Fenix
Manual de Referencia

José Luis Cebrián Pagüe

Agosto 2000
Versión 0.7

Índice General

1 Introducción

1.1 ¿Qué es Fenix?

Fenix es un lenguaje pseudo-interpretado, de código libre, diseñado para facilitar la tarea de programar juegos en dos dimensiones. Se trata de un lenguaje estructurado que soporta tipos de datos complejos (incluyendo cadenas, estructuras, arrays, punteros, y varios tipos de datos numéricos), multitarea cooperativa, división del proyecto en múltiples ficheros, y está inspirado en una mezcla de C y Pascal. La librería del sistema incluye un motor gráfico capaz de hacer fácilmente juegos en 8 y 16 bits de color con transparencias, rotado y escalado de *sprites*, zonas de scroll, distintos efectos, animaciones, así como un sistema de sonido basado en la librería MikMod, que añade a lo anterior el soporte de ficheros sonido WAV, módulos MOD, S3M, XM e IT, y sonido nativo de CD.

Este manual no trata de servir de introducción a aquellos de vosotros que no tenéis experiencia programando en otros lenguajes. Al contrario, incluye una explicación completa de cada función del sistema, variable u orden del lenguaje, con el objetivo de servir de punto de referencia completo y obligado a la hora de hacer programas en Fenix. Las novedades entre una y otra versión se documentarán en prioridad en las páginas de este manual.

Aconsejo leer muy atentamente este manual a todos aquellos de vosotros que habéis programado en DIV y queréis hacer vuestros futuros juegos en Fenix o incluso portar los ya existentes. Aunque hay una cierta compatibilidad entre los lenguajes, las diferencias son numerosas y no siempre evidentes. En cuanto a aquellos de vosotros que tenéis experiencia en C o Pascal, creo que con un vistazo a la sección del lenguaje y una lectura de las posibilidades de la librería, no tendréis problema en poneros manos a la obra.

Si nunca has programado en ningún lenguaje y decides empezar con Fenix y leyendo este manual, sufrirás graves heridas mentales que te marcarán para el resto de tu vida. Lee antes el tutorial escrito para el caso (actualmente en preparación).

1.2 Mejoras respecto a DIV actualmente implementadas

Esta lista de diferencias se refiere a DIV 1. DIV2 incorpora varias de estas mejoras, sin embargo de forma incompatible con Fenix.

- Soporte de modos gráficos de 16 bits (ver variable global GRAPH_MODE, y las ayudas sobre las funciones LOAD_PNG, NEW_MAP, etc) incluyendo nuevos formatos FPG y MAP de 16 bits.
- Soporte avanzado de sonido, gracias a la librería MikMod (módulos MOD, S3M, XM, IT, sonido de 16 bits, surround, reverberación... Ver ayuda sobre las variables globales SOUND_MODE, SOUND_FREQ, PANSEP, REVERB, VOLUME y las funciones LOAD_MOD, PLAY_MOD, etc)
- Soporte de tipos de dato, incluyendo tipos de datos enteros (INT, WORD, BYTE), de coma flotante (FLOAT), cadenas (STRING), estructuras (TYPE), punteros (POINTER)...
- Gestión de cadenas de un tamaño ilimitado, incluyendo liberación automática de cadenas no usadas o temporales (sin que el programador deba preocuparse), operadores de cadenas, y funciones específicas (SUBSTR, LEN, ATOI/ITOA, ATOF/FTOA, CHR, ASC, FIND, LCASE, UCASE...)
- Soporte de múltiples ficheros fuente (ver INCLUDE)
- Dibujo simple de primitivas (líneas, círculos y rectángulos) sobre el fondo de pantalla o sobre gráficos en memoria.

- Acceso directo a los datos de los gráficos en memoria, a través de punteros (función MAP_BUFFER) así como creación en tiempo de ejecución de mapas nuevos para editar (NEW_MAP, WRITE_IN_MAP, map_clone)
- Gestión de memoria dinámica (ALLOC, FREE y REALLOC)
- Carga de ficheros PCX y PNG, además de MAP (LOAD_PNG, LOAD_PCX)
- Acceso directo a ficheros (FOPEN, FWRITE, FREAD...)
- Edición en tiempo de ejecución de grupos de colores de la paleta mediante las funciones GET_COLORS y SET_COLORS
- Apilado de ventanas de scroll una sobre la otra, y opción de transparencia en los planos de scroll.
- Efectos especiales para modo de 16 bits: transparencia variable, colorizado de gráficos y del fondo, iluminación, oscuridad...
- Funciones de búsqueda de caminos
- Soporte directo de animaciones, dentro de los propios ficheros MAP y FPG. Posibilidad de convertir un GIF animado en un fichero MAP, o de crear una animación manualmente.

Así mismo, se han incorporado algunas rutinas de utilidad como TEXT_WIDTH, TEXT_HEIGHT, COS/SIN/TAN, ACOS/ASIN/ATAN, RGB o EXISTS, y se han ampliado algunas rutinas y estructuras internas con nuevas posibilidades, como GRAPHIC_INFO.

1.3 Consideraciones sobre el futuro de Fenix

Aún hay unas cuantas cosas pendientes para realizar antes de la versión 1.0:

- Existirá una utilidad para añadir ficheros a un DCB, en lugar de la opción -a
- Existirá una utilidad para poder crear tipos de letra FNT
- Se soportará un formato FNT de 16 bits (extendido al igual que el MAP o el FPG)
- Se introducirá la posibilidad de ejecutar funciones externas ubicadas en DLL
- Compilador e intérprete estarán disponible como librería o DLL
- El control de errores se mejorará sustancialmente en el intérprete
- El intérprete incorporará un sencillo *debugger*

Cuando la versión 1.0 se haga realidad, se abrirá la discusión acerca de qué novedades incorporar al lenguaje. Hay algunas que han quedado pendientes de incluir en esta versión y que sin duda deberán ir en la 2.0, ya que son considerablemente importantes:

- Grabaciones de estado ("snapshots")
- Mapas compuestos por otros gráficos (casillas, mapas isométricos o hexagonales, etc)
- Un entorno de edición de los programas

1.4 Cambios introducidos en Fenix 0.7

- Primera edición de este manual. Muchas funciones que permanecían indocumentadas han sido documentadas por primera vez.
- Nuevas funciones blendop (página 58) que permiten realizar efectos de iluminación, colorizado y transparencia (sólo en modo de 16 bits).
- Funciones de búsqueda de caminos (página 57) de gran utilidad para juegos de estrategia o aventuras gráficas, permiten sortear obstáculos en busca de un destino empleando un mapa de durezas u obstáculos.
- Soporte de ficheros MAP animados (ver graphic_info, página 39, o graphic_set a continuación de ésta).
- Soporte de ficheros comprimidos (ver fopen, página 63). También se ha añadido la nueva función de utilidad file (página 66).
- Posibilidad de dibujar gráficos sin el color 0 como transparente (flag 128). Afecta a todas las funciones como map_xput que reciben un parámetro de flags, y a la variable local flags.
- El enlace con la librería mikmod ha sido reescrito para una mayor estabilidad en el funcionamiento del sonido. Además la función sound (página 70) admite ahora nuevas posibilidades de reproducción de una muestra de sonido.

1.5 Errores conocidos en la versión 0.7

Siendo ésta una versión beta, todavía hay errores por solventar. Algunos de ellos son conocidos. La página web de Fenix, ubicada en http://fenix.sourceforge.net/, contiene una página de *bugs* que lista éstos errores a medida que se van encontrando. Es imprescindible su visita para todos aquellos que deseéis probar una versión beta.

Aquí ofrezco una lista de los bugs conocidos en el momento de escribir este manual:

- Las rutinas de búsqueda de caminos no están aún completamente acabadas, y se han documentado algunos fallos que no he tenido tiempo de depurar.
- Las rutinas de dibujo funcionan incorrectamente si se dibuja un gráfico con un ángulo muy pequeño o cercano al límite de un cuadrante (pero distinto de 0, 90, 180 o 270 grados).
- Se han detectado algunos problemas con el sonido en la versión Windows, especialmente cuando la frecuencia (variable sound_freq) es superior a la normal de 22000 Hz.
- La opción -a del compilador FXC no funciona, debido a cambios en las rutinas de acceso a ficheros al añadir las opciones de compresión.
- En Windows hay problemas al pulsar ALT+TAB en un DCB a pantalla completa.
- El soporte GIF apenas ha sido probado y no puede considerarse aún acabado. Se han documentado igualmente algunos problemas.

1.6 Estado de la documentación

A fecha de hoy (15 de Agosto del 2000) este manual aún no está completo. La siguiente es una lista de las cosas que falta por incorporar:

- Una sección que explique el funcionamiento del sistema gráfico y del intérprete en general
- Una explicación detallada del funcionamiento de todos los operadores
- Un apéndice que describa el formato de todos los tipos de fichero empleados

Varias personas han colaborado en tareas como escribir programas de ejemplo y traduciendo la documentación anterior a este manual. Mi objetivo consiste en unificar estas colaboraciones ofreciendo una versión en inglés del manual y distribuyendo los ejemplos junto al manual.

Además, planeo escribir un manual de introducción que sirva como tutorial al lenguaje. Durante este manual se explicará paso a paso cómo crear un juego sencillo, destinado a personas con pocos o ningún conocimiento previo de programación.

2 Referencia del lenguaje

2.1 Definiciones

Un programa en Fenix consta de los siguientes apartados:

- Declaración del nombre del programa (opcional)
- Declaración de tipos de datos definidos por el usuario (opcional)
- Declaración de constantes (opcional)
- Declaración de variables globales (opcional)
- Declaración de variables locales (opcional)
- Procesos (opcional)
- Proceso principal

En los siguientes apartados iremos definiendo cada uno de estos conceptos. A su vez, un proceso consta de:

- Declaración de nombre y tipo de dato de retorno del proceso
- Declaración de parámetros del proceso
- Declaración de variables privadas (opcional)
- Sentencias

La única diferencia entre el proceso principal y el resto estriba en que no se declara nombre, tipo de dato de retorno, ni parámetros del proceso. Sólo las secciones de declaración de datos privados y código se encuentran presentes.

¿Qué es una variable?

Una variable es una zona de memoria donde se almacena un valor en uso por el programa. Ese valor puede ser un número como 0 o 1240, o un texto como "Hola, mundo". Sin embargo estos datos pueden tener una naturaleza muy diferente: no es lo mismo guardar un número en memoria que un texto, e incluso podemos tener la flexibilidad de emplear más o menos memoria en guardar datos numéricos, para obtener más precisión o posibles números.

Debido a esto todas las variables tienen un *tipo*. El tipo indica qué clase de dato contiene la variable, y también permite averiguar cuánta memoria ocupa. Algunos tipos de dato básicos en Fenix son el tipo de dato int, que permite almacenar un número entero (sin decimales), o el tipo de dato string, que permite almacenar una secuencia de caracteres, también llamada una *cadena* de texto.

Para distinguir una variable de otra se le asigna un *nombre*. Dicho nombre debe empezar por una letra o por el signo _, y puede contener una combinación de letras, números o el signo _.

Cada programa necesita emplear variables diferentes. El proceso de crear una variable, reservando una zona de memoria para ella y asignándole un tipo de dato y un nombre identificativo, es el proceso de *declarar* una variable.

¿Qué es una variable global?

Las variables globales están disponibles a lo largo de todo el programa. Una variable global se crea en cuanto el programa se ejecuta, y permanece en memoria mientras el programa siga en ejecución. En cualquier momento del programa se puede leer o modificar el dato que contiene, accediendo a la variable por su nombre.

¿Qué es un proceso?

Fenix es un lenguaje estructurado, lo cual quiere decir que el código del programa no se acumula en desorden de principio a fin, sino que se organiza en secciones pequeñas llamadas procesos. Un proceso puede realizar operaciones clave como sacar un menú en pantalla o hacer un cálculo, y ser ejecutado en cualquier momento por otros procesos, incluso varias veces.

En Fenix los procesos tienen una utilidad añadida, radicalmente diferente a las funciones o procesos de otros lenguajes: un proceso hace a su vez de objeto en pantalla. Por ejemplo, un proceso puede ser el gráfico que identifica al jugador, otro proceso puede ser un disparo que se desplaza por pantalla... Este tipo de procesos se diferencia del resto en que permanece en funcionamiento *a la vez* que otros procesos, en multitarea cooperativa. Esta facilidad permite escribir código que desplace un enemigo en pantalla, y crear fácilmente múltiples enemigos haciendo varias llamadas al proceso, además de permitir que el enemigo sea independiente del resto de objetos en pantalla.

¿Qué es un parámetro?

Un proceso puede recibir parámetros. Los parámetros son variables que sólo conoce el propio proceso, y que son asignadas en el momento de ejecutarlo, desde otro proceso. Los parámetros permiten hacer procesos cuyo comportamiento se ajusta a nuestras necesidades. Por ejemplo, un proceso que muestra un enemigo en pantalla puede recibir como parámetros las coordenadas donde mostrarse, de manera que resulta mucho más sencillo crear enemigos en distintas posiciones.

¿Qué es el valor de retorno?

Un proceso puede devolver un valor al proceso que lo llamó, lo cual permite crear procesos que, en lugar de hacer de objetos en pantalla, realicen cálculos matemáticos o de otra índole con los parámetros que reciban y devuelvan el resultado al proceso que lo llamó.

¿Qué es una variable privada?

A menudo es necesario emplear variables en el interior de procesos para almacenar datos que sólo son de utilidad al nuevo proceso. Una variable privada se crea en el momento de llamar al proceso, y es destruída en el momento de abandonarlo. Cada proceso tiene sus propias variables privadas: incluso si se ejecuta múltiples veces un mismo proceso, cada copia de este proceso en memoria tiene sus propias copias de estas variables privadas, cuyos datos son independientes de cualquier otra copia del proceso que haya en ejecución. Las variables privadas se declaran en el momento de declarar un proceso.

¿Qué es una variable local?

Las variables locales se parecen a las privadas porque cada proceso que se crea contiene una copia de cada variable local, y ésta es creada e inicializada en el momento de crear el proceso, e igualmente se destruye cuando el proceso finaliza.

Sin embargo, las variables locales se declaran al inicio del programa, y tienen la particularidad de que todos los procesos creados comparten las mismas variables locales.

La utilidad de las variables locales reside en que un proceso puede examinar las variables locales de otro proceso distinto. Existiendo variables locales con datos tales como las coordenadas en pantalla de un proceso, es sencillo hacer código que compruebe en un proceso cuál es la distancia

con otro proceso. Por lo tanto, las variables locales serán siempre variables de utilidad general, en uso por gran cantidad de procesos del juego.

2.2 Tipos de dato

byte

El tipo de dato BYTE representa un byte de memoria. Puede contener un número en el rango 0-255 inclusive. Su utilización típica es guardar un pixel en el modo de 256 colores, aunque también tiene otras utilidades (por ejemplo, guardar componentes de color de la paleta).

Un BYTE se opera como un entero, y pueden hacerse sobre él cualquiera de las operaciones que son válidas con enteros, incluyendo rotaciones y operaciones a nivel de bit. A menudo ocurrirán problemas de truncamiento o de rango. Por ejemplo, en un byte "255+3" es igual a "2", ya que no existen bytes con valor superior a 255 y 255+1 pasa a ser 0.

Un dato de tipo byte ocupa 8 bits, un byte (obvio).

word

El tipo de dato WORD es un entero de rango limitado, que ocupa dos bytes de memoria. Puede contener un número en el rango 0 a 65535 includive. Su mayor utilidad consiste en crear tablas de datos (como coordenadas para movimientos, por ejemplo) ya que ocupa la mitad de memoria que un dato INT.

Un WORD es un entero y puede operarse como tal. Hay que tener en cuenta que los números negativos no pueden ser almacenado en un dato de tipo WORD, e intentarlo dará como resultado que la variable contenga un valor positivo diferente.

Un dato WORD ocupa 16 bits de memoria (2 bytes).

int

El INT es el tipo de dato básico de Fenix. Cuando una variable se declara sin indicar el tipo de dato al que pertenece, se crea como un INT. La mayoría de datos locales y globales predefinidos son INTs. Un INT es un entero de 32 bits (4 bytes) con un rango entre -2147483648 y 2147483647.

Los tipos de dato INT admiten todas las operaciones matemáticas habituales, tanto aritméticas (+, -, *, /, MOD) como a nivel de bit (<<, >>, XOR, OR, AND, NOT).

Un INT ocupa 32 bits de memoria (4 bytes).

El INT es el tipo de dato por defecto. Cuando se declara una variable simplemente indicando su nombre, sin poner un tipo de dato, se está declarando una variable de tipo INT.

float

El tipo de dato FLOAT representa un número en coma flotante. Dentro del código, un número FLOAT se distingue de un INT porque lleva el punto decimal. De esta forma "2.0" es un dato float, mientras "2" es un entero.

Los datos float pueden operarse de forma similar a los enteros. Algunas operaciones (operaciones binarias, como AND/OR/XOR/NOT o rotaciones) no están permitidas con datos float. Los float pueden asignarse a variables enteras o viceversa, en cuyo caso Fenix hará cualquier conversión o truncamiento que crea necesaria.

Un FLOAT ocupa 32 bits de memoria (4 bytes).

struct

Fenix permite declarar variables tipo *estructura*. Una estructura es una variable que contiene otras variables. Su utilidad está clara en casos en que conviene guardar una serie de datos relacionados con un mismo objeto o concepto. Por ejemplo, la estructura mouse contiene variables con datos

relacionados sobre el ratón; así, la variable mouse.x contiene la coordenada X en pantalla del ratón, mouse.y contiene lo propio con la coordenada Y, etc.

Para declarar una estructura se emplea la siguiente sintaxis:

```
STRUCT nombre
    variable ;
    variable ;
    ...
END
```

Las variables contenidas por la estructura pueden ser de cualquier tipo, incluso otras estructuras; se sigue la sintaxis normal de declaración de variables, con la posibilidad de inicializar sus componentes de igual manera que se de variables normales se tratasen. Por ejemplo la estructura mouse se declara de una forma similar a

```
STRUCT mouse
    int x, y ;
    int graph ;
    int file ;
    int z = -512 ;
    int angle ;
    int size = 100 ;
    int flags ;
    int region ;
    int left, middle, right ;
END
```

La cantidad de memoria ocupada por una estrucutra es igual a la suma de la memoria ocupada por cada uno de sus elementos.

Estructuras homogéneas: Una estructura es homogénea sí y sólo si todos sus datos pertenecen al mismo tipo. Por ejemplo, la estructura anterior del ejemplo (mouse) es homogénea, dado que todos sus datos son del tipo int. Si añadiéramos a la estructura un dato tipo float o byte, ésta dejaría de ser homogénea. La diferencia es importante de cara a la inicialización de variables y estructuras: una estructura no homogénea no puede inicializarse en la declaración empleando el signo =.

type

Además de las estructuras, Fenix permite declarar *tipos de dato definidos por el usuario*. En realidad, uno de estos tipo de dato es equivalente a una estructura, con la particularidad de que pueden crearse con mayor facilidad, ya que a la estructura en sí se le asigna un nombre.

Los tipos de dato definidos por el usuario se declaran fuera de las zonas de datos globales o locales, al comienzo del fichero de programa, siguiendo la sintaxis:

```
TYPE nombre
    ... variable ...
    ... variable ...
END
```

El hecho de declarar el tipo de dato en sí, no crea ninguna variable. Pero a partir de entonces disponemos de un tipo de dato *nombre*, que podemos usar como si de un tipo de dato normal se tratase. Cada dato de ese tipo que creemos será un realidad una estructura con los componentes tal cual aparecen en la declaración del tipo. Por ejemplo:

```
TYPE COLOR
    byte r, g, b ;
END

GLOBAL
    colorñegro;
END
```

pointer

Fenix permite crear variables de tipo puntero. Una variable de este tipo contiene la dirección de memoria de algún dato, y permite, a través de ésta, realizar modificaciones o hacer servir dicho dato.

Todos los punteros ocupan 32 bits de memoria (4 bytes), independientemente del tamaño del dato al que apunten.

Se pueden crear punteros a cualquier de los tipos de datos simples (int, word, byte, float o string), a otro puntero, o a un tipo de dato definido por el usuario.

Declaración de punteros Para declarar una variable tipo puntero, debe especificarse el tipo de dato al que señala, seguido de la palabra "pointer". He aquí algunos ejemplos de punteros válidos:

```
/* Punteros a datos simples */
int pointer iptr ;
word pointer sptr ;
byte pointer bptr ;
float pointer fptr ;
string pointer sptr ;

/* Punteros a punteros */

int pointer pointer ptr ;

/* Puntero a un dato definido por el usuario */

TYPE color
    BYTE r ;
    BYTE g ;
    BYTE b ;
END

color pointer cptr ;
```

Asignación de datos a un puntero Para asignar a un puntero la dirección de memoria de una variable existente, puede emplearse el operador & (OFFSET):

```
PRIVATE
    int contador ;
    int pointer ptr ;
BEGIN
    ptr = &contador ;
END
```

Hay que tener en cuenta que no es posible asignar a un puntero la dirección de memoria de una variable de otro tipo. No se puede asignar a un puntero de tipo INT la dirección de memoria de una variable tipo word, por ejemplo.

Acceso al dato referenciado por un puntero Puede accederse al dato referenciado por un puntero utilizando el operador *, o bien su forma de corchetes [...]. Por ejemplo:

```
*ptr = 2 ;     // Asigna un 2 a la variable a la que apunta ptr
[ptr] = 2 ;    // Equivalente a lo anterior
say (*ptr) ;   // Muestra en pantalla dicho contenido
```

La misma sintaxis permite emplear el dato en una evaluación, como realizar modificaciones al mismo mediante el operador "=".

Punteros a arrays El mismo tipo de puntero que se emplea para apuntar a un entero, puede hacerse servir para apuntar a un array. En este caso, se le asigna al puntero la dirección del primer elemento del array, y si se modifica mediante el mecanismo visto anteriormente, es el primer elemento del array el que sufre las modificaciones.

```
PRIVATE
    int datos[99] ;
    int pointer ptr ;
BEGIN
    ptr = &datos ;
    *ptr = 0 ;   // Afecta a datos[0]
    say(*ptr) ;  // Muestra datos[0]
END
```

Se incluye una facilidad, prestada del C, que permite utilizar con un puntero la sintaxis de direccionamiento de arrays. Esto permite acceder a los elementos del array por encima del 0. Siendo ptr un puntero de cualquier tipo que señala a un array, ptr[n] sirve para hacer referencia al elemento n del array. Por ejemplo:

```
PRIVATE
int datos[99] ;
    int pointer ptr ;
BEGIN
    ptr = &datos ;
    ptr[1] = 0 ;   // Afecta a datos[1]
    say(ptr[1]) ;  // Muestra datos[1]
END
```

Esta sintaxis permite acceder a elementos del array más limpiamente, como si ptr en sí fuera un array. Si se desea crear un proceso que necesite o modifique un array, se puede hacer que el proceso reciba un puntero al comienzo del mismo:

```
process rellena (int pointer array, intñ, int valor)
begin
    while (n-- > 0)
         array[n] = valor;
    end
end

private
    int cosas[99] ;
```

```
begin
    rellena (&cosas, 99, 24) ;
    say (cosas[1]) ; // Muestra "24"
end
```

Hay que tener cuidado en este caso de llamar al proceso empleando el operador "&", ya que recordemos que emplear el nombre de un array equivale a utilizar su primer elemento.

