
;       facer ambas metas al mismo tiempo, por lo que hemos elegido la utilidad
;       didáctica del programa al desarrollarlo.  De cualquier manera, las téc-
;       nicas de depuración de programas requieren una serie de artículos dedi-
;       cados en exclusiva para poder ahondar mínimamente en el tema.
;
; (9)   Valgan de pequeña muestra para ilustrar  la nota anterior estas líneas.
;       Para cargar un registro de segmento de datos  (DS, ES, SS, y en los 386
;       y superiores, FS y GS) podemos emplear diversos sistemas:
;
;               push cs             ; esta instrucción ocupa sólo 1 byte
;               pop  es
;
;       si queremos copiar el valor de uno en otro directamente;
;
;               mov  ax,SEG segmento_de_codigo
;               mov  ds,ax
;
;       si queremos cargar un segmento reubicable en uno de los registro;
;
;               push 8000h          ; ahorramos un byte, en lugar de: mov ax,8000h
;               pop  ds             ;                                 mov ds,ax
;
;       si deseamos cargar un registro con un valor fijo, invariable.
;       Esta opción tiene el inconveniente de que ensambladores como TASM
;       la interpretan como macro, y la sustituyen por una serie de ins-
;       trucciones que no tienen nada en común con la original;
;
;               mov  es,ds:[puntero_de_segmento]
;
;       si previamente hemos almacenado el valor del segmento en esta variable,
;       podemos cargarlo directamente. Este procedimiento es bastante normal en
;       casos especialmente complicados. Si cada vez que necesitamos acceder al
;       valor de un segmento tenemos que hacer cálculos, o llamar a una función
;       que nos lo devuelva,  es siempre más rápido hacerlo una vez y almacenar
;       dicho valor en una variable.
;
; (10)  Al ser el ILBM un formato nativo del Commodore Amiga (donde es conocido
;       como "Interchange File Format", IFF para acortar), todas las palabras y
;       dobles palabras ("word" y "long" en el Amiga, "word" y "dword" en el PC
;       o, lo que es lo mismo, en los procesadores Motorola e Intel) se encuen-
;       tran almacenadas en el orden inverso al que estamos acostumbrados.  Por
;       lo tanto,  el byte alto de una palabra se escribe antes que el bajo; en
;       las dobles palabras sucede lo mismo  para los cuatro bytes:  primero se
;       almacena el más significativo (bits 31-24),  inmediatamente después los
;       bits 23-16, a continuación los bits 15-8,  y por último,  el byte menos
;       significativo (bits 7-0).  Por esta razón no podemos utilizar las pseu-
;       do-instrucciones de declaración de datos (dw, dd) de palabra y de doble
;       palabra.   Este método para almacenar datos multi-byte es conocido como
;       "big endian", en contraposición al sistema Intel, conocido como "little
;       endian".
;
; (11)  Para cambiar la unidad/ruta/nombre del fichero de destino hay que tener
;       bastante cuidado. En el procedimiento Busca_opciones se altera directa-
;       mente este nombre si se elige la opción de grabar un fichero tipo "RAW"
;       en lugar de "LBM".  Por eso hay que tomar la precaución de modificar en
;       consonancia este procedimiento, de forma que se eviten problemas con la
;       grabación del fichero en disco.
;
; (12)  Asímismo conviene tener cuidado al modificar el programa para usos par-
;       ticulares,  que es uno de los objetivos que persigue esta modesta apli-
;       cación.   En el registro DX indicamos al Sistema cuántos párrafos de 16
;       bytes queremos dejar residentes. Cualquier alteración de la longitud de
;       este programa debe tener este factor bastante en cuenta. Si se diera el
;       caso de que el MSDOS liberase parte de la memoria que supuestamente iba
;       a permanecer residente, los resultados serían imprevisibles.
;
; ***************************************************************************
```

DOOM

Un curso muy especial

Por:

Joseph Priscok

Como introducción a este artículo, les daré un aviso: Si buscas ejemplares de código muy compacto y elegante, trucos y técnicas para acceso a la RAM de vídeo, o secretos del modo protegido de la CPU, deja ya de leer. Pero si por el contrario, tu sueño se rompe con pesadillas, tu pareja se asusta cuando te acuestas (muy tarde) depués de una larga sesión con la sierra mecánica, o sencillamente quieres poner tus propios toques especiales en uno de los mejores juegos de todos los tiempos, ¡sigue adelante!

Espero que este curso te sirva como una pequeña introducción a la maquinaria del mejor juego de acción en el pasado reciente - DooM!, de la compañía ID software.

Los dos elementos principales del juego son los ficheros DOOM.EXE y DOOM.WAD. De los dos, DOOM.WAD es el fichero de recursos, conteniendo las imágenes gráficas, tejidos, y datos de mapas que vamos a recorrer inicialmente. ("WAD" es la extensión asignada a estos ficheros de recursos por iD Software, los responsables por el juego.) DOOM.EXE, al contrario, a parte de ser la máquina real del juego, define el comportamiento de elementos del juego como monstruos y armas. La manera en que DOOM.EXE puede ser cambiado por nosotros será motivo de artículos venideros dentro de esta misma sección. Por ahora, exploraremos los elementos que componen un nivel de Doom, con la ayuda de DCK (Doom Construction Kit), que podéis encontrar dentro del DOOMKIT que se anuncia en esta misma revista.

Como sus antecesores, Wolfenstein 3D, Blake Stone, y Spear of Destiny, el universo de Doom consiste en salas y pasillos, detalles decorativos, monstruos, y armas (véase la Figura 1). Aumentando estos elementos básicos, a Doom se le ha añadido un aspecto vertical con escaleras, asensores, pozos, y más cosas. Una gran variedad de tejidos, interruptores inteligentes y puertas, efectos variables de iluminación, y plataformas de teletransporte proveen a DOOM con una riqueza que han sido muy poco vista en juegos de este género.

La versión registrada de Doom, que es la única versión que puede ser legalmente modificada, está compuesta de 27 niveles (o misiones) individuales, repartidos entre tres episódios distintos. Son identificadas, por ejemplo, E2M3 (episodio 2, misión 3), E1M4, E3M6, etc. Cada misión está compuesta de varios puzzles, trampas, y laberintos; la calidad de ellos depende del talento del diseñador.

Comenzaremos nuestro discurso sobre la construcción de niveles através de la creación de un nivel sencillo con DCK. (Si aún no has instalado DCK, es mejor que lo hagas ahora, descomprimiendo DCK10.ZIP en un subdirectorio temporal y ejecutando RUNME.EXE. A continuación, tienes que instalar BSP.EXE (incluido), ya que DCK lo utiliza al compilar el nivel del Doom, una vez fuera del redactor. Has de modificar el fichero DCK.CFG para que refleje la configuración de tu sistema, y listo para empezar!) Una vez arrancado, DCK presentará una pantalla similar a la de la Figura 2.

DCK (como la mayoría de redactores de ficheros WAD) tiene varios modos distintos de operación, según los objetos con los que quiere interactuar el diseñador. (V)ertices,

Figura 1

Figura 2

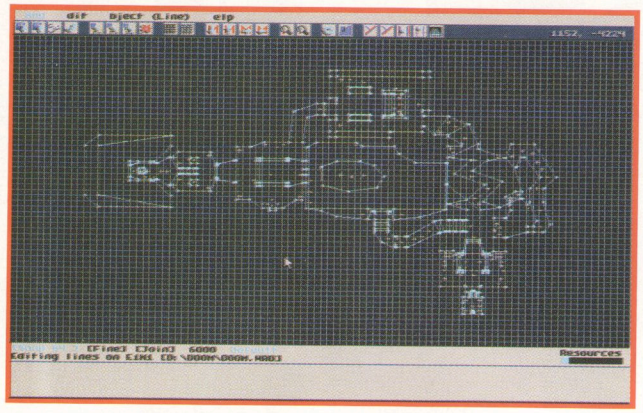


Figura 3

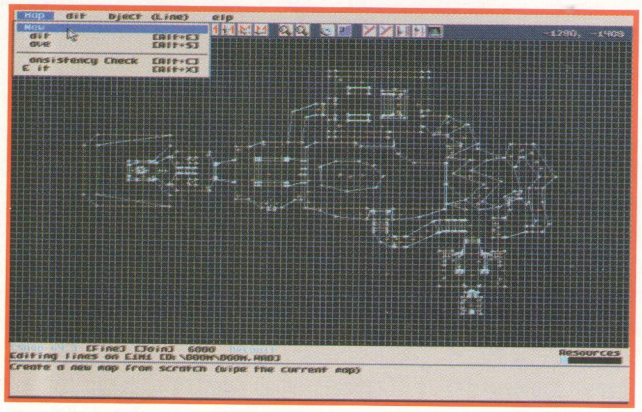
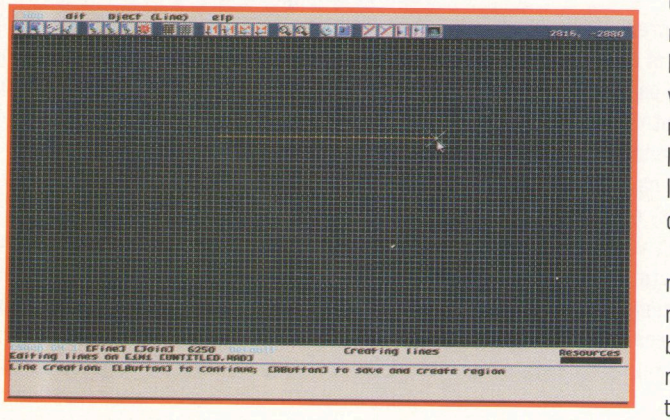


Figura 4



(L)ineas, (S)ectores, y (T)hings (cosas) son los objetos básicos de un nivel de Doom, y dentro de DCK, el modo de redacción puede ser cambiado tecleando la primera letra del tipo de objeto. Los acciones del ratón serán indicadas en este artículo por la clave: CI = 'clik' izquierdo, DCI = doble 'clik' izquierdo, CD = 'clik' derecho, y DCD = doble 'clik' derecho.

Debidi a que los aspectos generales de DCK están explicados en la documentación que acompaña al programa, no entraremos en un tratamiento general del redactor. Dado que las acciones y elementos básicos manejados durante este artículo son comunes a cualquier redactor de ficheros WAD (y hay varios disponibles). Esta informacion puede ser utilizada en conjunto con tu favorito (DEU, DOOMCAD, etc.) Sólo tendrás que encontrar y traducir las acciones equivalentes del redactor de tu elección.

Activando la opción (M)ap (mapa) en el menu superior (Figura 3), escoges (N)ew (nueva) para empezar el trabajo de crear un nivel nuevo. Tecleas 'L' para entrar en el modo de manipular líneas, y eliges qualquier punto en la pantalla. DCI, y llevas el ratón a la derecha hacia otro punto del 'grid' o rejilla (Figura 4). CI (para terminar este segmento de la línea), muevete hacia abajo, CI, muevete hacia la izquierda, CI, y hacia arriba hasta llegas al punto de empezar. CD, y la pantalla parecera Figura 5. Mientras que muevas el ratón através de las líneas que hayas creado, una variedad de información se te mostrará en la parte inferior de la pantalla. Mientras que una línea es illuminada, sus propias características son mostradas.

Bien, tecleando 'S' en este momento pondrás DCK en el modo de sectores, lo cual es un buen punto de comienzo para nuestra explicación de la estructura de los elementos de los ficheros WAD y los niveles de Doom. Moviendo el ratón por cualquiera de las líneas que hayas creado anteriormente, verás que el rectángulo entero se ilumina (figura 6). Este rectángulo consta de un sector, o una pista cerrada de líneas. Cada sector es definido en este modo, en términos de sus líneas componentes. El status en la parte inferior de la pantalla muestra, en este momento, datos relacionados con este sector, específicamente el número del sector, alturas de suelo y techo, nivel de iluminación, tejidos de suelo y techo, y el 'tag' o etiqueta del sector. Todos estos elementos pueden ser modificados, pero por el momento, trabajeremos con sus valores por defecto.

Tecleas de nuevo 'L', para volver al modo de líneas. Información distinta, específica de las líneas se puede ver ahora en pantalla. Tanto como los sectores son definidos por sus líneas componentes, estas mismas son definidas por sus vértices de origen y terminación. Tecleando 'V' nos pone en el modo de redactar los vertices, y la pantalla ahora refleja datos específicos de los vertices iluminados por la proximidad al ratón (Figura 7). Estos datos son limitados a la identificación numérica del objecto, y sus coordenadas X y Y.

Volvemos ahora al modo de líneas, y examinamos los elementos componentes de una de ellas, como aparece en la ventana de status en la parte inferior

de la pantalla.

Cada línea tiene un 'linedef', o definicion de línea. Esta se compone de los dos vertices del objeto, siendo el de origen y el de terminación. Como se puede ver, los números de identificación de estos vértices son mostrados, junto con una flecha que indica la dirección de la línea. Su longitud, 'tag' de sector asociado, y tipo de línea le siguen.

Refiriéndome a la pantalla, te habrás dado cuenta de que una línea corta, extendiéndose de un lado de cada línea. Esta, identifica el lado principal (o sidedef 1) de la misma. Cada lado de la línea tiene un sidedef asociado, o definición de lado. En nuestro ejemplo sencillo, todas tienen sólo un lado, dado que nuestro sector no tiene exterior - no hay manera para el jugador de salir del sector para ver el lado trasero de estas líneas-. Como las líneas no se adjuntan a otros sectores, son definidas como <one-sided> o de un sólo lado. Cada <sidedef> consta de definiciones para los tejidos superiores, inferiores y enteros (<upper>, <lower>, y <full>).

Hechamos un vistazo rápido a nuestro sector. Tecleas "T" para entrar el modo de redacción de <things> (cosas). DCI para abrir un dialogo de inserción de cosas (Figura 8). Tecleas los siguente: F2, 'Home', 'Enter', y CI en 'OK', insertando un punto de comienzo para un jugador (<single-player game>). Tecleas ALT-C, y CI 'Go' para forzar una prueba de consistencia del nivel. Dado que todo ha ido bien, un diálogo de errores aparecerá. No hacemos caso de los dos errores que aparecen. (Se tratan de juegos multi-jugador y lucha a muerte (<Deathmatch>). El error <Missing an exit line> será resuelto en seguida.) ESC saldrá de este diálogo, y dando al F9 invocará el menú de guardar ficheros. Asignas un nombre o quedate con el de por defecto (UNTITLED.WAD). Después de CI en 'OK', DCK saldrá, BSP construirá el mapa de nodos del nivel actual, y Doom será arrancado.

DCK se encargá de lanzar Doom con los parámetros correctos del <command-line>, cargar tu nuevo fichero WAD y saltar hacia él directamente. Para cargar un WAD externo fuera del DCK, lanzas Doom co los parametros "doom -file wadfile.wad", y el juego se arranca en el modo normal, pero con este nivel externo reemplazando el original.