Recorriendo la memoria Es posible alterar directamente el valor de un puntero, además de asignándole un valor directamente, mediante los operadores de suma y resta. Ambos requieren que el valor sumado al puntero sea un entero.

Cuando a un puntero se le suma el número "n", la operación significa adelantar el puntero en la memoria n posiciones.

Es más rápido utilizar el operador "*" sobre un puntero para acceder al valor que señala, que emplear la indirección como en el apartado anterior. Realizando sumas al puntero, se puede recorrer un array de forma más rápida. Por ejemplo, el proceso anterior podría reescribirse de manera ligeramente más rápida así:

```
process rellena (int pointer array, intñ, int valor)
begin
    while (n-- > 0)
        *array++ = valor;
    end
end
```

También se puede emplear la suma de enteros a un puntero para asignar a otro puntero una dirección de memoria:

```
ptr2 = ptr1 + 4 ; /* Salta 4 elementos */
```

Punteros a punteros Los punteros a punteros son un concepto avanzado que sólo resulta útil para aquellos programadores que quieran mantener estructuras complejas de datos empleando memoria dinámica.

En realidad no hay nada especial en la forma de hacerlos servir. Sólo hay que tener en cuenta que con ellos el dato apuntado es a su vez un puntero que puede tratarse como cualquier otro, y para acceder al dato final es preciso utilizar dos veces el operador * o [...].

```
PRIVATE
    intñ ;
    int pointer ptr ;
    int pointer pointer ptr2 ;
BEGIN
   ñ = 12 ;
    ptr = &n ;
    ptr2 = &ptr ;
    say (**ptr2) ; /* Muestra 12 */
    say ([[ptr2]]) ; /* Muestra 12 */
END
```

array

Cualquier variable puede declararse como array. Para ello basta con escribir, tras la variable, uno o más juegos de dimensiones con la sintaxis [*n*], donde n es el número final de índice de una dimensión (los elementos se numeran desde 0 a n, con lo que la dimensión contendrá $n + 1$ elementos.

Por ejemplo, para declarar una tabla de 10x10 números enteros basta con hacer:

```
INT tabla[9][9];
```

Es posible acceder al primer elemento del array sin necesidad de especificar su número. Así, siguiendo el ejemplo anterior es lo mismo escribir cualquier de estas tres combinaciones

```
tabla
tabla[0]
tabla[0][0]
```

Existe un límite en la cantidad de dimensiones que se admiten para un array. Normalmente este límite es de cinco dimensiones, aunque puede reducirse a menos en el caso de arrays de tipos de datos más complejos, como punteros o estructuras.

Un array ocupa tanta memoria como ocupe el dato original, multiplicada por el número total de elementos presentes en el array.

Arrays de estructuras Una estructura puede a su vez ser un array, es decir, contener más de una vez los datos de su interior. Para hacer esto basta con escribir [*n*] a la derecha del nombre de la estructura, donde n es el número máximo de índice (los números de índice empiezan por 0, por lo que una estructura declarada con [1] tendrá dos entradas, la entrada [0] y la [1]).

Es posible declarar arrays multidimensionales de estructuras escribiendo uno o más grupos de corchetes detrás del primero. Por ejemplo, el siguiente código crea una tabla de 10x10 colores:

```
STRUCT colores[9][9]
    byte r, g, b;
END
```

Para acceder posteriormente a un elemento de la tabla ya dentro del programa, es preciso indicar su posición mediante un código como

```
colores[0][0].r
```

2.3 Expresiones

La flexibilidad de los lenguajes actuales está en que, en la mayoría de casos, tenemos la libertad de escribir expresiones allí donde hubiera lugar para un número o un valor. Por ejemplo, a la hora de asignar un valor a una variable, podemos obtener ese valor empleando una expresión.

Las expresiones son esencialmente cálculos matemáticos realizados con variables y valores constantes, indistintamente. Para poder realizar estos cálculos, Fenix admite una serie de *operadores*.

Prioridad de los operadores

La siguiente tabla indica la prioridad de los operadores. En una expresión, los operadores de prioridad más baja se ejecutan primero: esto explica por qué $2 * 2 + 3 * 3$ es igual a 13. En este ejemplo, el operador $*$ tiene una prioridad más baja que el operador $+$, por lo que las multiplicaciones se operan primero, para luego resolverse la suma de sus resultados $(2 * 2) + (3 * 3) = 4 + 9 = 13$. Cuando dos operadores tienen la misma prioridad, los cálculos se resuelven de izquierda a derecha: por ejemplo, $2 + 4 - 3 + 1 = 6 - 3 + 1 = 3 + 1 = 4$. Siempre es posible emplear paréntesis para alterar el orden de prioridad de las operaciones, por ejemplo escribiendo $2 * (2 + 3) * 3$ obtendremos el resultado esperado (30).

La siguiente tabla indica los grupos de prioridad de los operadores. Los operadores de prioridad más baja aparecen primero, y todos los operadores en el mismo grupo tienen a misma prioridad.

Grupo	Operador	Descripción
1	()	Subexpresión
	TYPE	Tipo de proceso

Grupo	Operador	Descripción
	ID	Identificador de proceso
	OFFSET &	Dirección de memoria
	POINTER []	Direccionamiento de punteros
	SIZEOF()	Tamaño de variable
2	-	Negación
	.	Acceso a miembro de estructura o variable local
	[]	Direccionamiento de arrays o punteros
	++	Incremento
	–	Decremento
3	*	Multiplicación
	/	División
	%	Resto
4	+	Suma
	-	Resta
5	<<	Rotación binaria a la izquiera
	>>	Rotación binaria a la derecha
6	>	Mayor
	<	Menor
	>=	Mayor o igual
	<=	Menor o igual
	<>	Distinto
	==	Igual
7	AND && &	Y lógico
	OR || |	O lógico
	XOR ^	O exclusivo
8	=	Asignación
	+=	Suma a una variable
	-=	Resta a una variable
	*=	Multiplicación de una variable
	/=	División de una variable
	%=	Resto con asignación
	&=	Y lógico a una variable
	|=	O lógico a una variable
	^=	O exclusivo a una variable
	<<=	Rotación binaria de una variable a la izquierda
	<<=	Rotación binaria de una variable a la derecha

Observa que algunos operadores tienen sinónimos. Por ejemplo, es equivalente escribir && o AND en el código fuente.

Cierto/falso

Una expresión se considera **cierta** si es un número entero cuyo bit menos significativo está a 1, y se considera **falsa** en caso contrario. Los números impares cumplen la condición: el número 1 es una expresión cierta, mientras el número 0 es una expresión falsa. Los números en coma flotante o las cadenas se convierten a enteros a la hora de realizar la comprobación.

Los operadores de comparación (>=, <=, <>, ==, >, <) devuelven un número entero que será 1 ó 0 en función de si la comparación resulta cierta o falsa. Los operadores AND y OR realizan operaciones lógicas cuyo resultado es apropiado para juntar comparaciones: el resultado de AND es cierto sí y sólo sí los resultados de las expresiones a izquierda y derecha del AND son ciertos, mientras el resultado de OR es cierto sí y sólo sí al menos uno de los dos resultados es cierto.

Saber si una expresión es cierta o falsa es de utilidad sobre todo en sentencias como IF o WHILE. Fuera de estas sentencias, no tiene ningún efecto.

2.4 Sentencias

IF ... END

```
IF (expresión)
sentencia ;
...
    sentencia ;
END
```

La sentencia IF comprueba si una expresión es **cierta** y en el caso afirmativo, ejecuta una serie de una o más sentencias, todas ellas acabadas en punto y coma (;). Si la expresión fuese **falsa**, ninguna de estas sentencias se ejecutaría.

IF ... ELSE ... END

```
IF (expresión)
    sentencia ;
    ....
    sentencia ;
ELSE
    sentencia ;
    ...
    sentencia ;
END
```

Es posible añadir un bloque ELSE a una sentencia IF, de forma que a la sentencia IF le siguen dos bloques de sentencias. Si la expresión es **cierta** se ejecutará el bloque de sentencias ubicado entre el IF y el ELSE. En caso contrario, se ejecutará el bloque de sentencias entre el ELSE y el END.

Es posible no poner paréntesis y, en su lugar, acabar el IF con el signo :. Sin embargo, si los paréntesis se encuentran, deben englobar a toda la expresión del IF. Por ejemplo, la siguiente sentencia no es válida, ya que comienza por paréntesis:

```
IF (x+1)*2 < y:
```

En su lugar, puedes escribir la misma expresión de las siguientes dos maneras:

```
IF ((x+1)*2 < y)
IF 2*(x+1) < y:
```

LOOP ... END

```
LOOP
    sentencia ;
    ...
    sentencia ;
END
```

La sentencia LOOP ejecuta el bloque de sentencias acabadas en punto y coma indefinidamente, es decir, cada vez que se ejecute la última sentencia el programa volverá al comienzo del bloque a ejecutar la primera. La única forma de salir del bucle es mediante la sentencia BREAK, que abandona inmediatamente la ejecución del bucle LOOP y continúa la ejecución a partir de la primera sentencia posterior al bucle. La sentencia CONTINUE también puede hacerse servir en el interior del bucle LOOP, en este caso para saltar inmediatamente al comienzo del mimo, ignorando las sentencias que quedaran hasta el final del bloque.

WHILE ... END

```
WHILE (expresión)
    sentencia ;
    ...
    sentencia ;
END
```

La sentencia WHILE comprueba la expresión indicada y, si es cierta, ejecuta el bloque de una o más sentencias acabadas en punto y coma. Al acabar su ejecución, se vuelve a comprobar la expresión, y mientras siga siendo cierta, el proceso se repetirá indefinidamente. Si la expresión es falsa, las sentencias no se ejecutan ni siquiera una primera vez.

Es posible abandonar inmediatamente un bucle WHILE empleando la sentencia BREAK. También es posible utilizar la sentencia CONTINUE, que hace que el bucle vuelva al comienzo (la expresión vuelve a evaluarse y si es cierta, el bloque de sentencias vuelve a ejecutarse desde el comienzo; si fuese falsa, CONTINUE actuaría igual que BREAK).

En un WHILE los paréntesis también reciben el mismo tratamiento que con IF.

REPEAT ... UNTIL

```
REPEAT
    sentencia ;
    ...
    sentencia ;
UNTIL (sentencia) ;
```

La sentencia REPEAT ejecuta el bloque de una o más sentencias acabadas en punto y coma. Al acabar su ejecución, se comprueba la expresión, y mientras sea cierta, el bloque volverá a ejecutarse una y otra vez hasta que al final de su ejecución la evaluación de la expresión sea falsa. A diferencia del bucle WHILE, las sentencias siempre se ejecutan al menos una vez.

Es posible abandonar inmediatamente un bucle REPEAT empleando la sentencia BREAK. También es posible utilizar la sentencia CONTINUE, que hace que el bucle vuelva al comienzo (el bloque de sentencias vuelve a ejecutarse desde el comienzo y al sólo final de su ejecución la expresión será evaluada de nuevo; CONTINUE nunca actúa como BREAK en un bucle REPEAT).

SWITCH ... END

```
SWITCH (expresión)
    CASE expresión:
        sentencia ;
        ...
        sentencia ;
END
    CASE expresión, expresión, ... :
        sentencia ;
        ...
        sentencia ;
        END
    CASE expresión .. expresión:
        sentencia ;
        ...
        sentencia ;
        END
    DEFAULT:
        sentencia ;
```

```
        ...
        sentencia ;
        END
END
```

La sentencia SWITCH evalúa una expresión, y permite realizar diferentes operaciones en función del resultado de la misma. Entre el SWITCH y el END deben aparecer uno o más bloques CASE ... END, cada uno de los cuales estableciendo un curso de acción en función del resultado de la expresión SWITCH. En un CASE se puede indicar:

- Una expresión
- Varias expresiones, separadas por comas
- Un rango de expresiones (dos expresiones separadas por dos puntos "..")

Las sentencias entre el CASE y el END se ejecutaran si y sólo si la expresión resultante del SWITCH es una de las ennumeradas o entra dentro del rango especificado.

El bloque especial DEFAULT debe ser el último, y establece el curso de acción a seguir si ninguno de los bloques CASE anteriores se ejecuta.

La sentencia SWITCH sólo permite actualmente trabajar con datos de tipo INT. Si la expresión del SWITCH devuelve un dato de otro tipo, éste será internamente convertido a INT.

FOR ... END

```
FOR (expresión ; expresión ; expresión)
    sentencia ;
    ...
    sentencia ;
END
```

La sentencia FOR implementa el bucle más flexible. A diferencia de los bucles vistos anteriormente, consta de tres expresiones: una de inicialización, una segunda de comprobación, y una tercera de incremento. El bucle funciona de la siguiente manera:

1. Se ejecuta la expresión de inicialización (izquierda)
2. Se ejecuta el bloque de sentencias
3. Se ejecuta la expresión de incremento (derecha)
4. Se evalúa la expresión de comprobación (centro). Si la expresión es **cierta**, se repite desde el paso 2. Si fuera falsa, se abandona el bucle inmediatamente.

Una sentencia BREAK abandonaría el bucle inmediatamente si llegara a ejecutarse en mitad del paso 2. Una sentencia CONTINUE saltaría el resto de sentencias que quedaran por ejecutarse en el bloque y pasaría inmediatamente al paso 3.

El bucle FOR habitual utiliza una asignación (por ejemplo i = 0) como expresión de inicialización, un incremento o decremento (tipo i++) como expresión de incremento, y una comprobación (como i < 10) como expresión de comprobación.

FROM ... TO ... END

```
FROM variable = expresión TO expresión [STEP constante]
    sentencia ;
    ...
    sentencia ;
END
```

La sentencia FROM realiza un bucle empleando un contador. Requiere la utilización de una variable global, local o privada, cuyo nombre debe aparecer a la izquierda del signo =. A la derecha puede emplearse cualquier expresión de inicialización. El bucle tiene un incremento constante, que por defecto es 1, pero que puede especificarse indicando la cláusula STEP. El funcionamiento del bucle es el siguiente:

1. El valor a la derecha del = es asignado a la variable
2. El bloque de sentencias se ejecuta
3. El valor de incremento de la cláusula STEP se suma a la variable (o 1 si no existe cláusula STEP en el bucle)
4. Si la variable no sobrepasa el límite impuesto por la cláusula TO, se repite el paso 2

Una sentencia BREAK abandonaría el bucle inmediatamente si llegara a ejecutarse en mitad del paso 2. Una sentencia CONTINUE saltaría el resto de sentencias que quedaran por ejecutarse en el bloque y pasaría inmediatamente al paso 3.

RETURN

RETURN [*expresión*] ;

La sentencia RETURN abandona el proceso actual y opcionalmente devuelve un valor de retorno al proceso que lo llamó, el indicado después de la palabra RETURN. El proceso es eliminado de memoria inmediatamente, y sus variables locales y privadas, destruídas. Un proceso que vuelve con la función RETURN no será representado en pantalla en el próximo frame.

FRAME

FRAME [*expresión*] ;

La sentencia FRAME abandona la ejecución del proceso actual y vuelve al proceso que lo llamó, devolviendo como valor de retorno el identificador del proceso actual (todos los procesos que se ejecutan tienen un identificador único). Un proceso que utilice esta instrucción para retornar debe tener como tipo de dato de retorno el INT. Sin embargo el proceso no es destruído, sino que queda residente en memoria: el próximo frame será dibujado en pantalla, si tiene valores correctos en las variables locales necesarias.

Cuando todos los procesos acaban, bien porque se ejecuta la última instrucción, bien porque han vuelto con una instrucción RETURN, o bien porque han quedado en espera en una instrucción FRAME, Fenix se encarga de dibujar el siguiente frame y, una vez acabado, retoma la ejecución de los procesos que queden en memoria a partir de la próxima instrucción que siga a FRAME. Todos los procesos que deben permanecer en pantalla durante un tiempo deben incorporar la instrucción FRAME en el interior de un bucle de algún tipo.

El valor opcional que recibe la instrucción frame puede hacerse servir para hacer que un proceso vaya más o menos lento que el resto. Por defecto este valor es 100. Con un valor de 200, el proceso se ejecutará sólo una vez cada dos frames. Con un valor de 50, se ejecutará dos veces cada frame. Con otros valores menos exactos, el proceso se ejecutará un número variable de veces cada frame, pero siempre con la idea de que cada 100 ejecuciones de un proceso normal, éste se habrá ejecutado tantas veces como indique esta expresión (que puede o no ser constante).

Es posible emplear funciones del sistema como GET_ID para averiguar qué procesos están en activo en todo momento, y la instrucción SIGNAL para destruir o detener temporalmente el funcionamiento de un proceso (incluso a partir de otro proceso).

Asignación

variable = *expresión* ;

Probablemente la sentencia más habitual, después de la llamada a un proceso o función, sea la asignación, consistente en almacenar el resultado de evaluar una expresión en una variable local, global o privada.

Recordemos que cualquier expresión puede contener una asignación, así que es posible escribir asignaciones de un valor a múltiples variables anidando estas asignaciones, por ejemplo:

```
x = y = 0 ;              // Guarda 0 tanto en x como en y
```

De hecho es posible emplear, como asignación, cualquier expresión que modifique el valor de una variable. Las siguientes son sentencias de asignación perfectamente válidas:

```
x++ ;        // Incrementa el valor de X
++x ;        // Sentencia igual a la anterior
--x ;        // Decrementa el valor de x
x += 12 ;    // Suma 12 a X
x *= 16 ;    // Multiplica X por 16
```

etc.

Llamada

proceso ([*parámetros*]) ;
función ([*parámetros*]) ;

La sentencia probablemente más corriente es la llamada a un proceso o a una función del sistema. La mayoría de capacidades de Fenix vienen dadas por funciones del sistema que realizan tareas tales como recuperar gráficos de disco, manipularlos en memoria, hacer cambios a la paleta de colores y muchas otras actividades.

Tanto un proceso como una función pueden ejecutarse escribiendo su nombre seguido de, entre paréntesis, tantas expresiones como parámetros requiera la función o el proceso, separadas por comas. Algunas funciones y procesos no recibirán ningún parámetro, con lo que es perfectamente válida la forma nombre() para hacer la llamada.

En el caso de ejecutar un proceso, la ejecución del proceso actual continuará cuando el proceso llamado llegue a una instrucción FRAME o RETURN.

Es también posible hacer la llamada dentro de cualquier expresión, empleando el valor de retorno de una función o proceso para hacer algún cálculo o asignárselo a una variable. La sintaxis de estas llamadas son las mismas a la llamada normal como sentencia.

3 Variables globales

3.1 Entrada/salida

int mouse.x

int mouse.y

Estas variables contienen en todo momento las coordenadas actuales del ratón en pantalla. Las coordenadas (0,0) equivalen a la esquina superior izquierda. Sin embargo estas variables sólo son actualizadas durante la instrucción frame.

Incluso aunque el ratón no esté *visible* (para lo cual es necesario que tenga un gráfico asignado), el movimiento del ratón se sigue capturando igualmente y es posible leer estas variables.

Es posible cambiar cualquiera de las dos variables realizando una asignación, lo cual tendrá el efecto de desplazar inmediatamente el ratón a la nueva posición.

En muchos juegos el ratón no se implementa con un cursor, sino que se sigue el movimiento de éste para girar o desplazar elementos en pantalla. Para poder lograr este efecto una posibilidad sencilla consiste en mover el ratón al centro de pantalla al final de cada frame, y al comienzo del siguiente comprobar su desplazamiento para saber la cantidad de movimiento del ratón durante el frame anterior.

int mouse.file

int mouse.graph

Estas dos variables contienen, respectivamente, el identificador de librería y el identificador del gráfico dentro de dicha librería que debe dibujarse en la posición del ratón automáticamente cada frame. Por defecto, ambas variables contienen 0, lo cual tiene como resultado que no se dibuja cursor ninguno. Es preciso asignar un gráfico existente al menos a la variable mouse.graph para que el ratón sea visible.

int mouse.angle

int mouse.size

int mouse.flags

int mouse.region

int mouse.z

Estas variables controlan el ángulo, tamaño, flags, región de recorte y profundidad del cursor ratón. Actúan de la misma manera que las variables locales de un proceso, y permiten alterar el aspecto del cursor del ratón. Son modificables en cualquier momento, y los valores por defecto son 0 para angle, flags y region, 100 para size (lo cual indica un tamaño normal del 100%) y -512 para z, con lo que el cursor del ratón por norma general se visualizará por encima de cualquier otro proceso (recordemos que cuanto menor sea la variable z, más "cerca del usuario" se considera que está el proceso).

int mouse.left

int mouse.middle

int mouse.right

Estas tres variables globales contienen el estado de los tres botones del ratón (izquierdo, central y derecho respectivamente). Cada una de ellas contendrá 1 (cierto) si el botón correspondiente está pulsado, y 0 (falso) si no lo está. Al igual que las variables mouse.x y mouse.y, sólo se actualizan durante la instrucción frame. Modificarlas no tendrá ningún efecto.

int ascii

Contiene el valor ASCII de la tecla pulsada en el último frame. Si en el anterior frame no se pulsó ninguna tecla, esta variable contendrá 0.

Si en el último frame se pulsaron varias teclas en secuencia, el código ASCII de la primera de ellas será el contenido en esta variable. El resto de teclas no se pierden, sino que son almacenadas internamente y en los próximos frames esta variable va tomando una a una su valor. Esto significa que en rutinas de "input" que recojan la entrada por teclado del usuario, la velocidad máxima será de una tecla por frame (lo cual puede provocar un retraso perceptible si el juego es lento) pero, por otra parte, un proceso que examine cada frame el contenido de esta variable nunca "perderá" ninguna tecla.

int scan_code

Contiene el valor de tecla pulsada en el último frame. Este valor corresponderá a una de las constantes utilizables con la función key. A esta variable también se le aplica la misma norma que a la variable global ascii (ver más arriba), en cuanto a que nunca se perderá la pulsación de una tecla, pero es más rápido emplear la función key en juegos que deban responder rápidamente al teclado o admitan la pulsación de varias teclas a la vez.