La vista desde dentro de tu primer sector debe paracer similar a la figura 9 - un salón sencillo, rectangular con los tejidos por defecto asignados a las paredes, techo, y suelo. Puedes pasear y disparar tu arma, pero nada más, dado que no hay ningún otro objecto con los que interactuar. Te habrás dado cuenta que el <engine> de Doom se encarga de los efectos de sónido, música, generación de los <sprites>, y todos los aspectos de presentación del juego. El diseñador del nivel sólo tiene que preocuparse con los aspectos artísticos y estratégicos de su creación.

Sales de Doom (ESC, Q, Y) y volverás a DCK. Ahora, vamos examinar los componentes de una línea en mayor profundidad. La figura 10 muestra el diálogo de redacción de los <linedef> para una línea seleccionada. (Los objectos en DCK pueden ser seleccionados con CI en cada de ellos, y despues CD para redac-

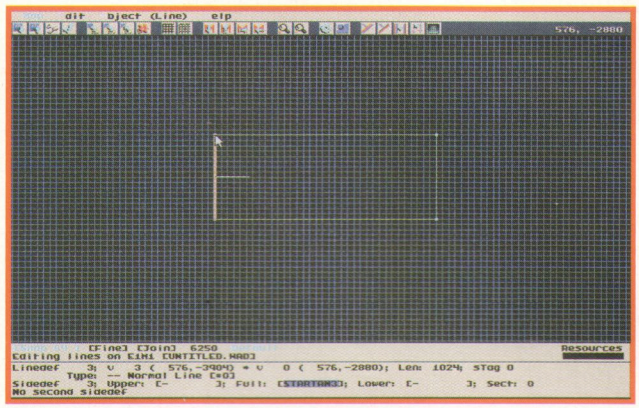


Figura 5

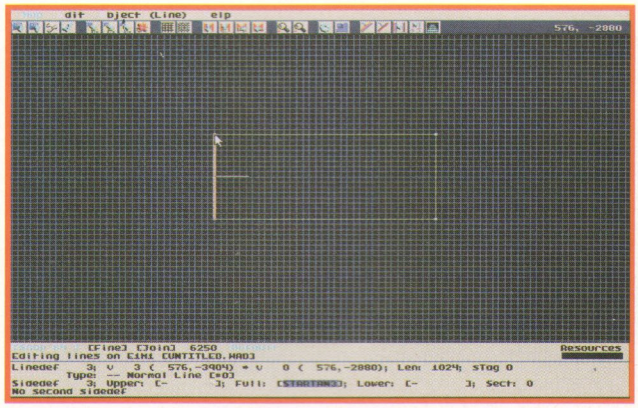


Figura 6

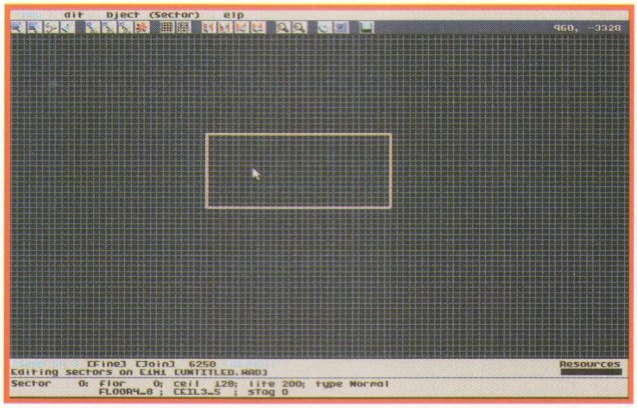


Figura 7

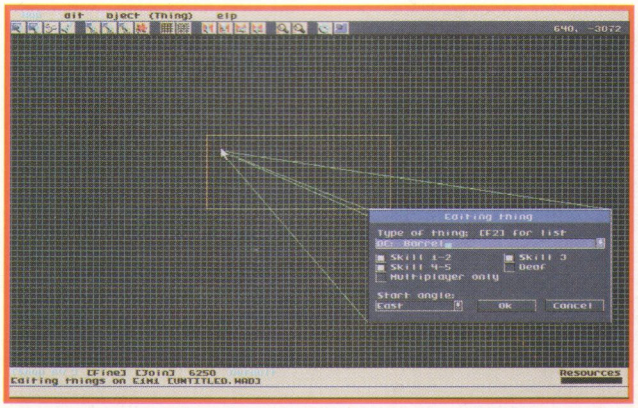


Figura 8

Figura 9

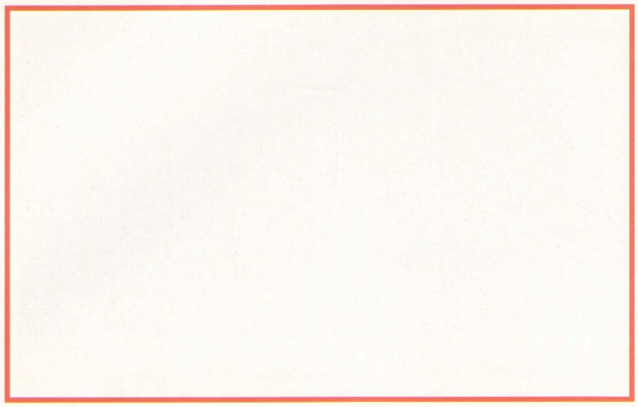


Figura 10

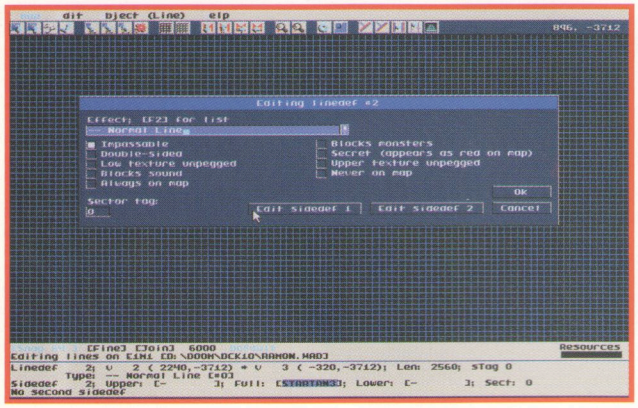


Figura 11

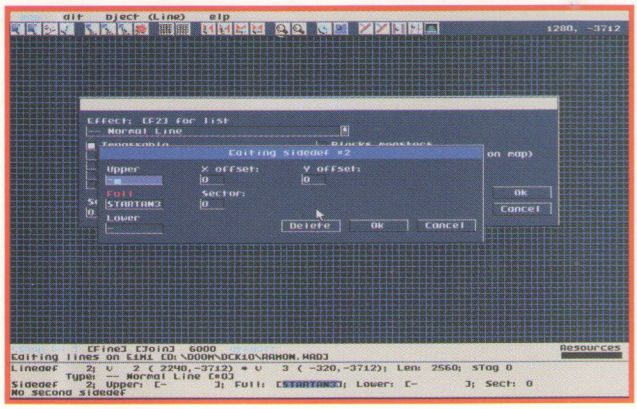
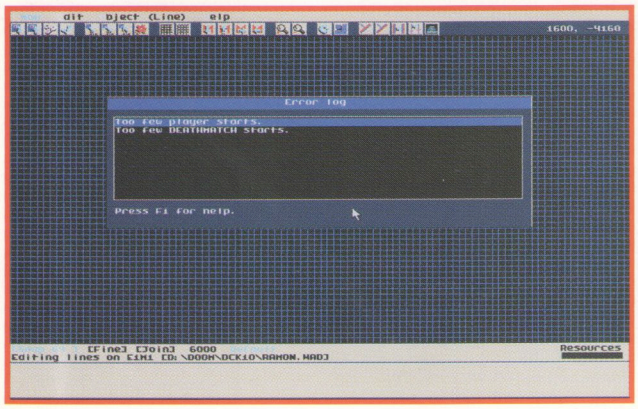


Figura 12



tar las características del grupo entero.)

Cada línea en un nivel de Doom posee un <linetype>, dentro de la gama disponible. El tipo por defecto es <Normal>, como puede ser visto en el campo iluminado del diálogo. Tecleando F2, puede repasar una lista extensa de opciones. Pulsando <Enter> en cualquiera de ellos selecionará aquel <linetype> para el objecto elegido. Paseas por la lista hasta llegar al <Switch EXIT> y cuando éste es iluminado dale al <Enter>.

Una vez fuera del dialogo por completo, el segmento de línea en pantalla habrá cambiado de color para reflejar su status nuevo. Si recuerdás, cuando hicimos la prueba de consistencia antes de lanzar Doom, uno de los erores encontrados fue <Missing an exit line>. La acción que hemos tomado, asignando <linetype> <Switch EXIT> a nuestra línea elegida, resuelve aquel problema. Cuando el nivel es recompilado y Doom es lanzado de nuevo, podrás acercarte a aquella línea, apretarla, y notarás que se ha terminado el nivel.

Otras características pueden ser modificadas desde este dialogo también. DCK es un redactor inteligente, y durante el expansión de nuestro nivel, notarás que, en general, toma buenas decisiones sobre las características de una línea generada. Los <flags> que pueden ser conmutados se relacionan al aspecto de una línea. Quizas sea mejor decir que se relacionan a la presentación de una línea, dado que su aspecto es determinado por los <sidedef>. Los <flags> son definidos así:

<Impassable>- por defecto, una línea de un lado. No se puede atravesar.

<Double-sided>- una línea que adjunta dos sectores (doble cara).

<Low texture unpegged> - comportamiento del tejido inferior

<Blocks sound> - evita que el sonido llege a los monstruos.

<Always on map>- siempre aparece en el mapa de jugador.

<Blocks monsters>- línea no pasable por monstruos.

<Secret>- Oculto a los ojos del jugador.

<Upper texture unpegged> - comportamiento del tejido superior.

<Never on map>- nunca aparece en el mapa.

Se puede ver que estos <flags> se dividen en dos categorias generales: los que son relacionados con el mapa, y los que modifiquen el comportamiento de la línea. Seleccionando cualquier opción conmutarás entre sus estados de ser habilitado y deshabilitado. Hay otros campos también disponibles para ser modificados:

El <Sector tag> tiene que ver con las acciones que pueden ser disparadas como resultado de interactuar con esta línea. Por ejemplo, podemos asignar el <linetype> como <Walk-R Lights turn on> (Andando, repetible, luces incendidas). Algún sector, sea donde sea en el mapa, tiene un <sector tag> arbitráriamente asignado a él, que será repartido con dicho <linedef>. Cuando el jugador atraviesa la línea, las luces del sector asociado serán iluminadas. Esta es un relación importante, dado que es la manera en que muchos de los efectos y detalles de Doom son

realizados.

<Edit Sidedef 1> y <Edit Sidedef 2> son dos campos de gran interés. Es gracias a ellos que podemos traer el talento artístico a nuestros esfuerzos, pintando un nivel según nuestros gustos. Los <sidedef> definen la aparencia real de una línea, una vez proyectada por el <engine> de Doom. En nuestro ejemplo, cambiábamos el <linetype> de una de nuestras líneas a <Switch EXIT>. Vamos hacer que el aspecto de la línea corresponda a su función (Figura 11). Con el 'TAB' o el raton, iluminas el campo <Full> y teclea as al F2. Todos los tejidos disponibles para este objecto se muestran junto con sus dimensiones (pasa através de ellos con las flechas hacia arriba a abajo). Si sabes el nombre de un tejido en particular, puedes arracar un diálogo de búsqueda tecleando ALT-F, y el nombre que necesites. La búsqueda puede ser repetida hacia adelante y atrás con las teclas / y \. Eliges qualquier tejido que te guste en ese momento, y con la tecla <Enter>, lo asignas al tejido <Full> de este <sidedef>.

De camino, tomas nota del campo <Sector>. A veces, puede ser necesario redactar manualmente este campo, especialmente en el momento de juntar sectores o bien al borrar sectores adjuntos. Los campos X e Y <Offset> son utilizados en el ajuste de tejidos por motivos "cosméticos".

Sal del dialogo <sidedef> seleccionando "OK", y una vez más para salir del diálogo <linedef>. Haz un chequéo de consistencia con ALT-C. Los resultados ahora tendrán que aparecer como en la Figura 12. Lanzas el Doom (F9), y observa los cambios que hayas efectuado. Recuerda acercarte a la pared modificada, tocarla, y salir del nivel.

Una ves de vuelta en DCK, abre de nuevo el diálogo <sidedef>, y tomas nota de los campos que definen los tejidos <Upper> y <Lower> (superior e inferior). En nuestro ejemplo, sólo la definición <Full> era requerida (iluminada en rojo). Cuando líneas son adjuntas a sectores múltiples, existen casos donde aquellos sectores pueden tenir alturas distintas de suelo o de techo. Donde una línea separa dos sectores semejantes, las definiciones <Upper> y <Lower> determinan el aspecto de las áreas expuestas de los sectores adjuntos que caen arriba o abajo de los elevaciones normales de las superficies del sector normal.

Como conclusión, vamos ejecutar rápidamente las siguentes operaciones, para darte algunas ideas con las que jugar mientras que llega el próximo ejemplar de la revista.

Asegúrate que estás en el modo de (L)íneas. Escoges una línea vertical (CI), arrastra el menú (O)bject hacia abajo, y eliges <Split linedefs at center> (divide <linedef> por el centro). Tu nivel tendrá que ser similar a la figura 13. Entra al modo de (V)értices, y arrastra el vértice nuevo hasta que esté alineada con una línea del <grid>. Ve al modo (L)ínea, y DCI en el vértice superior. Muévete hacia la derecha, CI, hacia abajo, CI, y hacia la izquierda hasta el vértice del centro.

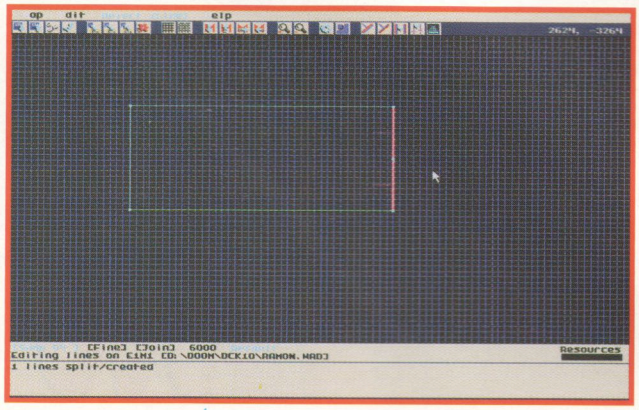


Figura 13

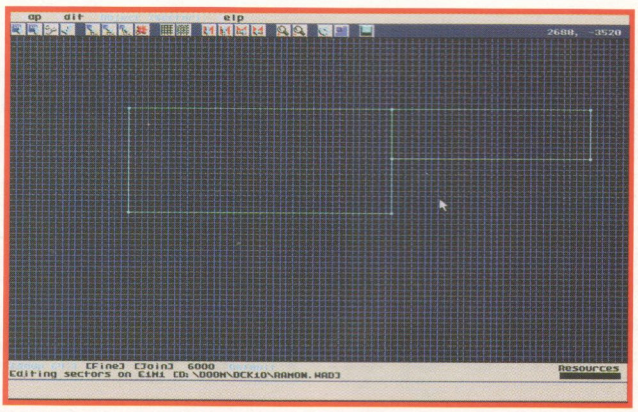


Figura 14

CD. Su pantalla deberá parecerse a la Figura 14. Entrando al modo (S)ector, verás que hay dos sectores distintos y separados, juntados por la línea adjunta a los dos. Ejecuta un ALT-C para comprobar la consistencia, y lanza Doom atraves del F9.