3.2 Control de tiempo

int timer[9]

Existen 10 "temporizadores" o cronómetros contínuamente disponibles en Fenix y accesibles empleando las variables timer[0] a timer[9]. Al igual que ocurre con todos los arrays, es posible acceder a timer como a una variable entera, lo cual equivale efectivamente a acceder al primer temporizador, timer[0].

Estos temporizadores son variables enteras que son inicializadas a cero al inicio del programa pero se incrementan automáticamente cien veces por segundo. Pueden hacerse servir para manejar animaciones u otros valores.

En cualquier momento puede asignarse un 0 u otro valor a un temporizador. El temporizador seguirá aumentando a partir de ese valor. Esta técnica puede hacerse servir para obligar a uno de los cronómetros a oscilar siempre dentro de un rango. Para hacer esto, sin embargo, es preciso utilizar código como

```
if (timer[0] > 1000) timer[0] = 1000; end
```

Y nunca cometer el error de hacer esto:

```
if (timer[0] == 1000) timer[0] = 1000; end
```

Ya que los temporizadores, al igual que el resto de variables globales, permanecen detenidos a nivel del lenguaje hasta la próxima instrucción frame. A no ser que el juego funcione a más de 100 frames por segundo, las sucesivas lecturas de un temporizador no leerán valores consecutivos. Recuerda que los temporizadores cuentan tiempo real, no frames, y nunca es seguro cuánto puede tardar un frame en ejecutarse.

int fps

Indica la cantidad de frames por segundo a la que el juego se está ejecutando. Este valor no es el mismo que el seleccionado empleando la función set_fps, sino un cálculo realizado a partir del tiempo que tardó en dibujarse el frame anterior. El valor por lo tanto oscila cada frame, y puede utilizarse como indicativo de si el juego está funcionando a velocidad suficiente.

3.3 Línea de comandos

Cuando un programa se ejecuta, puede opcionalmente recibir parámetros: cada palabra escrita en la consola después del nombre del ejecutable, es un parámetro independiente. Fenix almacena los parámetros recibidos en las siguientes variables globales, de manera que estén accesibles para el propio programa.

int argc

Contiene el número de parámetros recibidos por el programa. El propio nombre del programa es un primer parámetro, así que esta variable contendrá siempre 1 o más.

string argv[32]

Contiene los parámetros recibidos por el programa. argv[0] equivale al nombre del propio programa. Aunque Fenix deja espacio suficiente para recibir hasta 32 parámetros, el número real de parámetros recibidos es el indicado por la variable argc.

3.4 Modo gráfico

int graph_mode

Esta variable sirve para controlar diversos parámetros de configuración de video. Su valor sólo puede alterarse al comienzo del programa, ya que una vez inicializado el sistema gráfico (lo cual ocurre automáticamente la primera vez que se ejecuta cualquier acción gráfica, incluso cargar a memoria algún fichero gráfico con LOAD_FPG o similar) no podrá cambiarse.

Actualmente sirve para poder activar el modo gráfico de 16 bits. Por defecto, el intérprete funcionará en 8 bits. Para hacer un juego que funcione en 16 bits, es preciso ejecutar al comienzo del programa:

```
GRAPH_MODE = MODE_16BITS;
```

Hay que tener en cuenta que un modo de 8 bits, no pueden dibujarse en pantalla ningún gráfico de 16 bits (como es obvio). En cambio, en modo de 16 bits sí pueden dibujarse gráficos de 8 bits, y manejar la paleta de colores de la misma manera. A grandes rasgos, el modo de 16 bits es bastante compatible con el modo de 8 bits.

Nota: La paleta de colores (y las funciones que la afectan, como FADE o ROLL_PALETTE) sólo afectan a los gráficos de 8 bits. Los gráficos de 16 bits que haya en pantalla en ese momento no se verán afectados por los cambios en la paleta de colores.

3.5 Scroll

Existe una estructura global, struct scroll[9], que contiene datos de los 10 posibles scrolls, numerados de 0 a 9. Para iniciar un scroll es preciso utilizar la función start_scroll. Sin embargo, una vez inicializado, es posible modificar o consultar cada frame los miembros de esta estructura para determinar la posición o el aspecto de cada una de las áreas de scroll. Los componentes de la estructura se acceden empleando la sintaxis scroll[*n*].variable, por ejemplo scroll[0].speed accede a la velocidad de la primera área de scroll. Los componentes de la estructura son:

int scroll[*n*].x0

int scroll[*n*].y0

Estas dos variables indican las coordenadas X e Y de la esquina superior izquierda del gráfico de "suelo" del scroll. En el caso de que el scroll se haya inicializado con la opción de repetir vertical u horizontalmente el gráfico del suelo, estos valores pueden ser mayores al gráfico en sí, pues indican el comienzo del área visible de la zona de scroll.

Estas variables pueden alterarse a mano. Sin embargo para que esta operación tenga efecto hay que tener cuidado de no introducir ningún identificador de proceso en la variable scroll[*n*].camera, ya que de lo contrario Fenix actualizaría automáticamente los contenidos de estas variables para "perseguir" al proceso indicado, ignorando cualquier cambio manual realizado.

int scroll[*n*].x1

int scroll[*n*].y1

Estas dos variables indican las coordenadas X e Y de la esquina superior izquierda del gráfico de "fondo" del scroll. En el caso de que el scroll se haya inicializado con la opción de repetir vertical

u horizontalmente el gráfico de fondo, estos valores pueden ser mayores al gráfico en sí. Por otra parte los valores son independientes de la zona "visible" del área de scroll, pues el gráfico de fondo se emplea como mera decoración del scroll.

Estas variables pueden actualizarse manualmente; sin embargo para ello es preciso que la estructura contenga 0 en la variable scroll[*n*].ratio, o de lo contrario Fenix actualizará a partir de las variables x0 y x1 (ver más arriba) automáticamente, ignorando cualquier cambio manual hecho sobre ellas.

int scroll[*n*].z

Indica la profundidad del scroll, relativa a todos los procesos que se encuentran fuera del scroll (es decir, con un ctype de 0). No hay ningún problema en cuanto a apilar áreas de scroll una tras otra (asignándoles a las áreas de scroll la misma región, o secciones que se solapen) o a utilizar procesos que se dibujen fuera del área de scroll. Esta variable permite considerar al área de scroll entera como un proceso más, y ajustar el orden en el que se dibuja. Por defecto la variable contiene el valor 512, lo cual la hará aparecer por detrás de la mayoría de procesos (que se crean con una z original de 0).

int scroll[*n*].camera

Por defecto contiene un 0. Cuando contiene el identificador de un proceso, Fenix modifica automáticamente cada frame las coordenadas x0 y y0 del scroll para que dicho proceso aparezca centrado en pantalla (o, si no es posible, que al menos aparezca visible).

Hay que tener cuidado en detener el scroll o bien en volver a asignar un 0 a esta variable si en cualquier momento el proceso en cuestión desapareciera.

int scroll[*n*].ratio

Por defecto contiene un 200, e indica en porcentaje la velocidad de movimiento automática del gráfico de fondo. Cuando esta variable contiene un valor diferente de 0, Fenix actualiza automáticamente cada turno las coordenadas x1 e y1 del scroll para que el fondo se desplace automáticamente en la misma dirección que el gráfico de suelo, pero con una velocidad diferente. Con el valor por defecto esta velocidad será un 200% de la del gráfico de suelo.

Para poder manejar manualmente la posición del gráfico de fondo (alterando a mano las variables x1 e y1) es preciso asignar un 0 a esta variable.

int scroll[*n*].flags1

Permite asignar "flags" de dibujo al gráfico de suelo, de manera que éste pueda dibujarse invertido horizontal o verticalmente, o con transparencias. El valor de la variable puede ser la suma de cualquiera de los valores de la siguiente tabla:

1	Espejo (inversión) horizontal
2	Espejo (inversión) vertical
4	Transparencia
128	Trata el color 0 como uno más

int scroll[*n*].flags2

Variable equivalente a flags1 (ver más arriba), pero afecta al gráfico de fondo en lugar de al de suelo.

3.6 Sonido

int sound_mode

Esta variable especifica diversos parámetros de configuración para el sonido. Su valor sólo puede alterarse al comienzo del programa, ya que una vez inicializado el sistema de sonido (lo cual ocurre automáticamente la primera vez que se llama a load_mod, load_pcm, etc) no podrá cambiarse. Su valor puede ser una mezcla de las siguientes constantes predefinidas:

MODE_STEREO	Sonido en stereo
MODE_16BITS	Calidad de sonido de 16 bits en lugar de 8
MODE_HQ	Mezclador de alta calidad
MODE_SURROUND	Sonido surround
MODE_INTERPOLATION	Interpolación de sonido
MODE_REVERSE	Intercambia los altavoces izquierdo y derecho

Se aconseja utilizar el operador "|" (OR) para unir las constantes deseadas y asignar así un valor a sound_mode. Cada una de estas constantes indica un "bit" y puede ser combinada con el resto. Por ejemplo, para activar el SURROUND, el modo stereo y la calidad de sonido de 16 bits haríamos, al comienzo del programa:

```
sound_mode = MODE_STEREO | MODE_16BITS | MODE_SURROUND;
```

Fenix utiliza la librería MikMod para reproducir cualquier sonido. Esta librería incluye una serie de mezcladores por software que están dedicados a reproducir ficheros MOD, S3M, IT, etc, con una gran calidad. Sin embargo, activar todas las opciones de calidad (16 bits, mezclador de alta calidad, interpolación, etc) supone una pesada carga para la CPU, incluso en un Pentium II o Pentium III. Por ello debes elegir con cuidado las opciones de sonido:

MODE_STEREO especifica sonido en estereo. Es la única opción activada por defecto. El sonido stereo es el doble de lento de mezclar que el mono, así que si no lo necesitas en tu juego puedes preferir desactivarlo.

MODE_16BITS selecciona la calidad de sonido de 16 bits. Con ello se obtiene una notable mejora en la calidad de sonido (especialmente con sonidos PCM de poca calidad), aunque es el doble de lento de mezclar que el modo por defecto, de 8 bits.

MODE_HQ activa el mezclador de calidad. Este mezclador es mucho más lento que el normal (incluso cuatro veces o más), aunque su calidad es magnífica. Usarlo puede significar aumentar considerablemente los requisitos de tu programa.

MODE_SURROUND activa el modo surround. Con este modo activado, los efectos de sonido (función SOUND) suenan en "surround". La mejor forma de describirlo es que lo pruebes con algún juego existente.

MODE_INTERPOLATION activa la interpolación en el mezclador. Esto mejora la calidad especialmente a baja frecuencia, a costa de un uso más intensivo de la CPU.

MODE_REVERSE simplemente intercambia los canales izquierdo y derecho del stereo. No tiene efecto alguno sobre el rendimiento.

Por defecto el valor de la variable es 0, con lo que ninguna de las opciones está activada.

int volume

Especifica el volumen global del sonido. No debe confundirse con el volumen del módulo de sonido (véase mod_get o mod_get), ni con el volumen de cada sonido (SOUND). Este es un volumen global, que afectará a todos los sonidos, y puede emplearse para obtener un efecto de "fade" con el sonido, ya que cualquier cambio en esta variable afectará al sonido inmediatamente. Su rango puede oscilar entre 0 (silencio) y 127 (máximo volumen).

int sound_freq

Esta variable especifica la calidad de la reproducción del sonido. Valores típicos pueden ser 44100 (calidad de CD), 22050 (calidad de radio) o 11025 (calidad horripilante).

Si tu juego emplea sonido MOD de muchos canales, a la vez de múltiples efectos de sonido, bajar este valor acelerará el rendimiento del juego.

Este valor sólo puede modificarse antes de ejecutar cualquier instrucción de sonido (ni siquiera load_mod), típicamente al comienzo de programa. Una vez se inicializa el sonido, ya no es posible cambiar la frecuencia de salida, y un cambio en el valor de esta variable no hará efecto.

int reverb

Esta variable permite añadir una "reverberación" al sonido. Su valor puede situarse entre 0 (sin reverberación) hasta 15 (máxima). Puede cambiarse en cualquier momento, y el sonido se adaptará inmediatamente.

La reverberación puede resultar inapropiada si es excesiva (el sonido parece como reproducido "dentro de una cueva"), pero un valor de 4 ó 5 puede mejorar notablemente el sonido de ficheros MOD de poca calidad.

Es mucho más lento mezclar sonido con reverberación que sin ella. Deja la variable a 0 para aumentar el rendimiento de tu programa.

int pansep

Esta variable especifica la separación entre los canales izquierdo y derecho, cuando el sonido se encuentra en modo stereo. Puede alterarse en cualquier momento, y el sonido se adaptará a su valor inmediatamente.

El valor por defecto de 64 es un punto medio. La variable puede adquirir cualquier valor entre 0 (que elimina el efecto stereo) y 127 (separación completa entre canales).

La variable es útil especialmente al reproducir ficheros MOD de pocos canales antiguos, donde cada canal suena sólo por uno de los altavoces, lo cual produce una molestia apreciable al escuchar el sonido mediante cascos, por ejemplo.

4 Variables locales

Las variables locales son variables comunes a todos los procesos, lo cual significa que todos los procesos tienen las mismas variables locales. Por supuesto las variables locales de un proceso son independientes de las de los demás: aunque todos los procesos tengan una variable local x, cada proceso tiene su propia x, que puede modificar o consultar independientemente de las variables x de los demás procesos en ejecución.

Muchas variables locales están ya predefinidas: todos los procesos creados en Fenix disponen de estas variables, que en muchos casos son empleadas por el intérprete para definir el aspecto del proceso en pantalla. Basta con alterar ciertas variables locales como graph o file para que el proceso aparezca en pantalla en la próxima ejecución de la sentencia FRAME.

4.1 Jerarquía de procesos

int father

Esta variable contiene el identificador del proceso padre, es decir, el proceso que ejecutó al actual. Al igual que con cualquier variable entera que contenga un identificador de proceso, es posible acceder a las variables de éste utilizando construcciones como father.graph.

Es posible que el proceso padre muera mientras el hijo sigue vivo. En ese caso esta variable pasará a contener 0.

int son

Esta variable contiene el identificador del último proceso creado por el actual. Contendrá 0 si el proceso actual no ha ejecutado todavía ninguno.

int smallbro

Esta variable contiene el identificador del siguiente proceso creado por el proceso padre después del actual. Contendrá 0 si el proceso padre no ha creado ningún proceso más todavía.

int bigbro

Esta variable contiene el identificador del último proceso creado por el proceso padre, antes del actual, o 0 si el proceso actual es el primero que ha creado el padre.

4.2 Control de ejecución

int priority

Esta variable contiene la *prioridad* del proceso actual. Los procesos que tienen una prioridad mayor se ejecutan antes, ya que a la hora de dibujar cada frame del juego, el intérprete emplea el valor contenido por esta variable global para establecer un orden adecuado de ejecución.

Hay que tener en cuenta que los procesos se ejecutan primero y sólo cuando ya no queda ningún proceso por ejecutar en el frame actual, se dibujan en pantalla.

Si lo que buscas es establecer qué procesos se verán por delante de otros, emplea la variable local z. Esta otra variable permite realizar cosas muy diferentes, por ejemplo, que una serie de procesos se "persigan" entre ellos, haciendo que se ejecute en primer lugar aquél cuyo movimiento es independiente y después, por orden, los procesos que han de seguirse unos a otros, comenzando por el proceso que perseguirá a éste.

4.3 Aspecto gráfico

int ctype

Permite especificar qué tipo de coordenadas emplea el proceso y, por lo tanto, dónde debe dibujarse. La variable puede contener como valor una de las siguientes constantes predefinidas:

C_SCREEN	Coordenadas referentes a pantalla
C_SCROLL	Coordenadas referentes a una zona de scroll
C_M7	Coordenadas referentes a una zona de modo 7

El valor por defecto es C_SCREEN. En el caso en que la variable contenga otro valor, es posible emplear la variable local cflags para especificar qué ventanas concretas de scroll o modo 7 son aquellas en las que se visualizará el proceso.

int cflags

Esta variable indica en qué ventanas de scroll o modo 7 el proceso va a visualizarse. Su valor puede calcularse sumando una o más filas de la tabla siguiente tabla:

1	Ventana 1
2	Ventana 2
4	Ventana 3
8	Ventana 4
16	Ventana 5
32	Ventana 6
64	Ventana 7
128	Ventana 8
256	Ventana 9
512	Ventana 10

El valor por defecto implica que el proceso se visualizará en todas las ventanas activas (una ventana de scroll o modo 7 debe estar activa para que en ella se visualice un proceso, independientemente del valor de esta variable).

Para que esta variable tenga sentido, es preciso inicializar también la variable local ctype.

int x

int y

Estas dos variables locales contienen la posición del gráfico del proceso en pantalla (o quizá en una zona de scroll; véase la variable ctype). Sólo cambiando los valores contenidos por las variables, el gráfico indicado por la variable local graph del proceso se posicionará en la nueva zona en el frame siguiente.

Por defecto su valor está en pixels, de manera que la coordenada (0, 0) corresponde a la esquina superior izquierda de pantalla y ambos valores crecen en positivo a medida que se desplazan a derecha y abajo respectivamente. Es posible usar coordenadas más pequeñas que un pixel empleando la variable local resolution. En ese caso, la posición auténtica será redondeada al pixel más cercano.

La posición indicada por estas variables será la que ocupe el *centro* del gráfico. Normalmente el centro del gráfico equivale al centro geométrico delimitado por su ancho y alto, pero un gráfico puede contener un punto de control 0 que especifique un nuevo centro.

int resolution

Esta variable permite aumentar la resolución de las coordenadas x e y de un proceso, ya que indica cuántas unidades existen en un pixel en pantalla. Por ejemplo, con un valor de 100 en resolution, avanzar 400 veces la coordenada x implica un avance de sólo 4 pixels, ya que 100 unidades corresponden a un sólo pixel.

Es importante emplear esta variable en casos en que un objeto debe moverse lentamente en pantalla, ya que gracias a ella es posible sumar valores fácilmente a sus coordenadas de manera que el objeto avance tan sólo un pixel cada varios frames. También permite dotar de más precisión y suavidad a objetos que son afectados por la gravedad o diversas formas de aceleración y fricción.

Por defecto esta variable contiene 0 y el intérprete no realiza ningún cálculo, asumiendo que los valores en x e y corresponden directamente a pixels.

int angle

Esta variable especifica el ángulo de visualización del gráfico del proceso, en milésimas de grado.

Normalmente este ángulo es utilizado por el intérprete para rotar el gráfico del proceso en tiempo real. Un valor de 90000 girará el gráfico 90º a la izquierda, mientras un valor de 0 deja el gráfico original sin modificar.

Opcionalmente, se puede emplear la variable local xgraph para que este valor sirva simplemente para escoger un gráfico entre una tabla, que se dibujará sin rotar.

int size

Esta variable especifica el temaño del gráfico. El intérprete escalará el gráfico en función de este valor, que viene dado en porcentaje: un 100 (que es el valor por defecto) dibuja el gráfico al tamaño original, mientras un 50 dibujará el gráfico a mitad de ancho y alto.

int flags

Este valor especifica activa opciones adicionales a la hora de dibujar el gráfico del proceso, por parte del intérprete. Puede contener la suma de uno o más de los valores de la tabla siguiente:

1	Espejo (inversión) horizontal
2	Espejo (inversión) vertical
4	Transparencia
128	Trata el color 0 como uno más

El espejo es distinto a la rotación, y también más rápido de dibujar. Es preferible utilizar el flag de espejo en lugar de una rotación cuando sea posible: los procesos que utilizan un ángulo de 0 y un tamaño estandar, se dibujan empleando una rutina alternativa más rápida, independientemente del valor contenido en esta variable.

Los gráficos que no contienen zonas transparentes, o que se dibujan sobre fondo negro, pueden dibujarse empleando el flag 128, lo cual agiliza la operación de dibujo especialmente en gráficos de gran tamaño, como un fondo.

int z

Esta variable especifica el orden en que los procesos se dibujan. Los procesos con un valor superior en esta variable se dibujan en primer lugar, de manera que aparecerán siempre por debajo del resto.

Si dos procesos tienen el mismo valor en esta variable, quedará indeterminado el orden en el que se dibujarán en cada frame.

int region

Permite especificar una región en pantalla, que debe haberse creado primero empleando la función define_region.

Si un proceso que no se dibuja en ventana de scroll ni de modo 7 es asignado a una región, entonces su gráfico será recortado por las rutinas de dibujo de tal manera que no pueda sobreescribir nada fuera de la región de pantalla asignada.

pointer xgraph

A esta variable es posible asignar el offset de una tabla de variables tipo int, de manera que el ángulo especificado por la variable angle no altera el gráfico del proceso, sino que sirve para escoger uno de los gráficos de dicha tabla.

La tabla deberá constar de una serie de enteros de los cuales el último debe ser 0. El intérprete dividirá los 360 grados entre los gráficos disponibles (de manera que el primer gráfico corresponde al ángulo 0 y siguientes) y asignará un gráfico a cada frame en función del valor que la variable angle contenga en ese momento.

Por defecto esta variable contiene 0 (el 0 es el único valor válido aparte de un offset que puede asignarse a un puntero) y la variable angle cumple su cometido normal.

int file

Esta variable indica el número de fichero FPG donde se encuentra el gráfico de un proceso. Todos los gráficos en Fenix se identifican con dos números enteros: uno para identificar el fichero que lo contiene, y otro con el código del gráfico dentro del fichero.

El primer fichero FPG en abrirse con la función LOAD_FPG siempre recibirá el código 0, que es también el valor por defecto de esta variable. Si tu juego sólo utiliza un único fichero FPG, o bien sólo usa gráficos cargados con LOAD_MAP o similares (que también utilizan un código de fichero 0), puedes despreocuparte de la existencia de esta variable.

Por otra parte asignando valores a esta variable puedes conseguir efectos tales como objetos con distintas animaciones que pueden utilizarse indistintamente con el mismo proceso, al estar cada uno en un fichero FPG cuyos códigos de gráfico corresponden entre ellos a la misma posición o animación predefinida.

int graph

Esta variable indica el código de gráfico del proceso. Si esta variable no contiene un código de gráfico válido, ninguna de las variables locales que modifican el aspecto o posición del proceso tienen utilidad. Es la variable local principal que todos los procesos que cumplen la función de un objeto en pantalla deben inicializar, bien mediante un código predefinido que tiene un objetivo concreto dentro de las librerías FPG de tu juego, bien mediante el código devuelto por funciones como LOAD_MAP o NEW_MAP.

5 Funciones del sistema

5.1 Inicialización y salida

set_mode (int *modo*)

Parámetros:

modo Indica la resolución de pantalla. Puede ser una de las siguientes constantes:

M320x200
M320x240
M320x400
M360x240
M376x282
M640x400
M640x480
M800x600
M1024x768

Funcionamiento: Esta función establece un modo gráfico, cuya resolución viene indicada por la constante que recibe como parámetro la función (*ancho* x *alto*). Se desaconseja la utilización de los modos de M320x400, M360x240 y M376x282, ya que pueden no estar disponibles en función de la tarjeta gráfica o el sistema operativo.