Al volver a DCK, ten cuidado de examinar los <line-> y <sidedef> de la línea que está repartida entre los dos sectores. Verás que ya es de doble cara (<two-sided>), y que todos los tejidos han sido convertidos a <->, o trasparente, automáticamente por DCK.

En el ejemplar siguente, de Código Juego empezaremos a añadir sectores adicionales, creando escaleras y puertas, plataformas de teleportación, y un poco de peligro al entorno.

Averiguarás lo fácil que resulta construir tus propios niveles para tu deléite y el de los demás Doomaniacos!!

Hasta entonces - ¡Feliz DooM!

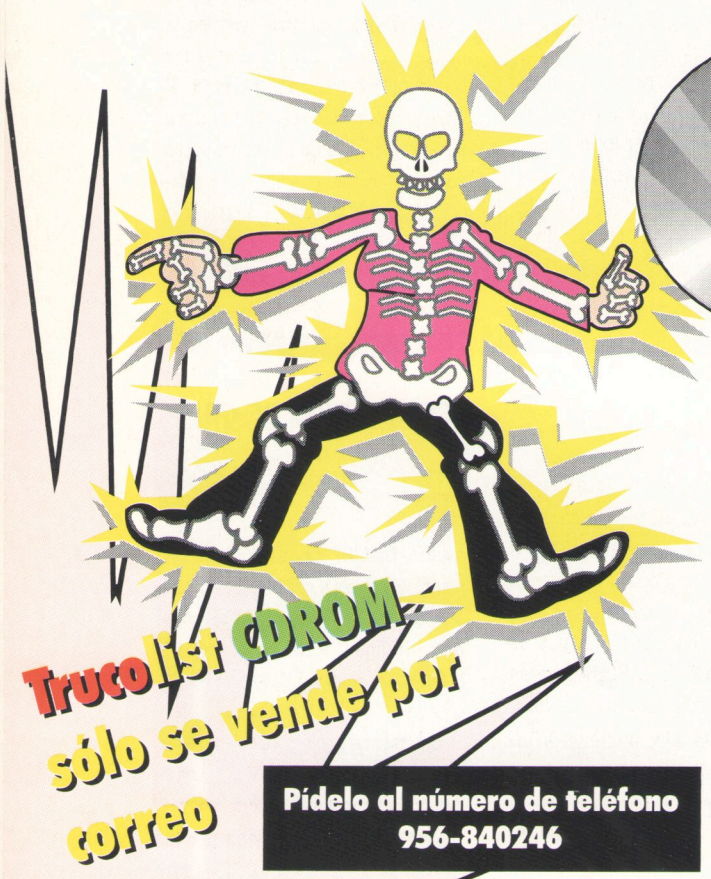
GRACIAS AL EDITOR DCK (DOOM CONSTRUCTION KIT) PODEMOS REALIZAR LA COLOCACIÓN DE OBJETOS, PAREDES Y PERSONAJES A NUESTRO ANTOJO, CON UN FÁCIL INTERFACE QUE SE MANEJA A GOLPE DE RATÓN.

3000 amantes de la buena programación ya tienen

PROGRAMACIÓN PRÁCTICA DE VIDEOJUEGOS

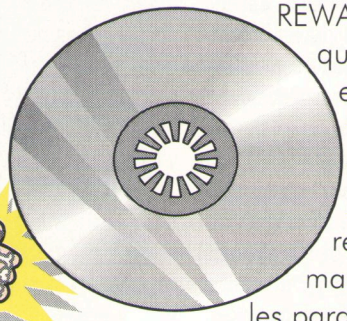
TRUCOLIST

pero ahora viene la bomba: El primer CDROM para progamadores de juegos



Trucolist CDROM sólo se vende por correo

Pídelo al número de teléfono 956-840246



Se trata de la colección de software más electrizante del mercado. Miles de programas de SHAREWARE y de DOMINIO PUBLICO que han sido seleccionados por expertos en la materia para que te ayuden en la confección de tus propios juegos.

Listados, trucos, juegos, editores de sprites, tutores de programación, editores de texto especiales para programadores, programas de música para interface MIDI, ficheros MOD, CMF, WAV y VOC que podrás incluir en tus programas, las ultimísimas demos que han salido al mercado... En definitiva todo lo que siempre has deseado obtener pero te era inaccesible. Ahora gracias a TRUCOLIST PC lo vas a tener todo junto en un único almacen de datos, en el que la mayoría de los programas no estarán comprimidos, para que los puedas usar directamente desde el CDROM, ahorrando mucho espacio en el disco duro. Anímate a entrar en el mundo maravilloso de la programación de juegos para ordenadores PC. ¡Ah! y gana muchísimo dinero haciéndolo.

Oferta: Su precio es de 8000 pts más gastos de envío, pero su precio especial de salida es de 5000 pts. Su edición es limitada y no se distribuye en librerías ni kioscos. Lee la tarjeta de la solapa de la revista para ver las condiciones de compra. (Esta oferta es sólo válida hasta 31-12-94)

Programación del Hardware del AMIGA

2ª PARTE

Por:

Alvaro Mateos

El Commodore Amiga es un ordenador, ideal como plataforma para juegos, de la mano de nuestro amigo Alvaro Mateos conoceremos los distintos coprocesadores que tiene esta máquina.

Continuando con la descripción y programación de los "custom chips" del AMIGA, vamos a explicar esta vez como está organizado el sistema de vídeo para diseñar nuestro modo de pantalla a medida.

Para empezar, vamos a definir una serie de términos que deben ser conocidos:

PlayField. Podríamos traducirlo como campo de juego o escenario donde imprimiremos los gráficos. El escenario no tiene porqué ser de las mismas dimensiones que el modo de pantalla que utilicemos y, de hecho, suele ser mayor para conseguir que los gráficos móviles aparezcan o desaparezcan lentamente por el monitor.

Bitplano. Un PlayField está compuesto por bitplanos. Un bitplano es una región de memoria que representa la imagen del escenario del juego. Cada píxel tendrá asociado un bit en cada uno de los bitplanos que conforman el PlayField. La combinación de estos bits dará el número de registro de color que tiene el píxel; por tanto, el número total de colores será de 2 elevado al número de bitplanos. El primer bitplano, que llamaremos BPL1, tendrá el bit menos significativo de cada píxel, mientras que el último bitplano tendrá el bit más significativo.

El AMIGA 500 nos permite usar un máximo de 4 bitplanos en alta resolución (un total de 16 colores) y hasta 6 bitplanos en baja resolución (64 colores).

Pantalla de Juego. Llamaremos así a la parte del escenario que será visible en nuestro monitor. Las dimensiones de la pantalla serán las del modo de vídeo que programemos. Con ayuda del COPPER podremos tener varias pantallas simultáneamente, cada una con una resolución o modo de vídeo diferentes.

Nosotros vamos a crear una pantalla de 640x200 píxels con 16 colores, que será nuestra pantalla de juego, y otra de 320x40 píxels con 32 colores, que utilizaremos como marcador.

A continuación veremos cómo los chips de vídeo entienden los bitplanos (en definitiva, bloques de memoria).

Como dijimos en el artículo anterior, el chip AGNUS recoge los datos de la memoria y los envía al chip DENISE; por tanto, AGNUS deberá tener un registro puntero por cada bitplano, y así es: los registros BPL1PT,BPL2PT,..., BPL6PT serán los punteros de cada uno de ellos; lógicamente, sólo tendremos que escribir en los registros de los bitplanos que vayamos a utilizar.

El AGNUS recoge los bytes por líneas. En baja resolución (320 píxels de ancho) cogerá bloques de 320/8 = 40 bytes de cada bitplano, para que DENISE dibuje la línea horizontal correspondiente. En alta resolución (640 píxels), deberá recoger el doble (80 bytes). Una vez que haya leido una línea de bytes, AGNUS suma a sus punteros el valor del registro BPLMOD. Este registro contiene lo que llamamos módulo. El módulo es el número de bytes que hay entre el último byte de una línea y el primero de la siguiente. Con este registro conseguimos que el puntero se coloque al principio de la línea siguiente cuando el escenario tiene unas dimensiones mayores que la pantalla. Existen dos registros de este tipo: BPL1MOD, que se sumará a los bitplanos impares, y BPL2MOD, que se sumará a los pares.

Para ilustrar todo esto con un ejemplo práctico, vamos a crear nuestro PlayField con unas dimensiones de 704x264 píxels. Como la pantalla mide 640x200, tendremos un margen de 32 píxels a cada lado (4 bytes a la izquierda y otros 4 a la derecha de cada línea, y 32 lineas de 88 bytes arriba y abajo). Nuestro módulo medirá, por tanto, 8 bytes. Por otra parte, deberemos calcular dónde empieza el primer byte de la pantalla dentro del escenario, luego tendremos que sumar (32*88)+4 a la dirección de inicio de cada bitplano.

Por último, hay que señalar que, debido a que los registros BPLxPT se van incrementando para la recogida de los bytes de cada bitplano, tendremos que volver a cargar estos registros cada vez que termina el refresco de pantalla; para ello haremos uso del COPPER.

PROGRAMACIÓN DEL MODO DE VÍDEO.

Para programar el modo de vídeo tenemos que escribir en los registros BPLCON0, BPLCON1 y BPLCON2. Para el ejemplo que estamos tratando sólo nos interesa el BPLCON0. De todas formas, haremos una breve descripción de cada uno de ellos:

BPLCON0.

- bit 15. Selecciona alta o baja resolución. (Alta = 1)
- bits 12,13,14. Estos tres bits indican el número de bitplanos que vamos a utilizar. Nosotros pondremos estos bits a 100 (4 bitplanos) para la pantalla de juego y a 101 (5 bitplanos) para el marcador.
- bit 11. Selecciona el modo HAM. Este es un modo especial del AMIGA que permite hasta 4096 colores simultáneamente en pantalla. Nosotros no utilizaremos este modo y pondremos un 0.
- bit 10. Este bit activa el modo "Dual PlayField", que permite el uso de dos escenarios; uno detrás del otro. Como no vamos a utilizarlo por ahora, pondremos este bit a 0.
- bit 2. Activa el modo entrelazado, que duplica la resolución vertical. Tampoco lo utilizaremos y pondremos aquí un 0.

Los otros bits, o bien no se usan, o se refieren a la utilización de Genlocks o Lápices Ópticos. Los pondremos todos a 0.

BPLCON1.

Este registro se utiliza para hacer el scroll horizontal de la pantalla. Como en principio no usaremos scroll, le daremos el valor $0000.

BPLCON2.

Con este registro se gestionan las prioridades de los sprites con respecto a los dos escenarios en modo "Dual PlayField". También lo pondremos a cero.

Con lo explicado hasta ahora es suficiente para crear nuestro modo de pantalla.

Existen otros registros que debemos programar y que nosotros cargaremos con valores estándar. Estos registros nos permiten sacar más partido a los diferentes modos de pantalla del AMIGA, pero su uso se escapa del objetivo de este artículo. De todas formas, daremos una breve descripción de ellos. A todo el que desee más información sobre éstos y otros registros, le remito al libro "AMIGA Hardware Reference Manual" de la editorial Addison Wesley.

DIWSTRT y DIWSTOP.

Estos dos registros seleccionan las coordenadas donde queremos centrar el display de pantalla.

DIWSTRT contiene las coordenadas de la esquina superior izquierda. El byte alto se corresponde con la coordenada vertical y el byte bajo con la horizontal.

DIWSTOP contiene las coordenadas de la esquina inferior derecha, de la misma forma.

Sobre esto tenemos que hacer una observación. El sistema de vídeo europeo (PAL) tiene 312 líneas, frente a las 262 del sistema americano (NTSC). Nosotros, que somos europeos, podríamos tener una resolución de hasta 312 líneas de altura en pantalla, pero en la práctica no se utilizan todas estas líneas debido a que el marco del monitor oculta alguna de ellas. Normalmente se suelen dejar 44 líneas de margen superior, así que pondremos $2c en la coordenada vertical superior. Para calcular la coordenada vertical inferior calculamos el número de líneas que nos hacen falta: 200 de la pantalla de juego, 4 líneas de separación y 40 del marcador, que hacen un total de 244 líneas, luego la coordenada vertical inferior valdrá 44+244 = 288. Los expertos en la materia se preguntarán ¿Como podemos meter en un solo byte ese número?. Eso no debe preocuparnos. En hexadecimal, 288 es igual a $120; quitamos el 1 y ya está, pues $20 es menor que $2c (la coordenada vertical superior) y el coprocesador entiende que queremos decir $120. Por tanto, ya tenemos los valores de las coordenadas verticales. ¿Que pasa con las horizontales?. En este caso utilizaremos los valores estándar para centrar la pantalla, que son $81 y $c1 respectivamente. Ya tenemos pues los valores de estos registros:

DIWSTRT = $2c81
DIWSTOP = $20c1

DDFSTRT y DDFSTOP.

En este registro cargaremos los valores estándar correspondientes, según sea alta o baja resolución:

- alta resolución $003c y $00d4
- baja resolución $0038 y $00d0

Programando adecuadamente estos registros podríamos conseguir nuevas resoluciones distintas a las normales del AMIGA. Así, por ejemplo, podríamos crear un modo de vídeo que ocupara toda la pantalla del monitor, sin bordes.

PROGRAMACIÓN DE NUESTRO PLAYFIELD.

Para programar nuestro PlayField, seguiremos los siguientes pasos:

1- Crear una "copper list" (ver artículo anterior), y situarla en CHIP RAM. La copper list escribirá en los registros que definen el modo de pantalla, en los registros de color, y en los punteros de cada bitplano. Después esperará a la línea 244 (44 lineas de borde superior + 200 líneas de la pantalla de juego) y cambiará al otro modo de pantalla, escribiendo el nuevo juego de registros.