En el caso de que un modo gráfico no esté soportado, Fenix tratará de emularlo estableciendo el modo gráfico siguiente de mayor tamaño, lo cual no tiene ningún efecto por norma general, salvo que provoca que aparezca un borde alrededor de la pantalla en algunas configuraciones. Si no hubiera ningún modo gráfico disponible, el programa abortará con un error.

Nota: Esta función establece por defecto el modo gráfico de 8 bits indicado. Si se desea pasar a un modo de 16 bits, antes de llamar a esta función hay que actualizar acordemente la variable global GRAPH_MODE.

set_fps (int *fps*, int *salto*)

Parámetros:

fps — Indica el número de frames por segundo que se considera ideal para el juego.

salto — Indica el número máximo de frames seguidos que Fenix puede saltarse (no dibujar) para lograr que el número de fps se acerque más al ideal en ordenadores lentos.

Funcionamiento: Esta función establece la velocidad del juego, es decir, el número de frames que se deben completar y visualizar por segundo. Si no se llama a esta función, el número de frames por segundo por defecto es de 25.

Internamente, esta funcion establece un mínimo de tiempo para cada frame. Por ejemplo, con 25 fps el límite es de $1000/25 = 40$ ms. Si un frame tarda en dibujarse (lo que incluye ejecutar todos los procesos activos hasta que mueran o lleguen a la orden frame) menos de 40 ms, Fenix hará una pausa hasta que pase ese tiempo antes de procesar el siguiente.

Se aconseja elegir unos frames por segundo tales que el tiempo de ejecución de un frame sea múltiplo de 10. Eso incluye cantidades como 25, 40 ó 50 fps. Normalmente 30 fps son suficientes para obtener una suavidad de movimiento aceptable.

El segundo parámetro de set_fps permite a Fenix "saltarse" la visualización de un frame, si el anterior tarda demasiado en dibujarse. Este salto incluye solo la parte de dibujo en pantalla, mientras los procesos siguen ejecutándose. Un valor de 1 permite a Fenix saltarse un máximo de un frame consecutivo. El valor por defecto es de 0, por lo que siempre se visualizarán todos los frames, y en un ordenador lento la acción se enlentecerá. Con un valor de 1 o más, en dicho ordenador lento la acción se volverá más "brusca", a costa quizá de jugabilidad, pero probablemente la velocidad de ejecución será la misma.

La acción especial set_fps(0,0) elimina cualquier pausa o intento de saltar frames y hace que el juego vaya a la máxima velocidad posible, sin saltarse nada. Es útil a propósitos de *benchmark* para saber cuál es la máxima velocidad a la que podría ir un juego en nuestra plataforma.

exit (string *texto*, int *valor*)

Parámetros:

texto — Indica un texto a visualizar en pantalla al abandonar Fenix.

valor — Indica el *código de retorno* de la aplicación

Funcionamiento: Abandona la ejecución del programa inmediatamente. El parámetro "texto" identifica un texto que podría visualizarse en pantalla, mientras el "valor" es un valor de retorno. Actualmente estos parámetros son ignorados.

5.2 Números aleatorios

int rand (int *mínimo*, int *máximo*)

Parámetros:

mínimo — Mínimo número aleatorio a obtener

máximo — Máximo número aleatorio a obtener

Funcionamiento: Devuelve un número pseudo-aleatorio comprendido en el rango entre los dos números indicados, inclusive. Por ejemplo, rand(1,100) devolverá un número aleatorio entre 1 y 100 inclusive.

Puede utilizarse rand_seed para seleccionar una semilla determinada, a partir de la cual se generan los números aleatorios. Al comienzo del programa se selecciona una semilla aleatoria (en función de la hora del día).

rand_seed (int *número*)

Parámetros:

número Valor de "semilla"

Funcionamiento: Establece la "semilla" a partir de la cual se generan los números pseudoaleatorios que devuelve la función rand. Puede establecerse una semilla concreta para que los números aleatorios sean siempre los mismos, a partir de la llamada a la función rand_seed. O puede emplearse rand_seed(time()) para incrementar la aleatoriedad de los mismos (véase la función time).

5.3 Distancia y colisiones

int get_distx (int *angulo*, int *distancia*)

Parámetros:

angulo Indica un ángulo en milésimas de grado (1000 = 1º)

distancia Indica una magnitud de distancia

Funcionamiento: Esta función devuelve el ancho del rectángulo cuya diagonal es una línea orientada en el ángulo especificado, y de la longitud dada por el parámetro de distancia. Dicha cantidad, junto con la devuelta con get_disty, tiene bastantes utilidades en juegos donde el movimiento de los procesos se puede determinar en cualquier ángulo, pues permite desplazar un proceso a partir de un ángulo, una distancia determinada.

Este valor se calcula multiplicando el coseno del ángulo, por la distancia.

int get_disty (int *angulo*, int *distancia*)

Parámetros:

angulo Indica un ángulo en milésimas de grado (1000 = 1º)

distancia Indica una magnitud de distancia

Funcionamiento: Esta función devuelve el alto del rectángulo cuya diagonal es una línea orientada en el ángulo especificado, y de la longitud dada por el parámetro de distancia. Dicha cantidad, junto con la devuelta con get_distx, tiene bastantes utilidades en juegos donde el movimiento de los procesos se puede determinar en cualquier ángulo, pues permite desplazar un proceso a partir de un ángulo, una distancia determinada.

Este valor se calcula multiplicando el seno del ángulo, por la distancia.

int fget_angle (int *x1*, int *y1*, int *x2*, int *y2*)

Parámetros:

x1 Coordenada X del primer punto

y1 Coordenada Y del primer punto

x2 Coordenada X del segundo punto

y2 Coordenada Y del segundo punto

Funcionamiento: Esta función devuelve, en milésimas de grado, el ángulo formado por la línea entre los dos puntos especificados, y el eje X (horizontal) de coordenadas.

Con esta función puede, fácilmente, orientar el ángulo de un proceso para que éste siempre esté "mirando" hacia un lugar concreto (como puede ser otro proceso).

int fget_dist (int *x1*, int *y1*, int *x2*, int *y2*)

Parámetros:

x1	Coordenada X del primer punto
y1	Coordenada Y del primer punto
x2	Coordenada X del segundo punto
y2	Coordenada Y del segundo punto

Funcionamiento: Esta función devuelve la distancia entre dos puntos dados. Esta es una función relativamente lenta, que devuelve un resultado exacto (aunque redondeado a un número entero de pixels) empleando una raiz cuadrada.

int near_angle (int *angulo1*, int *angulo2*, int *max_inc*)

Parámetros:

angulo1	Ángulo original
angulo2	Ángulo de destino
max_inc	Número máximo de milésimas de grado a incrementar el ángulo original

Funcionamiento: Esta función aproxima el primer ángulo al segundo un número máximo de milésimas de grado, y devuelve el nuevo ángulo. Este nuevo ángulo nunca sobrepasará el ángulo de destino: si la distancia entre los dos ángulos es menor al parámetro de incremento máximo, entonces el ángulo resultante será exactamente el de destino. Hay que tener en cuenta que el valor de max_inc deberá ser positivo; sin embargo, se le sumará o restará al ángulo original según el lado de la circunferencia donde la distancia entre los dos ángulos sea menor.

Esta función sirve de ayuda para hacer procesos que persiguen a otros pero disponen de una capacidad de giro limitada cada frame.

int get_angle (int *id_proceso*)

Parámetros:

id_proceso	Identificador de un proceso distinto al actual

Funcionamiento: Esta función devuelve el ángulo formado por la línea entre el proceso actual y otro proceso, cuyo identificador se le pasa como parámetro a get_angle. El resultado estará en milésimas de grado.

La utilización típica de esta función consiste en hacer que un proceso "apunte" o mire hacia otro, asignando a la variable local angle el valor de llamar a get_angle con el parámetro del proceso destino.

La función se limita simplemente a calcular el ángulo entre las coordenadas locales (x, y) de cada uno de los dos procesos. No se obtendrá un resultado apropiado si procesos están en un área distinta de scroll, por ejemplo, o si utilizan una resolución diferente.

int get_dist (int *id_proceso*)

Parámetros:

id_proceso Identificador de un proceso distinto al actual

Funcionamiento: Devuelve la distancia entre el proceso actual y otro proceso. No es necesario que dicho proceso esté despierto o visible: la función se limita a realizar los cálculos necesarios a partir de las variables locales x, y, y resolution.

La distancia resultante se devuelve según la resolución del proceso actual. Sin embargo, por simplicidad, la distancia será un número exacto de pixels.

int collision (int *tipo*)

Parámetros:

tipo Un tipo de proceso (obtenido mediante la instrucción type) o bien el entero 0.

Funcionamiento: Esta es la función principal de colisión: busca un proceso del tipo dado que, según sus variables actuales y las variables del proceso actual, colisionaría en el caso de ser dibujado actualmente. Para saber el código de un tipo de proceso, es preciso emplear la instrucción type.

La comprobación es perfecta a nivel de pixel, teniendo en cuenta tamaño, rotaciones, y funciona incluso aunque la parte que colisiona de los objetos no aparezca finalmente en pantalla. Esto significa, por ejemplo, que un objeto con forma de donut no colisiona con un objeto más pequeño en su interior si éste no toca algún pixel no transparente.

Si se pasa como parámetro type mouse, la función devuelve 1 si el proceso actual colisiona con el gráfico del ratón según los valores de la estructura mouse. En el resto de casos, la función devuelve el identificador del proceso que ha colisionado, o 0 si no se encontró ningún proceso.

Si el parámetro de tipo es 0, se busca colisión con cualquier proceso de cualquier tipo.

En todo caso, para que una colisión exista, el gráfico del proceso debe estar visible, lo cual generalmente implica que el estado del proceso debe estar despierto o congelado (ver la función signal) y con un gráfico asignado.

Las sucesivas llamadas a la función collision con los mismos parámetros devuelven los siguientes procesos que estén en colisión con el actual, hasta que ya no quede ninguno. La instrucción frame resetea el funcionamiento de todas las búsquedas por colisión. Este funcionamiento permite utilizar la función collision dentro de un bucle, para obtener así todos los procesos que están en colisión con un gráfico, en un momento dado.

5.4 Matemáticas y trigonometría

float abs (float *n*)

Parámetros:

n Un número del que se desea conocer el valor absoluto

Funcionamiento: Esta función devuelve el valor absoluto de un número dado. El valor absoluto de un número negativo es el número positivo de valor equivalente, mientras que los números positivos o el cero se dejan tal cual. Esencialmente, abs(-10) es igual a abs(10), que es igual a 10.

float pow (float *n*, float *pot*)

Parámetros:

n Un número que se desea elevar a una potencia dada

pot Potencia a la que elevar el número anterior

Funcionamiento: Esta función eleva un número a una potencia. pow(2,3) devuelve 2^3, por ejemplo.

float sqrt (float *n*)

Parámetros:

n Un número del que se desea saber la raíz cuadrada

Funcionamiento: Esta función devuelve la raíz cuadrada del número que recibe como parámetro. sqrt(49) devuelve $\sqrt{49}$, es decir, 7.

float cos (float *n*)

float sin (float *n*)

float tan (float *n*)

Parámetros:

n Un ángulo dado en milésimas de grado

Funcionamiento: Las funciones trigonométricas básicas devuelven una magnitud a partir del ángulo dado. Estas funciones no usan tablas precalculadas, por lo que son relativamente lentas, y admiten un parámetro con decimales para una mayor precisión.

float acos (float *n*)

float asin (float *n)*

float atan (float *n)*

Parámetros:

n Un número en coma flotante

Funcionamiento: Estas funciones devuelven un ángulo (en milésimas de grado) cuya operación trigonométrica da como resultado el valor que recibe como parámetro. Así, acos(0.5) devuelve el ángulo α tal que $\cos(\alpha) = 0.5$.

5.5 Gestión de procesos

signal (int *proceso*, int *señal*)

Parámetros:

proceso Identificador de un proceso

señal Tipo de señal a enviar al proceso; puede ser una de las siguientes constantes:

S_KILL
S_WAKEUP
S_SLEEP
S_FREEZE
S_KILL_TREE
S_WAKEUP_TREE
S_SLEEP_TREE
S_FREEZE_TREE

Funcionamiento: Esta función modifica el estado de un proceso o, con una de las constantes acabadas en _tree, modifica el estado tanto del proceso cuyo identificador recibe, como el de todos sus hijos y los hijos de éstos.

Existen cuatro posibles estados para un proceso:

WAKEUP El proceso se ejecuta normalmente. Su gráfico es visible, en caso de existir uno, y se desplaza según las modificaciones realizadas por el código del proceso a sus variables locales.

FREEZE El proceso está detenido temporalmente. El código no se ejecuta, sin embargo su gráfico permanece visible en pantalla según los valores contenidos por sus variables locales en el momento de ejecutar la función signal.

SLEEP El proceso está detenido temporalmente. Su gráfico no es visible ni se dibuja cada frame.

KILL El proceso está muerto. Su código no se ejecuta, ni es visible su gráfico, y será eliminado de memoria al finalizar el actual frame. A partir de ese momento su identificador dejará de ser válido.

La función signal puede cambiar el estado de un proceso en cualquier momento. Los estados kill deben considerarse especialmente, ya que ningún acceso a un proceso realizado después de utilizar un signal de tipo kill se considera válido (ni siquiera una segunda instrucción signal).

int get_id (int *tipo*)

Parámetros:

tipo Tipo de proceso, obtenido mediante la instrucción type, o bien el entero 0

Funcionamiento: Esta es una función que sirve para recorrer la lista de procesos activos. Cuando se le llama por primera vez, devuelve el identificador del primer proceso que encuentra. Cuando se le llama sucesivas veces, get_id devuelve uno a uno los identificadores de todos los procesos existentes. Cuando ya no queda ningún proceso por listar, get_id devolverá 0.

Opcionalmente get_id puede devolver sólo los identificadores de procesos de un tipo determinado. Para ello basta con pasarle como parámetro el identificador de tipo obtenido mediante la instrucción type.

get_id se resetea a cada frame. El primer get_id a ejecutarse después de cada frame, devuelve siempre el primer código de proceso.

El estado de get_id es local a cada proceso. Es decir, un proceso puede recorrer la lista de procesos activos mediante get_id y detenerse en cualquier momento: ello no afectará a un get_id llamado posteriormente en otro proceso, aún dentro del mismo frame.

let_me_alone ()

Funcionamiento: Esta función mata inmediatamente todos los procesos existentes, excepto el actual. Es muy útil cuando hay que abandonar el juego, lo cual normalmente implica matar gran cantidad de procesos.

int exists (int *proceso*)

Parámetros:

tipo Tipo de proceso, obtenido mediante la instrucción type, o bien el entero 0

Funcionamiento: Esta función comprueba si un proceso existe y devuelve 1 (CIERTO) si así es, y 0 (FALSO) si el identificador que recibe como parámetro no corresponde a ningún proceso.

Es un error grave tratar de acceder a un dato local de un proceso que ha muerto y dejado de existir. Esta función es inmediata, y mucho más rápida que un bucle get_id: se aconseja emplearla, incluso a cada frame, en aquellos procesos que dependen de los datos de otros procesos que pueden morir en cualquier momento.

El ejemplo típico es un juego de naves en los que los procesos enemigos se dirigen hacia la nave del jugador. Si el proceso de la nave del jugador fuera destruído, estas naves enemigas no pueden obtener su última posición con una instrucción del tipo nave.x, ya que el proceso ha sido eliminado de memoria. Será conveniente que estas naves enemigas comprueben primero con un if exists(nave) que es válido hacer esos cálculos.

5.6 Entrada/salida

int key (int *tecla*)

Parámetros:

tecla Código de una tecla

Funcionamiento: Esta función devuelve 1 si la tecla cuyo código recibe como parámetro está pulsada en ese mismo momento.

Los códigos de tecla pueden obtenerse gracias a una serie de constantes predefinidas que empiezan todas por el signo "_". Cualquiera de las siguientes constantes puede emplearse como parámetro para key().

int get_joy_button (int *boton*)

Parámetros:

boton Número de botón del joystick. El botón principal es el 0.

Funcionamiento: Esta función devuelve el estado de uno de los botones del joystick: 0 si el botón no está pulsado, o un número mayor de 0 en caso contrario.

Todos los joysticks se supone que tienen al menos un botón, de número 0. El resto de botones tendrán números consecutivos (1, 2, 3...).

int get_joy_position (int *eje*)

Parámetros:

eje Número de eje. Un valor de 0 identifica el eje X. Un 1 identifica el eje Y.

Funcionamiento: Esta función devuelve la posición (en formato entero) de uno de los ejes del joystick. Actualmente sólo se soportan joysticks con dos ejes, el 0 (eje X) y el 1 (eje Y).

int select_joy (int *joystick*)

Parámetros:

joystick Número de joystick, empezando por 0, o bien -1 para no cambiar el joystick activo.

Funcionamiento: Selecciona un joystick. Las llamadas realizadas a las funciones get_joy_button y get_joy_position a partir de ese momento se referirán al joystick seleccionado por esta función.

Puede haber hasta 8 joysticks diferentes disponibles, numerados del 0 al 7. Por defecto se encuentra seleccionado el joystick número 0.

La función devuelve el número total de joysticks disponibles, y no hace nada más si el número de joystick indicado no se encuentra dentro del rango permitido. La forma recomendada de saber el número de joysticks instalados es llamar a esta función con el parámetro -1:

```
num_joysticks = select_joy(-1);
```

5.7 Gestión de gráficos y librerías

int load_map (string *fichero*)

int load_pcx (string *fichero*)

int load_png (string *fichero*)

Parámetros:

fichero Nombre de un fichero gráfico, incluyendo extensión.

Funcionamiento: Estas funciones cargan un fichero gráfico del disco y devuelven el identificador del nuevo gráfico. Dicho gráfico puede utilizarse a partir de entonces con todas las funciones que requieren un identificador de gráfico, especificando como código de librería el 0. También puede asignarse este identificador a la variable local graph de un proceso, por ejemplo.

Es preciso llamar a la función adecuada en función del formato gráfico del fichero que desea cargarse. A parte del requisito, no hay en realidad ninguna otra diferencia en su funcionamiento.

En el caso de que haya una librería 0 cargada, los gráficos cargados con una de estas funciones no se borrarán de memoria al descargarse dicha librería. Será preciso utilizar de forma independiente la instrucción unload_map con cada uno de ellos.

Nota: Los gráficos de 16 bits sólo pueden visualizarse en pantalla si el intérprete soporta 16 bits de color. Los tres formatos gráficos admiten gráficos de 16 bits. Los gráficos PNG de más profundidad de color serán reducidos automáticamente a 16 bits.

Notasans graphic_set.

Notasans int load_fpg (string *fichero*)

Parámetros:

fichero Nombre del fichero FPG a cargar

Funcionamiento: Esta función recupera una librería de gráficos FPG, y devuelve el código de la nueva librería. Dicho código debe preservarse, ya que es preciso utilizarlo en todas las llamadas a funciones que alteran o trabajan con un gráfico en memoria. También es preciso asignarle este identificador de librería a la variable FILE de todos los procesos cuyo gráfico pertenezca a la librería.

Una librería FPG puede contener hasta 1000 gráficos diferentes. El identificador de cada gráfico de una librería es independiente de los identificadores de gráficos de otras librerías.

La primera librería en cargarse siempre tendrá el código 0. Para juegos pequeños o demos que sólo utilicen una librería, es válido cargar ésta al comienzo del programa usando esta función y utilizar el código 0 directamente en posteriores llamadas a funciones (tampoco deben preocuparse por la variable FILE, que inicialmente contiene un 0 por defecto).

int graphic_info (int *librería*, int *gráfico*, int *tipo*)

Parámetros:

librería Código de librería a la que pertenece el gráfico

gráfico Identificador del gráfico dentro de la librería

tipo Especifica qué parámetro del gráfico se desea obtener. Puede ser una de las siguientes constantes predefinidas:

G_WIDE	Ancho
G_HEIGHT	Alto
G_CENTER_X	Coordenada X del centro
G_CENTER_Y	Coordenada Y del centro
G_PITCH	Distancia entre filas
G_DEPTH	Profundidad de color (8 ó 16)
G_FRAMES	Número de gráficos en la animación
G_ANIMATION_STEPS	Número total de pasos
G_ANIMATION_STEP	Número de paso actual
G_ANIMATION_SPEED	Velocidad en ms de la animación

Funcionamiento: Esta es una función multiuso que permite obtener información sobre un gráfico cualquier que haya en memoria. El primer y segundo parámetro identifican el gráfico, mientras el tercero escoge el tipo de información que se desea recibir, y debe ser una de las constantes anteriores.

Es posible emplear los identificadores especiales de gráfico BACKGROUND y SCREEN para obtener información sobre el fondo de pantalla, y sobre la pantalla en sí tal cual fue dibujada durante el frame anterior, respectivamente. En este caso es preciso especificar un código de librería 0.

Los cuatro últimos parámetros sólo tienen sentido en el caso de que el gráfico sea una animación. Un gráfico en un fichero .MAP, o que forme parte de un fichero .FPG, puede ser una animación, en cuyo caso está compuesto por una serie de gráficos de los cuales uno está activo en cada momento. Las funciones habituales de acceso a gráficos sólo modifican el gráfico actual. Además, la animación consta de una serie de pasos predefinidos que se ejecutan secuencialmente: cada paso indica un código de gráfico, de manera que no es necesario que los gráficos se muestren ordenadamente. Mediante esta función es posible obtener el número total de gráficos, el número de pasos, el paso actual, y la velocidad de la animación en milisegundos (esta velocidad es independiente de los fps elegidos por la función set_fps).

Las coordenadas del centro se dan de tal manera que (0, 0) corresponde a la esquina superior izquierda del gráfico.

g_pitch devuelve, en algunos casos, el mismo valor que g_wide. No obstante, no siempre ocurre así. g_pitch indica el número real de pixels que hay entre una fila del gráfico y la siguiente en memoria. Este valor interno sólo resulta de utilidad para quien desee acceder directamente a los datos del gráfico empleando punteros y map_buffer. Para todos los usos habituales, debe emplearse g_wide.

graphic_set (int *librería*, int *gráfico*, int *tipo*, int *valor*)

Parámetros:

librería Código de librería a la que pertenece el gráfico

gráfico Identificador del gráfico dentro de la librería

tipo Especifica qué parámetro del gráfico se desea modificar. Puede ser una de las siguientes constantes predefinidas:

G_CENTER_X	Coordenada X del centro
G_CENTER_Y	Coordenada Y del centro
G_ANIMATION_STEP	Paso actual de la animación
G_ANIMATION_SPEED	Velocidad de la animación en ms

Funcionamiento: Esta es una función multiuso que permite modificar datos concretos de un gráfico, de forma complementaria a la función graphic_info. Su principal utilidad consiste en alterar una animación en curso.

Una velocidad de 0 ms detiene la animación. Por ejemplo, la siguiente instrucción detiene la animación del proceso actual:

```
graphic_set (file, graph, G_ANIMATION_SPEED, 0) ;
```

unload_map (int *librería*, int *gráfico*)

Parámetros:

librería Código de librería a la que pertenece el gráfico

gráfico Identificador del gráfico dentro de la librería

Funcionamiento: Descarga un gráfico de memoria, sin importar su procedencia. Dicho gráfico puede haber sido creado manualmente con new_map, cargado de disco con load_map, o formar parte de una librería cargada con load_fpg, por ejemplo. De cualquiera de las maneras, la memoria empleada por el gráfico es liberada y cualquier acceso posterior al gráfico se considera inválido.

Se considera responsabilidad del programador no descargar nunca un gráfico que pueda estar en uso por algún proceso.

unload_fpg (int *librería*)

Parámetros:

librería Código de la librería a descargar

Funcionamiento: Descarga una librería entera de gráficos de memoria, que haya sido cargada previamente con la función load_fpg. Cualquier acceso a un gráfico de esta librería se considerará inválido a partir del momento de llamar a esta función. Se considera responsabilidad del programador no descargar nunca una librería entre cuyos gráficos hay alguno que pueda estar en uso por algún proceso.

int map_clone (int *librería*, int *gráfico*)

Parámetros:

librería Código de librería a la que pertenece el gráfico

gráfico Identificador del gráfico dentro de la librería

Funcionamiento: Crea un nuevo gráfico a partir de otro. El nuevo gráfico tendrá el mismo tamaño, profundidad de color, puntos de control y demás características (además del gráfico en sí). Sin embargo, el gráfico creado no pertenecerá al mismo fichero que el original, sino al fichero 0. El código del nuevo gráfico será el devuelto por la función.

En el caso de que haya una librería 0 cargada, los gráficos creados con esta función no se borrarán de memoria al descargarse dicha librería. Será preciso utilizar de forma independiente la instrucción unload_map con cada uno de ellos para descargarlos de memoria.

int new_map (int *ancho*, int *alto*, int *profundidad*)

Parámetros:

ancho	Ancho en pixels del nuevo gráfico
alto	Alto en pixels del nuevo gráfico
profundidad	Profundidad de color (8 ó 16) del nuevo gráfico

Funcionamiento: Crea un nuevo gráfico en memoria. Dicho gráfico podrá utilizarse en cualquier lugar donde se emplee un identificador de gráfico: para ello, debe emplearse como si perteneciera a la librería 0. Por ejemplo, puede asignarse el valor devuelto por new_map para asignarlo a la variable local graph de un proceso, o usarlo como fondo de pantalla con un put_image.

Los parámetros indican el ancho y alto del nuevo gráfico, así como la profundidad de color, que puede ser 8 ó 16, para indicar un gráfico de 8 bits o de 16 bits respectivamente.

Los gráficos creados con esta función tienen todos los pixels inicialmente a 0 (transparentes).

5.8 Puntos de control

get_point (int *librería*, int *gráfico*, int *punto*, pointer *x*, pointer *y*)

Parámetros:

librería	Código de librería del gráfico
gráfico	Identificador del gráfico dentro de la librería
punto	Número del punto de control a obtener (0 para el centro)
x	Dirección de memoria de una variable entera donde la función almacenará la coordenada X del punto de control especificado.
y	Dirección de memoria de una variable entera donde la función almacenará la coordenada Y del punto de control especificado.

Funcionamiento: Esta función recupera las coordenadas locales de un punto de control de un gráfico. Las dos coordenadas del punto son almacenadas en dos variables cuyas direcciones recibe esta función. Para pasarle estos dos parámetros, es preciso emplear el operador & (también válido offset) con dos variables locales, globales o privadas de tipo entero. Por ejemplo:

```
get_point (0, grafico, 0, &x, &y);
```

Las coordenadas se devolverán de tal manera que la coordenada (0,0) corresponda a la esquina superior izquierda del gráfico.

Es posible que el gráfico no contenga un punto de control con el número especificado. En ese caso, esta función no altera el contenido de las dos variables cuyas direcciones recibe.

El punto de control 0 equivale al centro del gráfico. Sin embargo, es posible que el gráfico no tenga ni siquiera un centro asignado. A todos los efectos el centro del gráfico en ese caso se considera el centro de éste, pero esta función no recuperará sus coordenadas.

get_real_point (int *punto, pointer x*, pointer *y*)

Parámetros:

punto	Número del punto de control a obtener (0 para el centro)
x	Dirección de memoria de una variable entera donde la función almacenará la coordenada X en pantalla del punto de control especificado.
y	Dirección de memoria de una variable entera donde la función almacenará la coordenada Y en pantalla del punto de control especificado.

Funcionamiento: Esta función recupera un punto de control del gráfico asignado al proceso actual, pero en lugar de recuperar la ubicación del punto dentro del gráfico como la función get_point, convierte sus coordenadas a partir de la posición, tamaño, ángulo y resolución del proceso actual. El resultado es que la función permite obtener las coordenadas en pantalla de un punto de control del gráfico. Por lo demás el funcionamiento es equivalente a la función get_point.

El punto de control 0 equivale al centro del gráfico. Sin embargo, es posible que el gráfico no tenga ni siquiera un centro asignado, en cuyo caso el centro del gráfico en ese caso se considera el centro real de éste, y esta función recuperará sus coordenadas.

set_point (int *librería*, int *gráfico*, int *punto*, int *x*, int *y*)

Parámetros:

librería	Código de librería del gráfico
gráfico	Identificador del gráfico dentro de la librería
punto	Número del punto de control a modificar (0 para el centro)
x	Nuevo valor de la coordenada X del punto de control
y	Nuevo valor de la coordenada Y del punto de control

Funcionamiento: Esta función cambia un punto de control de un gráfico en memoria, a las coordenadas indicadas. No es necesario que el punto de control estuviese definido previamente. Un gráfico puede contener hasta 1000 puntos de control, numerados del 0 al 999. El punto de control 0 equivale al centro del gráfico, y puede ser cambiado con esta función como si de cualquier otro se tratase.

La coordenada (0, 0) equivale a la esquina superior izquierda del gráfico.

set_center (int *librería*, int *gráfico*, int *x*, int *y*)

Parámetros:

librería	Código de librería del gráfico
gráfico	Identificador del gráfico dentro de la librería
x	Nuevo valor de la coordenada X del punto de control
y	Nuevo valor de la coordenada Y del punto de control

Funcionamiento: Esta función cambia el centro de un gráfico en memoria (es decir, el punto de control 0), a las coordenadas indicadas. Por lo demás equivale a set_point.

5.9 Regiones

define_region (int *region, int x*, int *y*, int *ancho*, int *alto*)

Parámetros:

region	Número de región a definir, de 1 a 31
x	Coordenada X en pantalla de la esquina superior izquierda de la región
y	Coordenada Y en pantalla de la esquina superior izquierda de la región
ancho	Ancho en pixels de la región
alto	Alto en pixels de la región

Funcionamiento: Define los parámetros de una de las 32 regiones diponibles. No es válido redefinir la región 0, que siempre contiene los límites de la pantalla. En el caso de que los parámetros especifiquen una región que sobresalga de pantalla, su tamaño se ajustará para que quepa en el interior.

Las regiones tienen diversas utilidades, como especificar la zona de pantalla que ocupa un scroll o una o más áreas de juego independientes, o indicar una zona límite a emplear con la función out_region.

int out_region (int *proceso*, int *region*)

Parámetros:

proceso	Identificador de un proceso activo
region	Código de región (de 0 a 32)

Funcionamiento: Esta función devuelve 1 (cierto) si el proceso indicado, en caso de dibujarse con las variables de posición, tamaño y ángulo actuales, quedaría completamente fuera de la región indicada. La región 0 corresponde a la pantalla entera.

5.10 Dibujo de gráficos sobre el fondo de pantalla

Este juego de funciones permite alterar el fondo de pantalla. Su mayor utilidad reside en la comodidad, ya que todas ellas disponen de una función equivalente en el apartado siguiente que permite realizar la misma operación sobre un gráfico cualquiera (incluyendo el propio fondo de pantalla, empleando 0 como identificador de librería y la constante predefinida background como identificador de gráfico).

put (int *librería*, int *gráfico*, int *x*, int *y*)

Parámetros:

librería	Código de librería del gráfico
gráfico	Identificador del gráfico dentro de la librería
x	Coordenada X
y	Coordenada Y

Funcionamiento: Esta función simplemente dibuja el gráfico sobre en el fondo de pantalla, sobreescribiendo cualquier gráfico que hubiera bajo él, pero respetando sin embargo los pixels a 0.

La coordenada (X, Y) corresponderá al centro del gráfico en pantalla. Este centro puede variar si el gráfico contiene un punto de control 0.

put_screen (int *librería*, int *gráfico*)

Parámetros:

librería Código de librería del gráfico

gráfico Identificador del gráfico dentro de la librería

Funcionamiento: Esta función dibuja el gráfico indicado centrado como fondo de pantalla. Equivale a la función put con las coordenadas X, Y adecuadas.

put_pixel (int *x*, int *y*, int *color*)

Parámetros:

x Coordenada X en pantalla del pixel

y Coordenada Y en pantalla del pixel

color Indica un número de color (de 0 a 255 en modos de 8 bits, o de 0 a 65535 en modos de 16 bits)

Funcionamiento: Esta función escribe un pixel en el fondo de pantalla. Hay que tener en cuenta que mientras que en modo de 8 bits el valor del color es directamente un número de color en la paleta, en modo de 16 bits la representación del color varía de un ordenador a otro y es preciso utilizar la función rgb para averiguar el número adecuado a pasar a esta función, dado un color determinado.

int get_pixel (int *x*, int *y*)

Parámetros:

x Coordenada X en pantalla del pixel

y Coordenada Y en pantalla del pixel

Funcionamiento: Esta función recupera el color de un pixel del fondo de pantalla, dadas sus coordenadas. En modo de 8 bits este color corresponde directamente a un número de color en la paleta, de 0 a 255. En modo de 16 bits el significado del valor devuelto por esta función varía según el ordenador y es preciso usar la función get_rgb para conocer su significado real.

xput (int *librería*, int *gráfico*, int *x*, int *y*, int *ángulo*, int *tamaño*, int *flags*, int *region*)

Parámetros:

librería Código de librería del gráfico

gráfico Identificador del gráfico dentro de la librería

x Coordenada X

y Coordenada Y

ángulo Ángulo del gráfico, en milésimas de grado

tamaño Tamaño del gráfico, en porcentaje. 100 para respetar el tamaño original.

flags Contiene un código que permite modificar el funcionamiento de la función de dibujo para obtener distintos efectos. El código puede ser una suma de cualesquiera de los valores de la tabla siguiente:

1	Espejo (inversión) horizontal
2	Espejo (inversión) vertical
4	Transparencia
128	Trata el color 0 como uno más

region Indica un código de región. El gráfico será recortado de manera que ningún punto sobresalga fuera de esta región. 0 para la pantalla completa.

Funcionamiento: Esta función dibuja el gráfico indicado sobre en el fondo de pantalla, con todas las opciones posibles de dibujo, como si de un proceso se tratase.

5.11 Dibujo de gráficos sobre otros gráficos

map_clear (int *librería*, int *gráfico*, int *color*)

Parámetros:

librería Identificador de la librería a la que pertenece el gráfico

gráfico Identificador del gráfico, dentro de la librería

color Indica un número de color (de 0 a 255 en modos de 8 bits, o de 0 a 65535 en modos de 16 bits)

Funcionamiento: Esta función borra el contenido de un gráfico en memoria, asignando un mismo color a todos los pixels del gráfico. Hay que tener en cuenta que mientras que en modo de 8 bits el valor del color es directamente un número de color en la paleta, en modo de 16 bits la representación del color varía de un ordenador a otro y es preciso utilizar la función rgb para averiguar el número adecuado a pasar a esta función, dado un color determinado.

El color especial 0, tanto en 8 como en 16 bits, indica "ningún color". A través de esos pixels, completamente transparentes, puede verse el fondo de pantalla u otros gráficos que hubiera detrás de éste antes de dibujarlo.

int map_get_pixel (int *librería*, int *gráfico*, int *x*, int *y*)

Parámetros:

librería Identificador de la librería a la que pertenece el gráfico

gráfico Identificador del gráfico, dentro de la librería

x Coordenada X en el gráfico del pixel

y Coordenada Y en el gráfico del pixel

Funcionamiento: Esta función recupera el color de un pixel del interior de un gráfico, dadas sus coordenadas.

En modo de 8 bits este color corresponde directamente a un número de color en la paleta, de 0 a 255. En modo de 16 bits el significado del valor devuelto por esta función varía según el ordenador y es preciso usar la función get_rgb para conocer su significado real.

int map_put_pixel (int *librería*, int *gráfico*, int *x*, int *y*, int *color*)

Parámetros:

librería	Identificador de la librería a la que pertenece el gráfico
gráfico	Identificador del gráfico, dentro de la librería
x	Coordenada X en el gráfico del pixel
y	Coordenada Y en el gráfico del pixel
color	Color a asignar al pixel

Funcionamiento: Esta función asigna el color dado a un pixel del interior de un gráfico, dadas sus coordenadas.

Hay que tener en cuenta que mientras que en modo de 8 bits el valor del color es directamente un número de color en la paleta, en modo de 16 bits la representación del color varía de un ordenador a otro y es preciso utilizar la función rgb para averiguar el número adecuado a pasar a esta función, dado un color determinado.

map_put (int *librería*, int *gráfico*, int *gráfico2, int x*, int *y*)

Parámetros:

librería	Código de librería del gráfico
gráfico	Identificador del gráfico de destino dentro de la librería
gráfico2	Identificador del gráfico a dibujar dentro de la misma librería
x	Coordenada X
y	Coordenada Y

Funcionamiento: Esta función dibuja un gráfico directamente sobre otro, utilizando unas coordenadas dentro del gráfico de destino, como si éste fuera el fondo de pantalla.

Una limitación de esta función es que el gráfico de origen y el de destino deben pertenecer a la misma librería. Habitualmente los gráficos de destino suelen ser gráficos temporales creados con new_map . Para solventarla, se aconseja cargar en primer lugar la librería que contenga los gráficos que con más frecuencia van a ser empleados como origen en esta función: de esta forma ambos gráficos contendrán el número de librería 0. Alternativamente se puede usar map_clone para obtener una copia de un gráfico, pero perteneciente a la librería 0 (dado que el nuevo gráfico ocupa espacio adicional de memoria, es posible descargar el antiguo mediante unload_map).

map_xput (int *librería*, int *gráfico*, int *gráfico2,* int *x*, int *y*, int *ángulo*, int *tamaño*, int *flags*, int *region*)

Parámetros:

librería	Código de librería del gráfico
gráfico	Identificador del gráfico de destino dentro de la librería
gráfico2	Identificador del gráfico a dibujar dentro de la misma librería
x	Coordenada X
y	Coordenada Y

ángulo Ángulo del gráfico, en milésimas de grado

tamaño Tamaño del gráfico, en porcentaje. 100 para respetar el tamaño original.

flags Contiene un código que permite modificar el funcionamiento de la función de dibujo para obtener distintos efectos. El código puede ser una suma de cualesquiera de los valores de la tabla siguiente:

1	Espejo (inversión) horizontal
2	Espejo (inversión) vertical
4	Transparencia
128	Trata el color 0 como uno más

region Indica un código de región. El gráfico será recortado de manera que ningún punto sobresalga fuera de esta región. 0 para la pantalla completa.

Funcionamiento: Esta función dibuja el gráfico de origen sobre el gráfico de destino, empleando coordenadas locales al gráfico de destino y con todas las opciones posibles de dibujo, como si de un proceso se tratase.

Una limitación de esta función es que el gráfico de origen y el de destino deben pertenecer a la misma librería. Habitualmente los gráficos de destino suelen ser gráficos temporales creados con new_map. Para solventarla, se aconseja cargar en primer lugar la librería que contenga los gráficos que con más frecuencia van a ser empleados como origen en esta función: de esta forma ambos gráficos contendrán el número de librería 0. Alternativamente se puede usar map_clone para obtener una copia de un gráfico, pero perteneciente a la librería 0 (dado que el nuevo gráfico ocupa espacio adicional de memoria, es posible descargar el antiguo mediante unload_map).

map_block_copy (int *librería*, int *destino*, int *dx*, int *dy*, int *origen*, int *ox*, int *oy*, int *ancho*, int *alto)*

Parámetros:

librería Identificador de librería devuelto por load_fpg

destino Identificador del gráfico de destino, dentro de la librería

dx Coordenada X de la esquina superior izquierda en el bloque de destino

dy Coordenada Y de la esquina superior izquierda en el bloque de destino

origen Identificador del gráfico de origen, dentro de la librería

ox Coordenada X de la esquina superior izquierda en el bloque de origen

oy Coordenada Y de la esquina superior izquierda en el bloque de origen

ancho Ancho en pixels del bloque a copiar

alto Alto en pixels del bloque a copiar

Funcionamiento: Copia un fragmento rectangular dentro de un gráfico, al interior de otro gráfico, posiblemente en coordenadas diferentes. Los gráficos deben ser diferentes; si se desea hacer una copia de parte de un gráfico dentro de sí mismo, es preciso trabajar con una copia del gráfico empleando map_clone. Una limitación de esta función es que el gráfico de origen y el de destino deben pertenecer a la misma librería.

5.12 Dibujo de primitivas gráficas

drawing_color (int color)

Parámetros:

color Un número de color de 0 a 255 en modo de 8 bits, o bien un número de 0 a 65535 obtenido con la función rgb, en modo de 16 bits.

Funcionamiento: Selecciona el color que deben usar, a partir de su llamada, las funciones de dibujo de primitivas gráficas como draw_line, draw_rect, etc.

drawing_map (int *librería*, int *gráfico*)

Parámetros:

librería Código de librería del gráfico

gráfico Identificador del gráfico de destino para las primitivas gráficas

Funcionamiento: Selecciona el gráfico que servirá a partir de esta llamada, de destino a todas las funciones de dibujo de primitivas gráficas que tengan el prefijo draw. Es *imprescindible* llamara a esta función antes que a ninguna otra función de dibujo de primitivas.

El gráfico especial (0, background) selecciona el fondo de pantalla.

draw_line (int *x1*, int *y1*, int *x2*, int *y2*)

Parámetros:

x1 Coordenada X del primer punto

y1 Coordenada Y del primer punto

x2 Coordenada X del segundo punto

y2 Coordenada Y del segundo punto

Funcionamiento: Traza una línea entre dos puntos, sobre un mapa o el fondo actual de pantalla. Cualquiera de los puntos (o ambos) puede estar fuera de la zona del gráfico sin que ello produzca error. El último punto no forma parte de los contenidos por la línea.

draw_rect (int *x1*, int *y1*, int *x2*, int *y2*)

Parámetros:

x1 Coordenada X del primer punto

y1 Coordenada Y del primer punto

x2 Coordenada X del segundo punto

y2 Coordenada Y del segundo punto

Funcionamiento: Traza un rectángulo (sin rellenar, sólo el borde) entre dos puntos, por los que el rectángulo pasa. Antes de llamar a esta función, es preciso preparar las funciones de dibujo empleando una llamada a drawing_map para seleccionar un gráfico, y seguramente también a drawing_color para elegir el color de dibujo.

draw_box (int *x1*, int *y1*, int *x2*, int *y2*)

Parámetros:

x1	Coordenada X del primer punto
y1	Coordenada Y del primer punto
x2	Coordenada X del segundo punto
y2	Coordenada Y del segundo punto

Funcionamiento: Traza un rectángulo (rellenado, todos los pixels del interior son escritos con el mismo color) entre dos puntos, por los que el rectángulo pasa. Antes de llamar a esta función, es preciso preparar las funciones de dibujo empleando una llamada a drawing_map para seleccionar un gráfico, y seguramente también a drawing_color para elegir el color de dibujo.

draw_circle (int *x*, int *y*, int *radio*)

Parámetros:

x	Coordenada X del centro del círculo
y	Coordenada Y del centro del círculo
radio	Radio del círculo

Funcionamiento: Dibuja un círculo no relleno en el gráfico actual de dibujo, a partir del punto de su centro y del radio. Antes de llamar a esta función, es preciso preparar las funciones de dibujo empleando una llamada a drawing_map para seleccionar un gráfico, y seguramente también a drawing_color para elegir el color de dibujo.

draw_fcircle (int *x*, int *y*, int *radio*)

Parámetros:

x	Coordenada X del centro del círculo
y	Coordenada Y del centro del círculo
radio	Radio del círculo

Funcionamiento: Dibuja un círculo relleno (todos los pixels en el interior del círculo son dibujados con el color seleccionado con drawing_color) en el gráfico actual de dibujo, a partir del punto de su centro y del radio. Antes de llamar a esta función, es preciso preparar las funciones de dibujo empleando una llamada a drawing_map para seleccionar un gráfico, y seguramente también a drawing_color para elegir el color de dibujo.

5.13 Paleta de colores

load_pal (string *fichero*)

Parámetros:

fichero	Nombre del fichero PAL, MAP o FPG

Funcionamiento: Recupera una paleta de colores de un fichero. La paleta actual de colores es sustituída por la contenida en el fichero especificado. Esta función puede recuperar la paleta de un fichero MAP o FPG de 8 bits (el resto de contenido gráfico no será cargado en memoria) o específicamente de un fichero PAL.

fade (int *r*, int *g*, int *b*, int *velocidad*, int *dirección*)

Parámetros:

r	Componente R del color de destino (0-63)
g	Componente G del color de destino (0-63)
b	Componente B del color de destino (0-63)
velocidad	Máximo cambio de una componente de color por frame. Cuanto menor sea este número más rápido funcionará el fade. (1-63)
dirección	-1 para hacer un fundido hacia el color indicado - para hacer un fundido desde el color indicado hasta la paleta de colores actual

Funcionamiento: Inicializa un fundido de colores, alterando cada frame la paleta de colores temporalemente de manera que los colores se conviertan progresivamente en un color determinado cuyas componentes recibe esta función. Estos campos a la paleta, sin embargo, son sólo de nivel interno, y las funciones que recogen los valores de color de ésta, no se verán afectadas.

El fundido de colores empezará a tomar efecto en el frame siguiente, y progresivamente irá tomando efecto según el parámetro de velocidad.