2- Pedir memoria para cada uno de los bitplanos que necesitemos y guardar las direcciones de estos. En nuestro caso, necesitamos 4 bitplanos de (704/8)*264 = 23.232 bytes cada uno para la pantalla de juego. Para el marcador a 32 colores nos harán falta 5 bitplanos de (320/8)*40 = 1.600 bytes.

Como puede verse, ambas pantallas son totalmente independientes, y cada una está relacionada con un 'PlayField' distinto.

3- Escribir en la copper list (situada en CHIP RAM) las direcciones de inicio del primer byte de cada bitplano que sea visible en la pantalla. Escribiremos también la paleta de colores para la pantalla de juego y para el marcador.

4- Por último, escribir en el registro COP1LC la dirección de nuestra copper list.

El sistema del AMIGA utiliza su propia copper list. Por eso, si tenemos pensado volver al sistema, tendremos que guardar el puntero de su copper list para restaurarlo antes de salir.

En nuestro ejemplo, una vez creadas nuestras dos pantallas (la de juegos y el marcador), dibujamos en ambas unos cuadros de colores para ver que están ahí. Podemos cambiar los valores de esta rutina para variar los colores de los cuadros.

En otro artículo veremos como imprimir nuestros propios gráficos en ambos "PlayFields".

Hasta el mes que viene.

AMH.

```
;*********************************************************
;               PROGRAMACION DEL HARDWARE DEL AMIGA (II)
;               por Alvaro Mateos Herrera
;               Listado II.1
;*********************************************************
Inicio:
                bsr     Abre_GfxLib ; abrimos la libreria grafica
                tst.l   d0                      ; y si hay error
                beq     salir_lib       ; salimos
                bsr     Copia_Copper            ; pedimos memoria CHIP y copiamos
                tst.l   d0                      ; el "copper list",
                beq     salir_cop               ; si no hay memoria, salimos
                bsr     Memoria_Planos          ; pedimos memoria para los bitplanos
                tst.l   d0                      ; de la pantalla de juego y del marcador,
                beq     salir_pln               ; si no hay memoria, salimos
                bsr     Escribe_Copper          ; escribimos en el "copper list" la
                                                ; direccion de nuestros bitplanos, asi
                                                ; como la tabla de colores
                bsr     Dibuja_Cuadros          ; dibuja cuadros en la pantalla
                                                ; y el marcador
                bsr     Activa_Copper           ; activamos nuestra "copper list"
espera          btst.b  #6,$bfe001      ; esperamos a que se pulse el boton
                bne     espera                  ; izquierdo del raton
                bsr     Desact_Copper           ; desactivamos nuestra "copper list"
                                                ; y activamos la del sistema
                bsr     Libera_Planos           ; liberamos la memoria CHIP usada por
                                                ; los bitplanos
salir_pln:
                bsr     Libera_Copper           ; liberamos la memoria CHIP usada por
                                                ; el COPPER
salir_cop:
                bsr     Cierra_GfxLib           ; cerramos la libreria grafica
salir_lib:
                RTS
;*************************************
;               Copia Copper y Libera Copper
;*************************************
Copia_Copper:
; Reservamos memoria CHIP para nuestra "copper list"
                move.l  #copper_fin-copper_list,d0
                move.l  #MEMF_CHIP,d1
                move.l  Exec_base,a6
                jsr     AllocMem(a6)
                move.l  d0,copper_base
                beq     copia_ret
; Copiamos nuestra "copper list" a la memoria CHIP
                lea     copper_list,a0
                move.l  copper_base,a1
                move    #copper_fin-copper_list-1,d0
lop             move.b  (a0)+,(a1)+
                dbf     d0,lop
copia_ret:
                rts
Libera_Copper:
; Liberamos la memoria usada para la "copper list".
                move.l  copper_base,a1
                move.l  #copper_fin-copper_list,d0
                move.l  Exec_base,a6
                jsr     FreeMem(a6)
                rts
;*************************************
;               Pide memoria para los bitplanos
;*************************************
Memoria_Planos:
; Reservamos memoria CHIP para los 4 bitplanos
; de la pantalla de juego
                move.l  Exec_base,a6
                lea     tabla_BPjuego,a2
                moveq   #3,d2
lopj            move.l  #264*704/8,d0                   ; tamaño del Playfield en bytes
                move.l  #MEMF_CHIP,d1           ; ha de ser memoria CHIP
                jsr     AllocMem(a6)
                move.l  d0,(a2)+
                beq     planos_ret
                dbf     d2,lopj
; Reservamos memoria CHIP para los 5 bitplanos
; del marcador
                lea     tabla_BPmarcador,a2
                moveq   #4,d2
lopm            move.l  #40*320/8,d0; tamaño del marcador (320*40)
                move.l  #MEMF_CHIP,d1
                jsr     AllocMem(a6)
                move.l  d0,(a2)+
                beq     planos_ret
                dbf     d2,lopm
planos_ret:
                rts
Libera_Planos:
; Liberamos la memoria usada por los bitplanos.
                move.l  Exec_base,a6
                lea     tabla_BPjuego,a2
```

```
                moveq     #3,d2
lpj             move.l    #264*704/8,d0
                move.l    (a2)+,d1          ; si el puntero vale 0
                beq       libp_ret          ; es porque no se cogio memoria
                move.l    d1,a1
                jsr       FreeMem(a6)
                dbf       d2,lpj
                lea       tabla_BPmarcador,a2
                moveq     #4,d2
lpm             move.l    #40*320/8,d0
                move.l    (a2)+,d1
                beq       libp_ret
                move.l    d1,a1
                jsr       FreeMem(a6)
                dbf       d2,lpm
libp_ret:
                rts
;**************************************
;               Escribe en la "copper list"
;**************************************
Escribe_Copper:
                ; Escribe la direccion de los bitplanos en la "copper list"
                move.l  copper_base,a1
                lea       bitplanos_juego(a1),a1
                lea       tabla_BPjuego,a2
                moveq     #3,d0             ; 4 bitplanos
buc1            move.l    (a2)+,d1          ; cogemos la direccion del bitplano
                add.l     #88*32+4,d1 ; y nos saltamos 32 lineas por arriba
                                            ; del escenario y 4*8=32 pixels por la izqda.