Normalmente el fundido suele emplearse para hacer transiciones entre pantallas, aunque también es útil para lograr efectos de color que afecten a toda la pantalla. Cambiar la paleta de colores es una operación inmediata que no enlentece el desarrollo del programa.

En modos de 16 bits, sólo los gráficos de 8 bits que formen parte de los procesos, y el fondo de pantalla, en el caso de que sólo se haya dibujado un gráfico de 8 bits sobre él con put_screen, se ven afectados por la paleta de colores, lo cual obliga a buscar un método alternativo para hacer transiciones de pantalla.

fade_on ()

Funcionamiento: Restaura un fundido de colores desde el negro. Equivale a llamar a la función fade con los parámetros (0, 0, 0, 16, 1), indicando un fundido desde el color negro de velocidad media-rápida. Este fundido seguirá activo mientras los siguientes frames continúen.

fade_off()

Funcionamiento: Pone toda la paleta en negro progresivamente (un fundido al negro). A diferencia de fade_on(), esta función no vuelve inmediatamente, sino que efectúa el fundido completamente antes de volver. Deja la paleta de colores en la situación ideal para hacer cualquier cambio necesario a pantalla y llamar a fade_on().

int find_color (int *r*, int *g*, int *b*)

Parámetros:

r	Componente R del color a buscar (0-63)
g	Componente G del color a buscar (0-63)
b	Componente B del color a buscar (0-63)

Funcionamiento: Esta función devuelve el número de color de la paleta de colores (0-255) que más se aproxime al color cuyas componentes recibe como parámetro. Es una función muy rápida.

El número resultante sólo es válido emplearlo como color a la hora de escribir o dibujar sobre gráficos de 8 bits. En modos de 16 bits, es necesario usar la función rgb para obtener un número de color válido.

int rgb (int *r*, int *g*, int *b*)

Parámetros:

r	Componente R del color a buscar (0-63)
g	Componente G del color a buscar (0-63)
b	Componente B del color a buscar (0-63)

Funcionamiento: Esta función devuelve el color de 16 bits cuyas componentes recibe como parámetro, según el hardware actual. En función de la tarjeta, sistema operativo y modo gráfico actual el significado de un color de 16 bits puede variar. Es preciso utilizar esta función para obtener el código de color adecuado y nunca almacenar este código en un fichero que pueda ser utilizado posteriormente desde otro ordenador.

La función requiere que el modo gráfico actual sea de 16 bits (lo cual implica que la función graph_mode debe contener la constante mode_16bits antes de llamar a la función set_mode). En modo de 8 bits esta función equivale a find_color, lo cual puede resultar conveniente para la facilidad de uso.

get_rgb (int *color*, pointer *r*, pointer *g*, pointer *b*)

Parámetros:

color	Un color en 8 o 16 bits en función del modo gráfico actual
r	La dirección de una variable tipo INT, que la función rellenará con el valor de la componente R del color dado.
g	La dirección de una variable tipo INT, que la función rellenará con el valor de la componente G del color dado.
b	La dirección de una variable tipo INT, que la función rellenará con el valor de la componente B del color dado.

Funcionamiento: Esta función recupera las componentes de un color dado. En modo de 16 bits, un mismo color se puede representar de distintas maneras en función de la tarjeta gráfica y otros parámetros. Es preciso emplear esta función para interpretar el valor devuelto por funciones como get_pixel.

Si el modo gráfico es de 8 bits, la función devuelve las componentes del color de la paleta especificado (entre 0 y 255 inclusive).

convert_palette (int *librería*, int *gráfico*, pointer *tabla*)

Parámetros:

librería	Código de librería del gráfico a modificar
gráfico	Identificador del gráfico dentro de la librería
tabla	Puntero a una tabla de 256 datos del tipo INT

Funcionamiento: Esta función permite hacer alteraciones en un gráfico de 8 bits. Recibe como parámetro un puntero a una tabla de 256 entradas: la primera entrada indica el color por el cual se deben cambiar todos los puntos que estén a 0 en el gráfico, la segunda entrada indica el color por el cual cambiar el color número 1, y así sucesivamente hasta la última entrada.

Esta función requiere un uso algo avanzado del lenguaje, pero permite realizar de forma relativamente rápida cambios interesantes en los gráficos.

roll_palette (int *primero*, int *número*, int *desplazamiento*)

Parámetros:

primero Número del primer color de un rango de colores de la paleta

número Número de colores de que consta el rango

desplazamiento Cantidad de desplazamiento

Funcionamiento: Esta función desplaza un rango de colores de la paleta. Si el valor de desplazamiento es positivo, el desplazamiento se realizará hacia la derecha, y hacia la izquierda si es negativo. Los colores se desplazarán hacia dicha dirección tantas posiciones como el valor absoluto del desplazamiento, de manera que los colores que desaparezcan por un lado reaparecerán por el contrario.

Con una elección adecuada de paleta de colores, es posible hacer efectos de cambio de color e incluso un movimiento limitado. Dado que los cambios de paleta son inmediatos, resulta mucho más económico para la CPU que tener que dibujar a mano las animaciones.

5.14 Dibujo de Textos

int load_fnt (string *fichero*)

Parámetros:

fichero Nombre del fichero .FNT a cargar

Funcionamiento: Esta función recupera de disco un tipo de letra FNT y lo carga en memoria. Prácticamente todas las operaciones realizadas con textos requieren un identificador de tipo de letra. Aunque el tipo de letra del sistema (con código fijo 0) siempre está disponible, es práctica general emplear un tipo de letra personalizado para cada juego (o varios), lo cual requiere cargarlos antes con esta función.

unload_fnt (int *fuente*)

Parámetros:

fuente Identificador de un tipo de letra cargado con la función load_fnt

Funcionamiento: Esta función descarga de memoria un tipo de letra cargado previamente mediante la función load_fnt. Ningún acceso posterior al tipo de letra será válido. Es responsabilidad del programador no descargar de memoria ningún tipo de letra que esté en uso (por ejemplo, cuando hay textos en pantalla escritos con write o write_int que usan ese tipo de letra).

int write (int *fuente*, int *x*, int *y*, int *centrado*, string *texto*)

Parámetros:

fuente Identificador de un tipo de letra cargado con la función load_fnt.

x Coordenada X donde dibujar el texto

y Coordenada Y donde dibujar el texto

centrado Código que indica cómo interpretar el punto (x, y) respecto al texto a dibujar. Puede ser uno de los valores de la siguiente tabla:

0	Esquina superior izquierda
1	Centro superior
2	Esquina superior derecha
3	Centro izquierda
4	Centro
5	Centro derecha
6	Esquina inferior izquierda
7	Centro inferior
8	Esquina inferior derecha

texto Cadena a escribir en pantalla

Funcionamiento: Esta función escribe un texto fijo en pantalla. Todos los textos escritos con la función write permanecen activos por encima de los procesos: cada frame se dibujan como si del gráfico de un proceso se tratase.

La utilización de este parámetro permite justificar el texto a partir de las coordenadas. Así, un valor de 4 centrará el texto en el punto pasado en los parámetros x, y a la función.

Esta función devuelve un identificador del texto. La función delete_text permite emplear este identificador para eliminar este texto en concreto. Es preciso tener mucho cuidado para no llamar a la función write a cada frame sin haber borrado el texto del frame anterior, ya que ambos se dibujarán en pantalla al frame próximo y la acumulación de textos enlentecerá la ejecución del programa.

Existe un máximo de 512 textos simultáneos en pantalla. Si se alcanzara, el programa se detendría inmediatamente con un error.

int write_int (int *fuente*, int *x*, int *y*, int *centrado*, pointer *valor*)

Parámetros:

fuente Identificador de un tipo de letra cargado con la función load_fnt.

x Coordenada X donde dibujar el texto

y Coordenada Y donde dibujar el texto

centrado Código que indica cómo interpretar el punto (x, y) respecto al texto a dibujar. Puede ser uno de los valores de la siguiente tabla:

0	Esquina superior izquierda
1	Centro superior
2	Esquina superior derecha
3	Centro izquierda
4	Centro
5	Centro derecha
6	Esquina inferior izquierda
7	Centro inferior
8	Esquina inferior derecha

valor Dirección de una variable en memoria tipo INT

Funcionamiento: Esta función escribe un número en pantalla, con la particularidad de que si el contenido de la variable cambiara, el texto se adaptaría inmediatamente. Para ello es preciso que ese número se encuentra en una variable local, privada o global, cuya dirección puede recibir la función write_int empleando el operador &. Por lo demás la función equivale exactamente a la función write. Todos los textos escritos con la función write permanecen activos por encima de los procesos: cada frame se dibujan como si del gráfico de un proceso se tratase.

La utilización de este parámetro permite justificar el texto a partir de las coordenadas. Así, un valor de 4 centrará el texto en el punto pasado en los parámetros x, y a la función.

Esta función devuelve un identificador del texto. La función delete_text permite emplear este identificador para eliminar este texto en concreto. Es preciso tener mucho cuidado para no llamar a la función write a cada frame sin haber borrado el texto del frame anterior, ya que ambos se dibujarán en pantalla al frame próximo y la acumulación de textos enlentecerá la ejecución del programa.

Existe un máximo de 512 textos simultáneos en pantalla. Si se alcanzara, el programa se detendría inmediatamente con un error.

int write_in_map (int *fuente*, int *centrado*, int *texto*)

Parámetros:

fuente Identificador de un tipo de letra cargado con la función load_fnt.

centrado Código que indica donde ubicar el centro (punto de control 0) en el nuevo gráfico.Puede ser uno de los valores de la siguiente tabla:

0	Esquina superior izquierda
1	Centro superior
2	Esquina superior derecha
3	Centro izquierda
4	Centro
5	Centro derecha
6	Esquina inferior izquierda
7	Centro inferior
8	Esquina inferior derecha

texto Cadena de texto a dibujar

Funcionamiento: Esta función crea un nuevo gráfico en memoria, cuyo contenido será el texto indicado en el tipo de letra dado. El tamaño del gráfico será el mínimo necesario para que el texto encaje exactamente. El nuevo gráfico podrá utilizarse en cualquier lugar donde se emplee un identificador de gráfico: para ello, debe emplearse como si perteneciera a la librería 0. También es posible descargar el gráfico de memoria empleando la función unload_map.

El centro del gráfico (punto de control 0) puede adaptarse según el parámetro de centrado dado. Esto permite emplear el nuevo gráfico de manera similar a una instrucción write, pero con las ventajas de poder usar las funciones más avanzadas presentes en los flags de los procesos o la función xput.

move_text (int *texto*, int *x*, int *y*)

Parámetros:

texto Identificador de un texto, devuelto por write o write_int

x Nueva coordenada X

y Nueva coordenada Y

Funcionamiento: Esta función cambia la posición X, Y de un texto en pantalla. Es preciso que el texto esté visible, y no haya sido borrado anteriormente mediante delete_text.

delete_text (int *texto*)

Parámetros:

texto Identificador de un texto, devuelto por write o write_int, o bien el número 0

Funcionamiento: Elimina un texto de pantalla. En el próximo frame el texto indicado ya no será visible. Si el parámetro es el código especial 0, en lugar de un identificador devuelto por write o write_int, entonces la función borrará *todos* los textos activos.

int text_width (int *fuente*, string *texto*)

int text_height (int *fuente*, string *texto*)

Parámetros:

fuente Identificador de un tipo de letra devuelto por load_fnt, o 0 para indicar el tipo de letra del sistema

texto Cadena de texto

Funcionamiento: Estas funciones calculan y devuelven las dimensiones en pixels (el ancho y el alto, respectivamente) que ocuparía un texto en pantalla en un tipo de letra dado, sin escribirlo en ninguna parte.

5.15 Scroll

start_scroll (int *número*, int *librería*, int *gráfico*, int *fondo*, int *región*, int *flags*)

Parámetros:

número Número de scroll (0-9)

librería Identificador de la librería donde se encuentran los gráficos de fondo

gráfico Gráfico del "suelo" encima del cual se desplazan los procesos, o 0 para ninguno

fondo Gráfico de "fondo" que se desplaza por detrás del gráfico anterior, o 0 para ninguno

región Número de región que delimita el espacio en pantalla ocupado por la zona de scroll

flags Opciones que afectan al funcionamiento del scroll. Puede ser 0 (normal) o bien la suma de uno o más valores de la siguiente tabla:

1	Gráfico principal se repite horizontalmente
2	Gráfico principal se repite verticalmente
4	Gráfico de fondo se repite horizontalmente
8	Gráfico de fondo se repite verticalmente

Funcionamiento: Esta función inicializa una zona de scroll en pantalla. La zona de scroll consiste en uno o dos gráficos sobrepuestos que pueden desplazarse, por encima de los cuales se dibujan aquellos procesos que han sido preparados para visualizarse dentro de un scroll.

Pueden haber hasta 10 scrolls simultáneos en pantalla, diferenciados por un código entre el 0 y el 9. Dicho código identifica el scroll dentro de la estructura global scroll. Para que un proceso se visualice en el interior de un área de scroll (en lugar de en pantalla, como ocurre normalmente) su variable local ctype deberá contener el valor 2^n, donde n es el número de scroll. Para que el procesos se visualice en varias zonas de scroll simultáneamente, pueden sumarse los resultados. Así, con un valor en ctype de 7 $\left(2^0 + 2^1 + 2^2 = 1 + 2 + 4 = 7\right)$, el proceso se visualizará en las áreas de scroll 0, 1 y 2. Una vez un proceso tiene una variable ctype distinta de cero, sus coordenadas X e Y pasan a referirse a la zona de scroll y dejan de ser coordenadas en pantalla.

El tamaño de esta zona de scroll normalmente está limitada por el tamaño de los gráficos que se utilicen como suelo o fondo. Así, con un gráfico de fondo de 1024x1024, sólo pueden visualizarse zonas del tamaño de la pantalla que entren dentro de este rango de coordenadas. Si se desea un área ilimitada, es preciso indicar en el parámetro flags que estos gráficos se repiten horizontal, verticalmente o en ambas direcciones (formando así una trama o textura).

La función start_scroll inicializa el scroll y es necesaria para especificar los gráficos y la zona ocupada por el scroll, sin embargo la estructura grobal scroll permite modificar en mitad de la ejecución normal del juego parámetros del scroll tales como velocidad de movimiento, proceso al cual el scroll debe "seguir", activar la transparencia en los gráficos, y otras posibilidades de cada scroll. En el capítulo de datos globales se documentan los componentes de esta estructura.

stop_scroll (int *número*)

Parámetros:

número	Número de scroll (0-9)

Funcionamiento: Esta función desactiva un área de scroll. Dicha área dejará de dibujarse en pantalla a partir del próximo frame.

move_scroll (int *número*)

Parámetros:

número	Número de scroll (0-9)

Funcionamiento: Esta función actualiza las coordenadas de visibilidad del área de scroll indicada, en función del contenido de la estructura global scroll. En las últimas versiones de Fenix, el área de scroll se actualiza automáticamente y esta función se considera obsoleta.

5.16 Animaciones FLI/FLC

int start_fli (string *nombre*, int *x*, int *y*)

Parámetros:

nombre	Nombre del fichero FLI/FLC a abrir
x	Coordenada X de la esquina superior izquierda
y	Coordenada Y de la esquina superior izquierda

Funcionamiento: Esta función inicia la reproducción de un fichero FLI/FLC desde el disco duro. El fichero FLI/FLX puede posicionarse en pantalla en las coordenadas deseadas (aunque lo más común es emplear un fichero FLI/FLC que ocupe la pantalla completa y emplear las coordenadas 0,0).

La reproducción del FLI es automática, e independiente del funcionamiento de procesos y otros elementos. La velocidad de reproducción es la indicada originalmente en el fichero, independientemente de los frames por segundo a los que el juego funcione.

int frame_fli ()

Funcionamiento: Esta función devuelve 1 (CIERTO) si hay un fichero FLI/FLC en ejecución y 0 si no lo hay o el que había ya ha acabado de reproducirse.

end_fli ()

Funcionamiento: Esta función detiene la reproducción de cualquier FLI/FLC que estuviera en activo.

reset_fli ()

Funcionamiento: Esta función reinicia la reproducción del FLI/FLC (si hubiera uno en activo) desde el comienzo.

5.17 Búsqueda de caminos

int path_find (int *librería*, int *gráfico*, int *x1*, int *y1*, int *x2*, int *y2*, int *opciones*)

Parámetros:

librería Identificador de la librería a la que pertenece el gráfico de obstáculos

gráfico Identificador del gráfico de obstáculos dentro de la librería

x1 Coordenada X del punto de partida

y1 Coordenada Y del punto de partida

x2 Coordenada X del punto de destino

y2 Coordenada Y del punto de destino

opciones Puede ser 0 para una búsqueda normal, o bien la suma de una o más de las siguientes constantes predefinidas:

PF_NODIAG	No permite el movimiento en diagonal
PF_REVERSE	Devuelve los puntos en orden inverso

Funcionamiento: A partir de un gráfico *de 8 bits* especificado con los parámetros fichero y grafico, resuelve un camino que parta desde el punto en (x_inicial, y_inicial) y llegue hasta el punto en (x_final, y_final), evitando cualquier obstáculo que pueda encontrar a su paso. No se asegura que el camino encontrado sea óptimo al 100%, sin embargo se acercará razonablemente.

El gráfico de 8 bits contendrá normalmente una representación en miniatura del mapa o de la superficie del juego, donde cada punto del gráfico representará una casilla o un área que puede o no estar ocupada con un obstáculo. El algoritmo supone por defecto que el movimiento en diagonal es perfectamente válido, aunque es posible especificar lo contrario con una opción.

Cada punto del gráfico de 8 bits contiene un color de 0 a 255. En lugar de representar colores, indican para path_find si la casilla en cuestión puede traspasarse (valor 0) o no (valor de 1 o

más). Es posible configurar las rutinas de búsquedas de caminos mediante la función path_wall, de manera que los valores superiores a 0 (hasta un valor máximo, indicado como parámetro a path_wall) indiquen igualmente caminos traspasables, pero con un coste superior, indicado por el propio valor del punto. Así, las rutinas eligirán a veces un recorrido más largo, pero que pase por puntos de menor coste. Esta forma de operar puede tener gran utilidad en juegos en los que el mapa contiene casillas donde el de movimiento difiere; empleando esta facilidad, es posible hacer que los procesos escojan preferentemente moverse a través de carreteras y caminos antes que bosque a través, excepto cuando no hay ninguna carretera propicia para hacer el trayecto.

La función devuelve 1 si se encontró un camino, o 0 si no hay posible comunicación entre los puntos de inicio y fin. Para ir obteniendo sucesivamente los puntos intermedios de que consta el camino, es preciso emplear la función path_getxy.

int path_wall (int *valor*)

Parámetros:

valor Indica el valor mínimo para identificar una casilla no traspasable

Funcionamiento: Modifica el valor de "pared", empleado por la función path_find como el número de color mínimo a partir del cual un punto del gráfico es intraspasable. Por defecto este valor está a 1, lo que significa que sólo los puntos a 0 serán navegados por el camino a resolver. Cuando el valor es superior a 1, los puntos inferiores a éste se interpretan como el "coste" de movimiento de un punto.

Si a la función se le pasa como parámetro un valor menor a 0, no modifica el valor actual de la variable interna. Sin embargo, en ambos casos devolverá el valor final de ésta.

int path_getxy (pointer *x*, pointer *y*)

Parámetros:

x Dirección de memoria de la variable tipo INT donde se almacenará la coordenada X

y Dirección de memoria de la variable tipo INT donde se almacenará la coordenada Y

Funcionamiento: Esta función actualiza dos variables cuyas direcciones de memoria se le pasan, con el contenido del siguiente punto del camino encontrado por la última llamada válida a path_find. La función devuelve 0 si no quedan más puntos en el camino, y en ese caso, no modifica las variables.

Los puntos de inicio y final se encuentran entre los devueltos por la función. La primera llamada, actualizará x e y con los valores del punto de inicio del camino.

El siguiente ejemplo movería el proceso actual a lo largo del camino encontrado por path_find (suponiendo que el mapa de búsqueda es de una cuarta parte del tamaño de la pantalla):

```
WHILE (path_getxy(&x, &y))
    x *= 4;
    y *= 4;
    FRAME;
END
```

5.18 Blendops

int blendop_new ()

Funcionamiento: Crea una nueva tabla blendop. Una tabla blendop sirve para obtener efectos especiales de transparencia en lugar del efecto estandar. Gracias a una de estas tablas, es posible dibujar gráficos coloreados, o con una cantidad de transparencia distinta a la normal de 50%. Sin

embargo una tabla blendop ocupa una gran cantidad de memoria (256K) por lo que se aconseja reducir al máximo el número de tablas utilizadas por el programa.

La función devuelve un código que identifica a la nueva tabla blendop creada. Es conveniente almacenar este código en alguna variable, ya que es necesario utilizarlo en posteriores llamadas a funciones blendop.

El procedimiento normal para utilizar una tabla blendop es:

1. Crearla llamando a la función blendop_new
2. Rellenarla utilizando funciones como blendop_translucency o blendop_tint. Estas funciones permiten especificar el color y el grado de transparencia, y rellenan las tablas internas de forma acorde. Son funciones intensivas que consumen bastante tiempo.
3. Asignar la tabla blendop a un gráfico, mediante blendop_assign

Debido a las características de una tabla blendop, es preferible determinar al comienzo del programa qué tablas van a ser necesarias e inicializarlas allí. No es aconsejable crear o modificar tablas blendop en mitad del juego, ya que ello provocaría una pausa, quizá breve, pero perceptible.

blendop_tint (int *blendop*, float *cantidad*, byte *r*, byte *g*, byte *b*)

Parámetros:

blendop	Identificador de la tabla blendop creada por blendop_new
cantidad	Indica la cantidad de color a mezclar; es un número decimal entre 0 y 1
r	Componente R del color a mezclar
g	Componente G del color a mezclar
b	Componente B del color a mezclar

Funcionamiento: Esta función modifica una tabla blendop de manera que los gráficos que la utilicen se dibujen "entintados" con un color cuyas componentes r, g, b recibe la función como parámetros.

La cantidad de tinta con la que los gráficos aparecerán pintados se especifica como segundo parámetro, y es un número con decimales que puede oscilar entre 0 y 1. Un valor 1.0 indica que todos los pixels del gráfico aparecerán con el color indicado (lo cual no suele ser aconsejable). Un valor de 0.5 indica que cada punto del gráfico será una mezcla a partes iguales del color especificado y del original del gráfico en ese punto.

blendop_translucency (int *blendop*, float *cantidad*)

Parámetros:

blendop	Identificador de la tabla blendop creada por blendop_new
cantidad	Indica la cantidad de color a mezclar; es un número decimal entre 0 y 1

Funcionamiento: Esta función dota a una tabla blendop de transparencia. Los gráficos dibujados con esta tabla serán parcialmente transparentes. El segundo parámetro es un número entre 0 y 1: 0.5 representa la transparencia estandar de Fenix. Aparte de ello se puede usar cualquier valor, teniendo en cuenta que 1 es un objeto completamente opaco y 0, completamente transparente.

blendop_intensity (int *blendop*, float *cantidad*)

Parámetros:

blendop Identificador de la tabla blendop creada por blendop_new

cantidad Indica la cantidad de color a preservar; entre 0 y 1 oscurece el gráfico. Entre 1 y 2 lo ilumina.

Funcionamiento: Esta función crea una tabla blendop que modifica la intensidad de un gráfico al dibujarse, lo cual sirve para oscurecer o iluminar un gráfico. El parámetro ammount indica una cantidad en coma flotante: un número entre 0 y 1 oscurece el gráfico (el 0 representa el negro total), mientras un número superior a 1 sirve para dar más claridad. Se recomienda no usar valores superiores a 2.0 por motivos de precisión.

Esta función sobreescribe los valores de la tabla, por lo que debe ser llamada en primer lugar. Si se desea una tabla blendop que además de variar la intensidad del gráfico realice operaciones de entintado o transparencia, es posible llamar a blendop_tint o blendop_translucency después de ésta.

blendop_identity (int *blendop*)

Parámetros:

blendop Identificador de la tabla blendop creada por blendop_new

Funcionamiento: Reinicializa una tabla blendop, de manera que los gráficos que la tengan asignada serán dibujados opacos y sin modificaciones de color. Esta función puede utilizarse para cambiar el funcionamiento de una tabla blendop, ya que las funciones como blendop_translucency tienen un efecto acumulable.

Una tabla blendop recién creada cumple las características anteriormente citadas, por lo que no es necesario utilizar esta función justo después de llamar a blendop_new.

blendop_free (int *blendop*)

Parámetros:

blendop Identificador de la tabla blendop creada por blendop_new

Funcionamiento: Esta función libera la memoria utilizada por una tabla blendop, cuando no vaya a utilizarse más.

Nota: No es válido liberar con esta función una tabla blendop que aún esté asignada a un gráfico. Se supone que es responsabilidad del programador no liberar una tabla blendop en uso. Dibujar un gráfico que tiene una tabla blendop inexistente asignada puede colgar el programa irremediablemente.

blendop_assign (int *librería*, int *gráfico*, int *blendop*)

Parámetros:

librería Indica el código de librería a la que pertenece el gráfico

gráfico Identificador del gráfico que se desea operar

blendop Identificador de la tabla blendop creada por blendop_new

Funcionamiento: Esta función asigna una tabla blendop a un gráfico. Es importante hacer notar que:

- El gráfico a partir de entonces se dibujará con la transparencia, color y demás parámetros empleados en la creación de la tabla blendop (ver ayuda de la función blendop_new)
- El flag de transparencia (4) será ignorado a partir de entonces a la hora de dibujar el gráfico. Otros parámetros (tamaño, ángulo, espejo horizontal o vertical) serán respetados.

La tabla afecta a *todas* las operaciones, tanto si el gráfico está asignado a un proceso o al ratón en la variable graph, como si se emplea con funciones como map_put. Puede quitarse la asociación de una tabla blendop a un gráfico pasando un 0 como tercer parámetro a esta función, lo cual resulta útil ya que no es válido liberar con la función blendop_free la memoria utilizada por una tabla blendop en uso por algún gráfico.

blendop_apply (int *librería*, int *gráfico*, int *blendop*)

Parámetros:

librería Indica el código de librería a la que pertenece el gráfico

gráfico Identificador del gráfico que se desea operar

blendop Identificador de la tabla blendop creada por blendop_new

Funcionamiento: Esta función aplica una tabla blendop sobre los datos de un gráfico, modificando efectivamente el contenido de éste. Aunque las tablas blendop están preparadas para mezclar un gráfico con un fondo de pantalla u otros gráficos, esta función la aplica contra sí mismo, de manera que la transparencia no tiene sentido, pero los cambios de intensidad o entintado sí afectan al gráfico.

A diferencia de blendop_assign, esta función sólo puede operar con gráficos de 16 bits.

La aplicación progresiva de una o varias tablas blendop sobre un mismo gráfico no tiene el efecto deseado, debido a la poca precisión de color de los modos de 16 bits. Es preferible hacer una copia del gráfico original con MAP_CLONE y operar sobre ella.

blendop_swap (int *blendop*)

Parámetros:

blendop Identificador de la tabla blendop creada por blendop_new

Funcionamiento: Esta función altera el orden de la tabla blendop, de manera que los efectos se apliquen sobre el fondo en lugar de sobre el gráfico. Permite realizar efectos de tipo "lente", donde un gráfico no se dibuja como tal, pero sirve para colorear, oscurecer, o iluminar una región en pantalla. Hay que tener en cuenta que como efecto secundario esta función invierte la cantidad de transparencia: una tabla blendop opaca pasa a ser completamente transparente, y viceversa.

Debe ser llamada después de las funciones que crean la tabla blendop. Por ejemplo, para obtener una tabla blendop que sirve para pintar de rojo zonas de pantalla:

```
i = blendop_new();
blendop_tint (i, 0.50, 255, 0, 0);
blendop_swap (i);
```

En este ejemplo, se puede asignar a un gráfico la tabla blendop cuyo identificador está contenido en la variable i, y dicho gráfico no se dibujará como tal, sino que coloreará en un 50% de rojo los puntos en pantalla que no se correspondan con un punto a 0 en el gráfico.

5.19 Fecha y hora

int time()

Funcionamiento: Devuelve la fecha actual, en formato entero, como el número de segundos transcurridos desde el 1 de Enero de 1970. Esta fecha puede emplearse por la función FTIME para visualizarla en pantalla en formato legible, o almacenarse en fichero o variable para propósitos de comparaciones, por ejemplo.

string ftime (string *formato*, int *tiempo*)

Parámetros:

formato Cadena que especifica el formato a utilizar. Cualquier caracter en la cadena se respeta en la cadena devuelta, con la excepción de los siguientes comandos especiales, que son sustituídos por componentes de la fecha y hora indicadas (atención, las mayúsculas y minúsculas se tratan de diferente forma):

%d	Día, dos dígitos	01-31
%m	Mes, dos dígitos	01-12
%y	Año, dos dígitos	00-99
%Y	Año, cuatro dígitos	
%H	Hora, dos dígitos	00-23
%M	Minuto, dos dígitos	00-59
%S	Segundo, dos dígitos	00-59
%e	Día, uno o dos dígitos	1-31
%k	Hora, uno o dos dígitos	0-23
%a	Nombre abreviado del día	
%A	Nombre completo del día	
%b	Nombre abreviado del mes	
%B	Nombre completo del mes	
%C	Centuria, dos dígitos	19-99
%I	Hora americana, dos dígitos	01-12
%l	Hora americana, uno o dos dígitos	1-12
%p	Coletilla americana de la hora	AM/PM
%P	Coletilla americana de la hora	am/pm
%u	Día de la semana	1-7
%U	Día de la semana empezando por 0	0-6
%j	Día del año, tres dígitos	001-366
%U	Número de semana del año	00-53

tiempo Número de segundos transcurridos desde el 1 de Enero de 1970, según el valor devuelto por la función time()

Funcionamiento: Formatea una fecha y/o hora según una cadena de formato específica.Cualquier texto que no sea un control se respeta tal cual en la cadena de salida. Puedes usar %% para poner un signo % en la salida.

He aquí algunos ejemplos comunes de uso:

```
ftime ("%d/%m/%y", time());
ftime ("%d/%m/%Y", time());
ftime ("%d/%m/%y %H:%M", time());
```

5.20 Acceso a ficheros

int fopen (string *nombre*, int *tipo*)

Parámetros:

nombre Nombre del fichero a abrir o crear

tipo Indica si el fichero debe ser abierto para lectura o bien creado o truncado. Puede ser una de las siguientes constantes:

O_READ	Lectura
O_WRITE	Escritura
O_READWRITE	Lectura/escritura
O_ZREAD	Lectura de ficheros comprimidos
O_ZWRITE	Creación de ficheros comprimidos

Funcionamiento: Abre un fichero en disco, y devuelve un identificador de fichero que debe usarse con las funciones de acceso a disco (fread, fwrite, etc) en cualquier operación posterior con él.

Al abrir un fichero en modo O_WRITE, éste será creado si no existiera anteriormente, y truncado a tamaño 0 si ya tuviera datos anteriormente. Lo mismo es aplicable al modo O_ZWRITE.

Al abrir un fichero con O_ZREAD, no son válidas las funciones FLENGTH ni FSEEK (esta última será emulada, pero teniendo en cuenta que en un fichero comprimido sólo se puede avanzar, nunca retroceder ni ir directamente al final del fichero). Con O_ZWRITE estas dos funciones no son nunca válidas ni pueden emularse.

En caso de que el fichero no pueda abrirse, fopen devolverá 0. Esto ocurrirá por ejemplo si el fichero no existe en un modo de lectura.

Todos los ficheros abiertos con FOPEN **deben** cerrarse obligatoriamente con la función FCLOSE, especialmente con los ficheros abiertos en modo escritura (es posible que parte del fichero sólo se escriba realmente en el disco a la hora de cerrarlo con FCLOSE, quedando el fichero incompleto si no fuese así).

fclose (int *fichero*)

Parámetros:

fichero Identificador de un fichero abierto con fopen

Funcionamiento: Cierra un fichero. Cualquier cambio realizado a un fichero de escritura no será visible para el ususario, en según qué plataforma, hasta que el fichero no esté cerrado. Es necesario cerrar todos los ficheros que hayan sido abiertos. Además, existe un límite impuesto por el sistema operativo al número de ficheros que puede haber abiertos a la vez.

int fread (int *fichero*, pointer *datos*, int *bytes*)

Parámetros:

fichero Identificador de un fichero abierto con fopen

datos Dirección de una zona de memoria

bytes Número de bytes a leer

Funcionamiento: Lee una serie de bytes de un fichero, en formato binario. El identificador de fichero es un identificador del fichero, devuelto por fopen.

Puede utilizarse esta función para leer una zona de memoria antes escrita con la función fwrite. Deben tenerse en cuenta las siguientes consideraciones:

1. La zona de memoria no debe contener cadenas ni punteros
2. El tamaño es la suma, mediante el operador sizeof, de todas variables contenidas en la zona de memoria a recuperar.

int fwrite (int *fichero*, pointer *datos*, int *bytes*)

Parámetros:

fichero Identificador de un fichero abierto con fopen

datos Dirección de una zona de memoria

bytes Número de bytes a escribir

Funcionamiento: Guarda una serie de bytes en un fichero, en formato binario. El identificador de fichero es un identificador del fichero, devuelto por fopen. El fichero debe haberse abierto en modo escritura (o_write, o_readwrite o o_zwrite).

Puede utilizarse esta función para almacenar una zona de memoria que luego podrá recuperarse, en posteriores ejecuciones del programa, mediante la función fread. Deben tenerse en cuenta las siguientes consideraciones:

1. La zona de memoria no debe contener cadenas ni punteros
2. El tamaño es la suma, mediante el operador sizeof, de las variables contenidas en la zona de memoria a recuperar.

fseek (int *fichero*, int *posición*, int *donde*)

Parámetros:

fichero Identificador de un fichero abierto con fopen

posición Número de bytes

donde Indica a partir de dónde contar el número de bytes dado:

0	Desde el comienzo del fichero
1	Desde la posición actual
2	Desde el final del fichero

Funcionamiento: Modifica el punto de lectura/escritura en un fichero. El parámetro posicion puede ser negativo, lo cual resulta útil para el modo 1 o el 2.

En ficheros abiertos con o_zread, sólo es válido el modo 1 con valores positivos en posición.

En ficheros abiertos con o_write, ninguna opción es válida y el fichero entero debe ser escrito secuencialmente.

int ftell (int *fichero*)

Parámetros:

fichero Identificador de un fichero abierto con fopen

Funcionamiento: Devuelve la posición actual del puntero de lectura/escritura, en número de bytes relativo al comienzo del fichero.

int flength (int *fichero*)

Parámetros:

fichero Identificador de un fichero abierto con fopen

Funcionamiento: Devuelve el tamaño total, en bytes, de un fichero abierto.
Esta función no es válida para ficheros abiertos con o_zwrite o o_zread.

int fputs (int *fichero*, string *cadena*)

Parámetros:

fichero Identificador de un fichero abierto con fopen

cadena Texto a escribir en el fichero

Funcionamiento: Escribe una cadena de texto en un fichero, añadiendo un salto de línea al final de la línea. Dicha cadena no se escribe tal cual, sino que determinados caracteres de control son precedidos del caracter de escape "\". Esto se hace así para permitir guardar cadenas en disco que incluyan el caracter de salto de línea "^", pero de manera que posteriormente una sola lectura mediante FGETS permita recuperar la cadena completa. Puede emplearse la función fputs para crear ficheros de texto legibles y editables por personas, como ficheros de configuración. El fichero debe haber sido abierto en un modo de escritura.

string fgets (int *fichero*)

Parámetros:

fichero Identificador de un fichero abierto con fopen

Funcionamiento: Lee una línea de texto de un fichero, y devuelve su contenido en una cadena. Sólo se leen líneas de hasta un máximo de 1024 caracteres. El caracter "\" se interpreta especialmente:

\n	Se interpreta como un salto de línea
\\	Se interpreta como un sólo caracter \
\	A final de línea, une la siguiente línea con la actual
\?	Cualquier otro caracter se deja tal cual (eliminando el \)

Esto permite almacenar cadenas mediante fputs de manera que luego puedan recuperarse tal cual mediante fgets. El fichero debe haber sido abierto en modo lectura.

int feof(int *fichero*)

Parámetros:

fichero Identificador de un fichero abierto con fopen

Funcionamiento: Devuelve 1 (CIERTO) si el puntero de lectura/escritura se encuentra al final del fichero.

string file (string *fichero*)

Parámetros:

fichero Nombre de un fichero

Funcionamiento: Esta función abre un fichero y recupera su contenido al completo en una cadena. Para funcionar correctamente, sin embargo, dicho fichero no podrá contener ningún byte a 0 (por lo que esta función resulta de utilidad con ficheros de texto principalmente).

save (string *fichero*, pointer *datos*, int *ints*)

Parámetros:

fichero Nombre de fichero a grabar

datos Puntero a la estructura o zona de datos a grabar

ints Tamaño en enteros de la zona de datos a grabar

Funcionamiento: Esta función almacena en un fichero una zona de datos. Al contrario que fwrite, no recibe como parámetro el número de bytes a escribir, sino el número de enteros (cada uno de 4 bytes) que ocupa. Se puede obtener este dato dividiendo por 4 el número de bytes obtenido con el operador sizeof.

Esta función se considera obsoleta, y su uso puede resultar problemático especialmente si se desean almacenar en fichero estructuras que contengan datos tipo byte o word.

La estructura o zona de datos a grabar no debe contener ningún dato tipo pointer ni string.

load (string *fichero*, pointer *datos*, int *ints*)

Parámetros:

fichero Nombre de fichero a recuperar

datos Puntero a la estructura o zona de datos a recuperar

ints Tamaño en enteros de la zona de datos a recuperar

Funcionamiento: Esta función recupera de un fichero previamente grabado con la función save, una zona de datos. Los parámetros *datos* e *ints* deben ser los mismos que recibió la función save a la hora de almacenar el fichero.

5.21 Cadenas

int len (string *cadena*)

Parámetros:

cadena Una cadena de texto o variable

Funcionamiento: Esta función devuelve la longitud (el número de caracteres) de una cadena.

string ucase (string *cadena*)

Parámetros:

cadena Una cadena de texto o variable

Funcionamiento: Devuelve una copia en mayúsculas de la cadena que recibe. Se pasan también a mayúsculas los caracteres acentuados y otros como Ñ o Ç. Los acentos se pierden ("á" pasa a ser "A") debido a limitaciones del juego de caracteres MS-DOS, utilizado internamente por Fenix. Para pasar una variable a mayúsculas es preciso hacer:

```
variable = ucase(variable);
```

string lcase (string *cadena*)

Parámetros:

cadena Una cadena de texto o variable

Funcionamiento: Devuelve una copia en minúsculas de la cadena que recibe. Para pasar una variable a minúsculas es preciso hacer:

```
variable = lcase(variable);
```

string substr (string *cadena*, int *inicio*, int *final*)

Parámetros:

cadena Una cadena de texto o variable

inicio Número de carácter inicial

final Número de carácter final

Funcionamiento: Devuelve una sub-cadena, es decir, un fragmento de la cadena que recibe, ubicado entre los caracteres "inicio" y "final", inclusive.

Si uno de los dos caracteres es menor de cero, se considera la posición a partir de la derecha de la cadena. Por ejemplo, la posición -1 corresponde al último caracter de la cadena, -2 al penúltimo, etc.

Si el inicio es mayor que el final, se intercambian sus posiciones.

substr devuelve una cadena nueva, no altera la original en el caso de que ésta se trate de una variable.

int find (string *cadena*, string *buscar*)

Parámetros:

cadena Una cadena de texto o variable

buscar La cadena o texto a buscar

Funcionamiento: Busca una subcadena dentro de otra, y devuelve el número de caracter (0 corresponde al primero, empezando por la izquierda) de la primera aparición de la subcadena.

En el caso de que la subcadena no se encuentre dentro de la cadena principal, la función devolverá -1.

string itoa (int *número*)

Parámetros:

número Un número entero que se desea convertir a cadena

Funcionamiento: Devuelve una cadena, a partir de un número. Por ejemplo, itoa(10) devuelve la cadena "10". Se utiliza para hacer conversiones de datos y poder así mostrar valores numéricos en los textos.

Hay que hacer notar que Fenix utiliza automáticamente ITOA cuando se trata de sumar un número a una cadena (el orden es importante: el número debe estar a la derecha de la suma). Eso permite una mayor flexibilidad a la hora de escribir textos con funciones como write:

```
write (0, 160, 100, 4, "Posición: " + x + ", " + y);
```

string ftoa (float *número*)

Parámetros:

número Un número en coma flotante que se desea convertir a cadena

Funcionamiento: Devuelve una cadena, a partir de un número en coma flotante. La cadena contendrá el mínimo número posible de decimales, por lo que ftoa(2.0) devolverá "2" y no "2.0". Así mismo, ftoa(2.2500) devolverá la cadena "2.25".

int atoi (string *cadena*)

Parámetros:

cadena Una cadena de texto o variable que contiene un número

Funcionamiento: Convierte una cadena a un valor numérico entero. Por ejemplo, si la cadena contiene "10", atoi devolverá el número 10. atoi nunca detendrá el programa con un error. Una cadena no válida se interpretará como el número 0 por atoi.

float atof (string *cadena*)

Parámetros:

cadena Una cadena de texto o variable que contiene un número

Funcionamiento: Convierte una cadena a un valor numérico con decimales. Por ejemplo, si la cadena contiene "10.5", atof devolverá el número 10.5. atof nunca detendrá el programa con un error. Una cadena no válida se interpretará como el número 0 por atof.

byte asc (string *cadena*)

Parámetros:

cadena Una cadena de texto o variable

Funcionamiento: Devuelve el código ASCII (0-255) del primer carácter de la cadena. Por ejemplo, asc(" ") devuelve el número 32. Fenix utiliza internamente el juego de caracteres IBM 437 de MS-DOS. Si la cadena está en blanco, asc devolverá 0.

string chr (byte *número*)

Parámetros:

número Un número de carácter

Funcionamiento: Devuelve una cadena con un sólo carácter: aquél representado por el código ACII que se le pase como parámetro a chr. Por ejemplo, chr(32) devuelve " ". Si el número fuera 0, chr devuelve una cadena en blanco ("").

5.22 Memoria dinámica

pointer alloc (int *bytes*)

Parámetros:

bytes Tamaño en bytes de la zona de memoria a reservar

Funcionamiento: Reserva una zona de memoria, y devuelve un puntero a la misma. Esta es una función avanzada, que requiere dominio del concepto de punteros. Dicha zona puede contener restos de un uso anterior, o datos aleatorios. Es responsabilidad del programador inicializarla.

El puntero devuelto por alloc puede asignarse a cualquier variable de tipo puntero (como un byte pointer o un int pointer). Se aconseja guardar el puntero en una variable global, y acceder a la memoria reservada usando el operador [] a lo largo del programa.

Si no quedara memoria disponible suficiente como para reservar el espacio requerido, alloc generará un error en tiempo de ejecución. Sin embargo es un problema raro, dado que el sistema operativo puede emplear el espacio libre en disco duro para emular la existencia de memoria.

El número de elementos que cabrán en la nueva zona de memoria equivaldrá al número de bytes, dividido por el tamaño de cada elemento. Si queremos reservar memoria para guardar 1000 datos de tipo "byte", debemos reservar 1000 bytes, mientras que si deseamos guardar 1000 datos de tipo int, será preciso reservar 4000 bytes (dado que sizeof(int) es 4).

El puntero devuelto por alloc no debe descartarse, ya que es preciso utilizarlo más adelante al llamar a la función free, para liberar la memoria ocupada.

free (pointer *memoria*)

Parámetros:

memoria Puntero a una zona de memoria, devuelto por una llamada anterior a alloc

Funcionamiento: Libera una zona de memoria reservada previamente por la función alloc. El parámetro pasado a free debe ser exactamente el mismo puntero devuelto por alloc. Una vez liberada, la memoria queda libre para posteriores usos.

pointer realloc (pointer *memoria*, int *bytes*)

Parámetros:

memoria Puntero a una zona de memoria, devuelto por una llamada anterior a alloc

bytes Tamaño en bytes deseado para la zona de memoria

Funcionamiento: realloc amplía o reduce el espacio reservado en un bloque de memoria. Para ello recibe dos parámetros: el primero, debe ser el puntero devuelto por alloc al reservar el bloque; el segundo, será el tamaño completo del nuevo bloque de memoria.