                move      d1,(a1)           ; escribe la parte baja de la direccion
                add.l     #4,a1             ; apunta al siguiente registro
                swap      d1                ; coge la parte alta
                move      d1,(a1)           ; y la escribe
                add.l     #4,a1
                dbf       d0,buc1
                move.l  copper_base,a1
                lea       bitplanos_marcador(a1),a1
                lea       tabla_BPmarcador,a2
                moveq     #4,d0             ; 5 bitplanos
buc2            move.l    (a2)+,d1
                move      d1,(a1)
                add.l     #4,a1
                swap      d1
                move      d1,(a1)
                add.l     #4,a1
                dbf       d0,buc2
                ; Escribe los colores en la "copper list"
                move.l  copper_base,a1
                lea       colores_juego(a1),a1
                lea       Tabla_Colores_Juego,a2
                moveq     #15,d0            ; 16 colores
buc3            move      (a2)+,d1
                move      d1,(a1)
                add.l     #4,a1             ; apuntamos al siguiente registro
                dbf       d0,buc3
                move.l  copper_base,a1
                lea       colores_marcador(a1),a1
                lea       Tabla_Colores_Marcador,a2
                moveq     #31,d0            ; 32 colores
buc4            move      (a2)+,d1
                move      d1,(a1)
                add.l     #4,a1
                dbf       d0,buc4
                rts
;**************************************
;               Rellena los PlayFields de la pantalla
;               de juegos y del marcador con cuadros
;**************************************
Dibuja_Cuadros:
                move.l  tabla_BPjuego,a1    ; primer bitplano del juego
                move.l    #$ffffffff,d0
                bsr       rellena_planoj
                move.l  tabla_BPjuego+4,a1  ; segundo
                move.l    #$ffff0000,d0
                bsr       rellena_planoj
                move.l  tabla_BPjuego+8,a1  ; tercero
                move.l    #$ff0000ff,d0
                bsr       rellena_planoj
                move.l  tabla_BPjuego+12,a1 ;  y cuarto
                move.l    #$ff0000ff,d0
                bsr       rellena_planoj
                move.l  tabla_BPmarcador,a1 ; primer bitplano del marcador
                move.l    #$ff0000ff,d0
                bsr       rellena_planom
                move.l  tabla_BPmarcador+4,a1 ; segundo
                move.l    #$0000ffff,d0
                bsr       rellena_planom
                move.l  tabla_BPmarcador+8,a1 ; tercero
                move.l    #$ff0000ff,d0
                bsr       rellena_planom
                move.l  tabla_BPmarcador+12,a1 ; cuarto
                move.l    #$00ffff00,d0
```

```
                bsr             rellena_planom
                move.l  tabla_BPmarcador+16,a1          ; y quinto
                move.l          #$ffff0000,d0
                bsr             rellena_planom
                rts
rellena_planoj:
                moveq           #10,d3                  ; 11 cuadrados de alto
plj             bsr     rellena_fila_pjuego
                rol.l           #8,d0
                dbf             d3,plj
                rts
rellena_planom:
                moveq           #3,d3                   ; 4 cuadrados de alto
plm             bsr     rellena_fila_pmarcador
                rol.l           #8,d0
                dbf             d3,plm
                rts
;               escribe en un bitplano de la pantalla de juego
rellena_fila_pjuego:
                moveq           #23,d1                  ; rellena 24 filas
otro1           moveq           #21,d2                  ; de 22*4 bytes cada una
otro2           move.l          d0,(a1)+                ; escribimos de 4 en 4 bytes
                dbf             d2,otro2
                dbf             d1,otro1
                rts
rellena_fila_marcador:
;               escribe en un bitplano del marcador
rellena_fila_pmarcador:
                moveq           #9,d1                   ; rellena 10 filas
otro3           moveq           #9,d2                   ; de 10*4 bytes cada una
otro4           move.l          d0,(a1)+                ; escribimos de 4 en 4 bytes
                dbf             d2,otro4
                dbf             d1,otro3
                rts
;***************************************
;               Activa y Desactiva Copper
;***************************************
Activa_Copper:
                ; Cogemos la direccion de nuestra "copper list" y la
                ; escribimos en el puntero de instrucciones del COPPER.
                lea             CUSTOMS,a0
                move            #$0020,dmacon(a0)       ; desactivamos los sprites
                move.l          copper_base,d0
                move.l          d0,cop1lc(a0)
                rts
Desact_Copper:
                ; Cogemos la direccion de la "copper list" del sistema y la
                ; restauramos.
                lea             CUSTOMS,a0
                move            #$8020,dmacon(a0)       ; activamos los sprites
                move.l          Gfx_base,a1
                move.l          38(a1),cop1lc(a0)
                rts
;***************************************
;               Abre y Cierra la libreria de graficos
;***************************************
Abre_GfxLib:
                ; Abre la "graphics.library"
                lea             gfx_nombre,a1
                moveq           #0,d0                           ; cualquier version
                move.l          Exec_base,a6
                jsr             OpenLibrary(a6)                 ; si la libreria no se abre
                move.l          d0,Gfx_base             ; d0 valdra 0
                rts
Cierra_GfxLib:
                move.l          Gfx_base,a1
                move.l          Exec_base,a6
                jsr             CloseLibrary(a6)
                rts
;***************************************
;***************************************
;               Aqui esta nuestra "copper list"
;***************************************
;***************************************
copper_list:
                dc.w            bplcon0,$c000           ; alta resolucion, 4 bitplanos
                dc.w            bplcon1,$0000           ; sin scroll.
                dc.w            bplcon2,$0000           ; sin sprites.
                dc.w            diwstrt,$2C81           ; inicio en la pos. vert. 44 ($2c)
                dc.w            diwstop,$20C1           ; 288 lineas de alto: 44 de borde, 200 de juego,
                                                        ; 4 de separacion y 40 del marcador
                dc.w            ddfstrt,$003c ; valores estandar...
                dc.w            ddfstop,$00D4           ; ...para alta resolucion.
                dc.w            bpl1mod,$0008           ; modulo de los planos impares = 8
                dc.w            bpl2mod,$0008           ; modulo de los planos pares = 8
                ; direccion de los bitplanos de la pantalla de juego.
                ; la subrutina Escribe_Copper escribira aqui
bp_j            dc.w            bpl1ptl,$0,bpl1pth,$0
                dc.w            bpl2ptl,$0,bpl2pth,$0
                dc.w            bpl3ptl,$0,bpl3pth,$0
                dc.w            bpl4ptl,$0,bpl4pth,$0
                ; lista de los 16 colores para la pantalla de juego
```

¿¿¿Todavía no te has enterado???

Ya está a la venta la primera y única referencia técnica seria para programadores de videojuegos de PC. TRUCOLIST es un compendio de información destinado a todos aquellos que quieren adentrarse en la programación de juegos de ordenador. Sólo en sus páginas vas a encontrar la ayuda que necesitas y que no puedes hayar en ningún otro libro de informática. 300 páginas llenas de trucos, tablas, listados, artículos, divididos en apartados; VGA, SOUNDBLASTER, MEMORIA, ASSEMBLER, C, JUEGOS, etc.

Todo tipo de información para programadores

PROGRAMACIÓN PRÁCTICA DE VIDEOJUEGOS

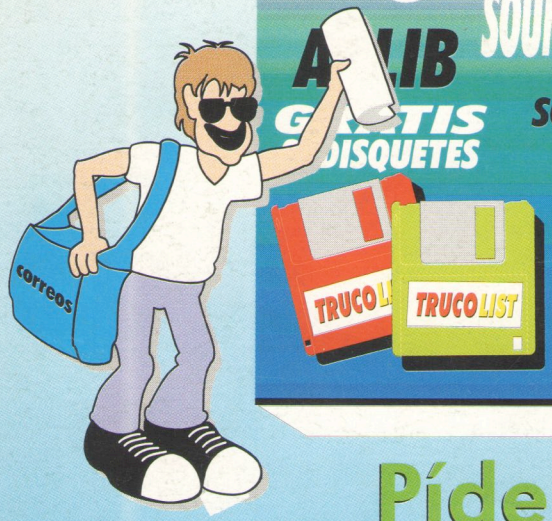
TRUCO LIST

Listados completos con explicación exaustiva

NÚMERO 1
Trimestre Oct-Nov-Dic 94

Movimiento SCROLL

SPRITES VGA HARDWARE FORMATOS GRÁFICOS SOUNDBLASTER INTERRUPCIONES ADLIB ANIMACIONES GRATIS DISQUETES SOFTWARE TABLAS DE REFERENCIA PRESENTACIONES TRUCOS PROFESIONALES



Gratis 2 disquetes



Pídelo al número

Todos los días, Sábados y Domingos inclusive



Horario de 0900 a 2100

956-840246

o con la tarjeta de la solapa de la revista

Agotada 1ª edición
a la venta la 2ª

Sólo venta por correo