Quizá sea necesario cambiar el bloque de lugar en la memoria, cosa que hará realloc automáticamente. realloc devuelve un puntero al comienzo del nuevo bloque de memoria: este puntero debe usarse a partir de entonces en lugar del devuelto originalmente por alloc, que deja de ser válido.

memset (pointer *memoria*, byte *valor*, int *bytes*)

Parámetros:

memoria	Puntero a una zona de memoria
valor	Valor al cual se desea asignar todos los bytes de la zona indicada
bytes	Tamaño en bytes de la zona de memoria

Funcionamiento: Esta es una función avanzada, que rellena una zona de memoria de "len" bytes de longitud a partir del valor asignado. Su mayor utilidad consiste en rellenar bloques de memoria reservados con alloc (ya que pueden contener datos arbitrarios), aunque también puede resultar útil para modificar el contenido de gráficos de 8 bits (véase la función map_buffer).

memsetw (pointer *memoria*, word *valor*, int *words*)

Parámetros:

memoria	Puntero a una zona de memoria
valor	Valor al cual se desea asignar a todas las palabras de la zona
words	Tamaño en palabras (1 palabra = 2 bytes) de la zona de memoria

Funcionamiento: Esta es una función avanzada, que rellena una zona de memoria de "len" palabras de longitud a partir del valor asignado. Su mayor utilidad consiste en modificar el contenido de gráficos de16 bits con un color devuelto por rgb (véase la función map_buffer). Hay que hacer notar que se están rellenando "len*2" bytes de memoria, ya que esta función rellena palabras, no bytes.

memcopy (pointer *destino*, pointer *origen*, int *bytes*)

Parámetros:

destino	Puntero a la zona de memoria de destino
origen	Puntero a la zona de memoria de origen
bytes	Tamaño en bytes de la cantidad de memoria a copiar

Funcionamiento: Esta función copia *bytes* bytes de memoria desde la posición *origen* a la posición *destino*. Su mayor utilidad reside en la gestión de bloques de memoria, aunque también puede ser útil para copiar grandes trozos de memoria.

No debe utilizarse ninguna función de acceso a memoria (memset o memcopy) para modificar zonas de memoria que contengan variables tipo string (por ejemplo, haciendo copias de estructuras que contengan cadenas) ya que ello puede provocar pérdidas de memoria e incluso cuelgues, ya que Fenix podría descartar la memoria empleada por una cadena cuando resulta que ha sido utilizada en otro lugar debido a una copia de memoria.

5.23 Sonido (muestras WAV)

int load_pcm (string *fichero*, int *repetir*)

int load_wav (string *fichero*, int *repetir*)

Parámetros:

fichero Nombre del fichero .WAV a cargar

repetir Indica opciones para la reproducción de la muestra de sonido. Puede ser 0 para reproducir la muestra normalmente y sin repetición, o uno de los valores de la siguiente tabla:

1	Repetir contínuamente
3	Repetir contínuamente, en modo ping-pong
4	Reproducir al revés
5	Repetir contínuamente, al revés
7	Igual que 3, pero la primera vez es al revés

Funcionamiento: Esta función recupera de disco un fichero PCM o WAV y lo almacena en memoria. Es preciso emplear la función unload_pcm para descargarlo cuando ya no sea preciso utilizarlo. La función devuelve un identificador que es práctica común almacenar en alguna variable global, ya que es preciso emplearlo para identificar la muestra de sonido a la hora de reproducirla.

El modo "ping-pong" indica que la muestra se reproducirá en primer lugar hacia adelante, y seguidamente en dirección inversa, y vuelta a empezar.

A la hora de hacer la primera llamada a la función load_pcm, el sonido se inicializa. Es posible controlar parámetros como stereo o calidad de la reproducción si se modifican las variables globales correspondientes antes de cargar el primer sonido de disco.

load_wav es un sinónimo de load_pcm.

int unload_pcm (int *muestra*)

Parámetros:

muestra Identificador de la muestra de sonido devuelto por load_pcm

Funcionamiento: Esta función descarga de memoria una muestra de sonido recuperada con la función load_pcm. No es válido descargar de memoria una muestra de sonido que se esté reproduciendo actualmente.

int sound (int *muestra*, int *volumen*, int *frecuencia*)

Parámetros:

muestra Identificador de la muestra de sonido devuelto por load_pcm

volumen Cantidad de volumen, entre 0 (silencio) y 63 (volumen máximo)

frecuencia Permite alterar la frecuencia de reproducción: 256 para la frecuencia original.

Funcionamiento: Esta función reproduce una muestra de sonido, y devuelve un identificador del canal por el que la muestra se está reproduciendo (que resulta útil a la hora de detener sonidos contínuos). Además, la función permite especificar el volumen y una variación de la frecuencia de reproducción.

Existe un límite de 32 muestras de sonido reproducibles a la vez.

stop_sound (int *voz*)

Parámetros:

voz Número de canal devuelto por sound

Funcionamiento: Esta función detiene la reproducción de una muestra de sonido, a partir del identificador del canal donde se está reproduciendo, devuelto por la función sound a la hora de reproducirla.

change_sound (int *voz*, int *volumen*, int *frecuencia*)

Parámetros:

voz Número de canal devuelto por sound

volumen Cantidad de volumen, entre 0 (silencio) y 63 (volumen máximo)

frecuencia Permite alterar la frecuencia de reproducción: 256 para la frecuencia original.

Funcionamiento: Esta función altera los parámetros de una muestra de sonido que se esté reproduciendo actualmente, a partir del identificador del canal donde se está reproduciendo, devuelto por la función sound a la hora de reproducirla. Los cambios son audibles inmediatamente.

5.24 Sonido de CD

play_cd (int *pista*, int *contínuo*)

Parámetros:

pista Número de pista del CD a reproducir (empezando por 0 para la primera pista)

contínuo 1 para reproducir la pista contínuamente, 0 para reproducirla una sola vez

Funcionamiento: Inicia la reproducción de una pista de audio del lector de CD principal del sistema. Dicha pista puede reproducirle una sola vez (la función is_playing_cd puede emplearse para averiguar cuándo acabó la reproducción) o bien contínuamente, en función del segundo parámetro.

int is_playing_cd ()

Funcionamiento: Devuelve 1 si el CD está actualmente reproduciendo alguna pista, o 0 si cualquier reproducción anterior ya ha finalizado.

stop_cd ()

Funcionamiento: Detiene la reproducción de cualquier pista de CD que esté actualmente en marcha.

5.25 Sonido (módulos de sonido MOD)

int load_mod (string *fichero*)

Parámetros:

fichero Nombre del fichero a cargar

Funcionamiento: Esta función recupera un módulo de disco y lo almacena en memoria, listo para reproducir en cualquier momento. La función devuelve un identificador que debe ser empleado posteriormente en cualquier llamada a las demás funciones de sonido de esta sección.

Fenix soporta módulos de sonido en formato MOD, S3M, XM e IT.

A la hora de hacer la primera llamada a la función load_mod, el sonido se inicializa. Es posible controlar parámetros como stereo o calidad de la reproducción si se modifican las variables globales correspondientes antes de cargar el primer sonido de disco.

unload_mod (int *módulo*)

Parámetros:

módulo Identificador del módulo de sonido, devuelto por load_mod

Funcionamiento: Esta función libera la memoria ocupada por un módulo de sonido. Es responsabilidad del programador asegurarse de que este módulo no está siendo reproducido en el momento de llamar a unload_mod.

play_mod (int *módulo*)

Parámetros:

módulo Identificador del módulo de sonido, devuelto por load_mod

Funcionamiento: Inicia la reproducción de un módulo de sonido. Si ese mismo módulo de sonido ya se estuviera reproduciendo, reinicializa la reproducción desde el comienzo del mismo. Si otro módulo de sonido diferente estuviera sonando, se detendrá su reproducción y se iniciará la del indicado.

Al acabar el módulo, éste se detendrá automáticamente. Para evitarlo, el propio módulo de sonido deberá incorporar los comandos adecuados para continuar desde la primera u otra pista al llegar al final.

stop_mod ()

Funcionamiento: Detiene la reproducción de cualquier módulo de sonido que se esté reproduciendo.

int mod_active ()

Funcionamiento: Devuelve 1 si hay algún módulo de sonido reproduciéndose en el momento de llamarla, o 0 si no fuera el caso.

int mod_get (int *tipo*)

Parámetros:

tipo Indica el tipo de parámetro que se desea recuperar. Puede ser una de las constantes predefinidas siguientes:

MOD_PAT	Número de patrón actual
MOD_VOLUME	Volumen global actual de la canción
MOD_POS	Número de posición
MOD_NUMPOS	Número total de posiciones de que consta la canción
MOD_TIME	Número de segundos transcurridos desde el comienzo

Funcionamiento: Esta función recupera información (un número entero) sobre el módulo de sonido que se esté reproduciendo actualmente. Puede emplearse para controlar el avance del módulo (mediante el parámetro mod_pos) y sincronizar acciones del juego con la música.

string mod_info (int *tipo*)

Parámetros

tipo Indica el tipo de parámetro que se desea recuperar. Puede ser una de las constantes predefinidas siguientes:

MOD_TYPE	Descripción del tipo de mod
MOD_NAME	Nombre de la canción

Funcionamiento: Esta función recupera información (una cadena descriptiva) sobre el módulo de sonido que se esté reproduciendo actualmente. La cadena suele tener un ancho máximo de 60 caracteres.

int mod_set (int *tipo*, int *valor*)

Parámetros:

tipo Indica el tipo de parámetro que se desea modificar. Puede ser una de las constantes predefinidas siguientes:

MOD_VOLUME	Volumen global actual de la canción
MOD_POS	Número de posición

Funcionamiento: Esta función modifica sobre la marcha el estado de reproducción del módulo de sonido que esté en marcha actualmente. Puede emplearse para forzar saltos en la reproducción, o para modificar el volúmen de la canción sin afectar a los efectos de sonido.

5.26 Depurado

say (string *texto*)

Parámetros:

texto Un texto cualquiera

Funcionamiento: Esta función muestra en la consola una línea con el texto indicado. En la versión Windows de Fenix no existe por ahora el concepto de consola, por lo que estos textos se almacenan en un fichero de nombre STDOUT.TXT.

A Uso de los programas desde la línea de comandos

A.1 Compilador FXC

El compilador FXC transforma un fichero de código fuente Fenix, normalmente de extensión .PRG, a un fichero de datos (en un formato interno similar al código máquina) de extensión DCB ejecutable por el intérprete.

Su ejecución es trivial y consiste simplemente en

```
FXC fichero.prg
```

Con lo cual FXC generará un fichero de datos de nombre *fichero*.dcb. No es necesario especificar la extensión del PRG, si esta es PRG. Si el código fuente contuviese algún error de sintaxis, FXC se detendrá inmediatamente informando de la línea del error y no continuará compilando el resto del fichero ni generará el DCB.

Existen algunas opciones que pueden escribirse antes o después del fichero indistintamente:

-i *directorio* Con esta opción, FXC añadirá el directorio especificado al *path*, una lista interna de directorios que el compilador mantiene y en los que buscará los ficheros incluídos mediante la instrucción INCLUDE.

-g Con esta opción, FXC almacenerá en el fichero DCB información adicional innecesaria para ejecutarlo, como los nombres de las variables y la posición de las instrucciones en los ficheros de código fuente. Esta información será necesaria para emplear el futuro depurador.

-c FXC interpreta normalmente los ficheros de código fuente como textos ASCII que emplean el juego de caracteres ISO8859-1, empleado en Windows y Linux indistintamente. Con esta opción FXC los interpretará como ficheros de texto bajo el juego de caracteres DOS CP 437. Emplea esta opción si quieres compilar ficheros escritos en editores de texto de MS-DOS.

-a Con esta opción FXC tratará de buscar cualquier fichero empleado en el código fuente y añadirlo al DCB, de manera que el intérprete los podrá coger de ahí automáticamente. Es posible que FXC incluya ficheros de configuración y otras cosas que deben permanecer externas, así que esta opción debe emplearse con cuidado, o preferiblemente dejar los ficheros fuera del DCB.

A.2 Intérprete FXI

El intérprete se encarga de ejecutar los ficheros DCB generados por el compilador. Para ejecutarlo basta con escribir

```
FXI fichero.DCB
```

No es necesario indicar la extensión si ésta es DCB. FXI permite emplear varias opciones que alteran la forma en que se ejecutará el programa:

-w Esta opción ejecuta el programa en una ventana, en lugar de a pantalla completa. Se aconseja su uso durante la etapa de desarrollo, ya que un error grave a pantalla completa puede tener terribles consecuencias, especialmente bajo DirectX.

-f En modos de 16 bits, habilita un sencillo filtrado que suaviza los bordes de los gráficos mezclando los colores entre ellos, a costa de perder claridad en la visualización. Esta opción es experimental y puede desaparecer en el futuro.

-b Habilita el modo de doble buffer. con lo que Fenix emplea una copia de pantalla en memoria en lugar de acceder directamente a la VGA. Esta opción puede ser más compatible en algunos casos, pero también mucho más lenta.

Obteniendo un EXE a medida Probablemente no desees que los usuarios de tus juegos deban emplear un fichero BAT o un mecanismo similar para ejecutar tu programa en formato DCB y desees emplear un EXE a medida. Como alternativa, FXI puede renombrarse al mismo nombre que tu juego. Si FXI comprueba que su propio nombre no es FXI, buscará un fichero DCB con el mismo nombre que él y no aceptará parámetros en la línea de comandos. Además puedes renombrar el fichero .DCB a .DAT y será localizado igualmente.

A.3 Utilidad MAP

Los ficheros MAP constituyen el centro de atención de los programas en Fenix. Un fichero MAP contiene un gráfico en 8 bits con paleta de colores, o bien un gráfico de 16 bits sin ella.

Al contrario que otros formatos gráficos como GIF, PNG ó JPEG, el formato MAP no comprime los datos del gráfico, por lo que resulta más rápido cargarlos en memoria, a costa de ocupar más espacio en disco.

Además de los datos en sí del gráfico, un fichero MAP contiene:

- Un nombre breve, describiendo el gráfico (32 caracteres)
- Hasta 1000 puntos de control. Cada punto de control indica un punto en concreto dentro del gráfico. El punto de control 0 indica el centro del gráfico, mientras el resto queda disponible para el usuario, que puede obtener esta información de cada gráfico mediante las funciones GET_POINT y GET_REAL_POINT.
- Una tabla de animación. El map puede contener una animación, en lugar de un único gráfico, como se detalla posteriormente en el comando +animate

Aconsejo crear los gráficos en formato PNG, y convertirlos posteriormente formato MAP añadiendo puntos de control y animaciones mediante esta utilidad. Actualmente la utilidad MAP almacena los ficheros en formato comprimido, por lo que no es posible leer estos ficheros desde DIV.

Uso de la utilidad MAP Básicamente sirve para editar un fichero MAP o crearlo a partir de otro. Tiene los siguientes modos de funcionamiento:

- Listar información sobre un fichero

  ```
  map -l fichero.map
  ```

  Muestra información del fichero MAP (nombre, dimensiones, bits, y puntos de control). Es la opción por defecto.

- Extraer la paleta de colores

  ```
  map -p fichero.map
  ```

  Extrae la paleta de colores del fichero MAP (no es válido en MAPs de 16 bits). Le da el mismo nombre que al fichero original, y extensión .PAL

- Convertir MAPs de 8 a 16 bits

  ```
  map -c fichero.map
  ```

  Convierte un fichero MAP de 8 bits, a 16 bits. Sobreescribe el fichero original.

- Convertir MAPs a PNG

  ```
  map -g fichero.map
  ```

Convierte un fichero MAP (tanto de 8 como de 16 bits) a PNG. Crea un fichero de mismo nombre y extensión .PNG

- Convertir PNGs a MAP

  ```
  map -m fichero.png
  ```

  Crea un fichero MAP (tanto de 8 como de 16 bits) a partir de un PNG. Crea un fichero de mismo nombre que el original y extensión .MAP

Todas las opciones admiten más de un fichero en la línea de comandos. Además, las opciones -l, -c y -m permiten editar mediante comandos adicionales, los puntos de control y el nombre del mapa. Estos comandos tienen el siguiente formato:

- Asignación de nombre

  ```
  +name=nombre
  ```

  Asigna un nuevo nombre al fichero MAP. Es preciso escribir el nombre entre comillas dobles si éste tiene espacios.

- Asignación de centro

  ```
  +center=x,y
  ```

  Asigna la posición x,y (donde x e y son números enteros) como el centro del gráfico. Usar "+center=" para eliminar el centro de la tabla de puntos de control.

- Asignación de punto de control

  ```
  +n=x,y
  ```

  Asigna un punto de control. "n" es un número entre 0 y 999. El comando funciona igual que el anterior.

- Asignación de una secuencia de animación

  ```
  +animation=a,b,c,d,...
  ```

  Establece una lista de animación para el MAP. Al establecer este parámetro, esta utilidad divide el gráfico en una serie de frames contiguos de izquierda a derecha, y esta lista especifica el orden en el que deben ir mostrándose en pantalla (el 1 corresponde a la imagen situada más a la izquierda).

  Para poder crear una animación automática en un MAP, es preciso crear un gráfico PNG con una utilidad externa y dividirlo con líneas verticales en tantas partes iguales como "frames" queramos que tenga la animación. A la hora de convertir el PNG en un MAP animado, es preciso indicar este parámetro.

  El número de divisiones se extrae del número más grande de la lista. Algunas listas de animación válidas para gráficos de 4 frames son:

  ```
  +animation=1,2,3,4
  +animation=1,2,3,4,3,2
  +animation=1,1,1,1,2,3,4
  +animation=1,1,1,1,2,2,2,3,3,4
  ```

  etc.

- Asignación de una velocidad a la animación

`+speed=#`

Especifica la velocidad en milisegundos por frame de un fichero MAP animado. No tiene efecto sobre MAPs no animados.

En el caso de que se estén listando, creando o convirtiendo múltiples ficheros MAP, estos comandos afectarían a todos por igual.

A.4 Utilidad FPG

Los ficheros FPG son colecciones de gráficos "bitmap" que han sido diseñados para mostrarse a la vez en pantalla. A cada uno de estos gráficos se le asigna un código numérico (entre 1 y 999) que sirve para identificarlo dentro de la colección. Fenix permite cargar múltiples ficheros FPG a la vez, mediante la función load_fpg. Esta función carga en memoria el contenido entero del FPG, y a partir de entonces es posible acceder a cualquiera de sus gráficos individualmente, a través del código numérico del mismo y además, si hay más de un FPG a la vez en memoria, usando también el código de colección devuelto por la función LOAD_FPG. Por ejemplo:

```
GLOBAL
    int fpg_1, fpg_2 ;
BEGIN
    fpg_1 = load_fpg ("pantallas.fpg") ;
    fpg_2 = load_fpg ("varios.fpg") ;

    /* Pone como fondo el gráfico 1 del primer fpg */
    put_screen (fpg_1, 1) ;

    /* Elige como gráfico del ratón el gráfico 200 del segundo */
    mouse.file = fpg_2 ;
    mouse.graph = 200 ;
END
```

Existen dos tipos de ficheros FPG. El comportamiento es el mismo, sólo hay ciertas variaciones internas en el formato:

FPG FPG, que es capaz de convertir un FPG de 8 bits a 16 bits, o bien de crear un nuevo FPG de 16 bits a partir de gráficos PNG o MAP.

Uso de la utilidad FPG FPG es una utilidad ejecutable desde la línea de comandos. Debido a las múltiples posibilidades, explicaré su funcionamiento en función de cada una de las tareas básicas a realizar con ella.

FPG admite opciones en la línea de comandos en la forma "-x", donde x es una letra del abecedario. Estas opciones pueden combinarse, de forma que es lo mismo ejecutar "fpg -lv" que "fpg -l -v".

- Descripción del contenido de un FPG

  ```
  fpg -l fichero.fpg
  fpg -lv fichero.fpg
  ```

  Muestra el contenido de un fichero FPG. El listado incluye el código de cada gráfico, el nombre original del fichero y el nombre del mismo. También se indica su tamaño en pixels.

  La forma adicional añadiendo la opción -v añade además los puntos de control de cada mapa en la lista. Recordemos que el punto de control 0 equivale al centro del gráfico.

  La opción -l es la opción por defecto. "fpg fichero.fpg" equivale al primer ejemplo, mientras "fpg -v fichero.fpg" equivale al segundo.

- Creación de un nuevo FPG de 16 bits

```
fpg -n fichero.fpg
```

Crea un nuevo FPG de 16 bits, de nombre "fichero.fpg". Inicialmente este FPG no contiene ningún bitmap.

```
fpg -n fichero.fpg grafico.map [...]
```

Crea un nuevo fichero FPG de 16 bits y añade al mismo uno o más mapas .MAP. Si dichos mapas fueran de 8 bits, se convierten automáticamente al añadirse. Se trata de conservar el identificador especificado en el fichero, si no estuviese ocupado por un mapa anterior.

Se pueden especificar ficheros PNG en lugar o junto a los MAP. Dado que un fichero PNG no incluye el concepto de "código", se le asignará identificadores del 1 en adelante.

Es posible especificar en lugar del código por defecto, un número de código distinto para el fichero empleando la sintaxis "1:fichero". Por ejemplo:

```
fpg -ñuevo.fpg 100:demo.png
```

crea un fichero nuevo.fpg con un único gráfico de nombre "demo" y de código 100.

- Creación de un nuevo FPG de 8 bits

```
fpg -o fichero.fpg -p paleta.pal
```

Crea un nuevo FPG de 8 bits, de nombre "fichero.fpg". Inicialmente vacío, la paleta incluída estará basada en el fichero "paleta.pal". También se puede extraer la paleta de un fichero MAP o PAL con la opción -p.

```
fpg -o fichero.fpg -p paleta.pal grafico.map [...]
```

Crea un nuevo FPG de 8 bits, y añade los gráficos indicados. No se hacen conversiones con los gráficos: indicados. Extrae la paleta de colores del primer fichero añadido, que debe ser un .MAP en lugar de un PNG.

- Eliminación de gráficos de un FPG

```
fpg -d fichero.fpgñumero [...]
```

Elimina el gráfico cuyo número se indica, del FPG especificado. Pueden indicarse múltiples números separados por comas o espacios, o rangos utilizando la sintaxis a-b, como por ejemplo 1-20.

- Extracción de gráficos de un FPG

```
fpg -x fichero.fpgñumero [...]
```

Extrae los gráficos cuyos números se indica del fichero FPG especificado. Los ficheros son creados según el nombre original que tenían, y en formato .MAP. Al igual que con la opción anterior, pueden indicarse múltiples identificadores y rangos.

```
fpg -e fichero.fpgñumero [...]
```

Equivale a la opción anterior, pero borra los ficheros del FPG después de extraerlos.

- Añadido de gráficos a un FPG

```
fpg -a fichero.fpg grafico.map [...]
```

Añade uno o más gráficos .MAP o .PNG a un fichero FPG. No se pueden añadir gráficos de 16 bits a un FPG de 8 bits. Por otra parte esta opción sigue el mismo funcionamiento que la opción de crear un nuevo FPG, -n o -o.

- Conversión de un FPG de 8 bits a 16 bits

```
fpg -c fichero.fpg
```

Esta opción convierte un fichero FPG de 8 bits, a 16 bits. Todos los gráficos incluídos son convertidos a 16 bits, y la paleta es eliminada del fichero.

- Extracción de la paleta de colores de un FPG

```
fpg -p fichero.fpg
```

Esta opción extrae la paleta de colores de un FPG. El fichero resultante tendrá el mismo nombre que el FPG pero con extensión .PAL en lugar de la original. Sólo los ficheros FPG de 8 bits incluyen paleta de colores.

Índice de Materias