¡Hola! ¿Qué tal están?

En esta primera entrega de la primera parte del tutorial sobre **Diseño e implementación de una interfaz gráfica de usuario** (GUI) utilizando QBasic, comenzaremos a ver cómo podemos conseguir nuestra meta paso a paso, con pequeños incrementos en cada entrega, para posibilitar todos sigan el ritmo y no se pierdan por el camino si no tienen tanta experiencia en programación.

Por supuesto, a medida la GUI progrese y se haga más grande, también asumiré han crecido los conocimientos de los participantes en este emprendimiento, y cada entrega podría hacerse más y más amplia y complicada al no detenernos tanto en detallitos más o menos insignificantes, aun si tampoco será la gran cosa en cuanto al esfuerzo necesario para comprenderla.

Nota: El código mostrado en el tutorial estará comentado hasta un punto que aun si no sabe nada de programación podrá entenderlo y hasta terminará sabiendo.

En todo caso, debo decir esta forma de hacer las cosas no es la única vía para implementar una interfaz gráfica de usuario, ni probablemente tampoco sea la más adecuada desde el punto de vista de una persona con más experiencia, es nada más mi manera de verlo según mis propias limitaciones de conocimientos, como les he mencionado antes, durante la presentación de la idea de comenzar con la elaboración de este texto básico.

Por lo anterior, me gustaría mucho contar con la participación de muchas personas, no importa si son expertos o aficionados (un experto por lo menos sería ideal), para poder contar por lo menos con sus comentarios expresando ideas, críticas, recomendaciones, etc.

En resumen, espero los interesados en este tema (si existiera alguno) tengan a mano todo lo necesario para empezar, tal como les comenté hace relativamente poco tiempo; es decir, un intérprete del BASIC como el QBasic de Microsoft, y un emulador del MSDOS como el DOSBox; a pesar de servirles también una PC ahora obsoleta en donde puedan instalar MSDOS o FreeDOS directamente.

En realidad, y aun cuando no lo había comentado antes por asumir lo sabrían, también podría servirnos Windows 3.0, Windows 95, o Windows 98, dado esos sistemas operativos, a pesar de ser multitarea, no nos impiden acceder al hardware directamente por utilizar el modo virtual 86 del microprocesador 386 para los programas de MSDOS (algo así como un emulador de un 8086, mas proveído directamente por el propio microprocesador de Intel), y lo mismo podría decirse de Windows NT 4.0.

Por último, si pueden conseguir el QB64 podrían hacer todo esto sin necesidad del DOSBox o QBasic, puesto esa implementación de BASIC es compatible con QBasic, y está diseñada para correr en los Windows más modernos de 64 bits, con la ventaja adicional de poder disfrutar de un incremento en la velocidad en tiempo de ejecución del programa, puesto ahora estaría corriendo directamente sobre el hardware de la computadora y sería compilado, en vez de hacerlo sobre un emulador del MSDOS y sobre in intérprete del BASIC.

Nota: En caso de conocer un software diferente a los mencionados de utilidad para nuestro programa sería bueno me lo recomendaran, puesto de todos modos llegará un momento cuando debamos usar más bien QuickBasic 4.5, para poder compilar el código, o como les comenté todo se moverá demasiado lento en un entorno sólo interpretado como QBasic, y más aún si lo corremos sobre un emulador en vez de hacerlo directamente sobre hardware.

Por mi parte, iré desarrollando las secciones siguiendo un método podríamos decir Top-Down, o sea, comenzaré por los elementos más generales como las ventanas y todo lo necesario para su manipulación, y después iré descendiendo poco a poco, y desarrollando distintos controles gráficos más específicos, a pesar de no detenernos tampoco en cada detalle necesario en una GUI de las reales, o esto podría tomarnos años.

En cuanto al código, lo colocaré en un solo archivo debido más a una limitación de QBasic que a mi propio deseo. Es lógico pensar sería más conveniente dividirlo en varios módulos y eso hubiera hecho, y hasta en clases dentro de dichos módulos si el lenguaje BASIC fuera orientado a objetos. Pero como QBasic no me permite ni utilizar una directiva "$INCLUDE", debo conformarme con colocar todo el código en un mismo archivo e intentar hacerlo de todos modos de manera organizada.

La estructura del programa se dividirá por tanto en "secciones", para llamarlo de alguna manera, y así intentaré mantenerlo en orden.

Por su parte, las mencionadas "secciones" serán como sigue:

1. -Constantes
2. -Estructuras
3. -Declaración de subrutinas y funciones
4. -Declaración de variables compartidas o globales
5. -Inicialización de las variables
6. -Código del programa principal
7. -Implementación de las subrutinas y las funciones

En caso de sentirse confundidos con lo anterior, no deben preocuparse, lo entenderán fácilmente nada más de ver el contenido del archivo fuente del programa una vez lo descarguen con el enlace correspondiente al final de este texto; lo único en verdad necesario para comprenderlo es tener un conocimiento más bien básico del lenguaje de programación usado (BASIC), y todavía más importante, no carecer de los deseos de implicarse en la tarea.

Por ahora sólo diré todas esas "secciones" mencionadas dispondrán de comentarios para dividirlas a su vez en las partes dedicadas a las diferentes subsistemas involucrados en el gestor de ventanas; así, como en QBasic no existe el tipo de datos Booleano, como por otro lado tampoco se encuentra en C, verán al comienzo del texto fuente algo como esto:

```
'*******************************************************
'*  Sección de declaración de constantes                *
'*******************************************************

'Constantes comunes
CONST False% = 0
CONST True% = NOT False
```

En forma parecida, todas las demás constantes dispondrán de un comentario indicando a cuál de las partes del código de la GUI le pertenecen; y lo mismo pasará dentro de las demás "secciones", como con la dedicada a las estructuras, a la declaración de variables, etc.

En fin, será mejor comencemos, y deberé hacerlo violando un poco todo eso del Top-Down, dado necesitamos ver las rutinas gráficas necesarias para poder colocar una ventanita en pantalla antes de poder situar una realmente en ella.

En caso de haber utilizado un entorno de desarrollo distinto al QBasic sobre MSDOS, lo normal hubiera sido desarrollar nuestras propias primitivas gráficas según las posibilidades del SO sobre el cual pensáramos utilizar la GUI. Por ser MSDOS un sistema en modo real y monotarea, nos es posible acceder directamente al hardware de video de la computadora, porque en modo real está permitido, y en todo momento un único proceso estaría modificando la presentación en pantalla y no se producirían contratiempos graves debido a esto. En cambio, en un sistema operativo multitarea corriendo en modo protegido no se nos permite acceder directamente al hardware de video, ni a ningún otro hardware (salvo por los Windows comentados antes menos NT), a no ser nuestro código sea corrido en modo kernel o modo supervisor como lo hace el núcleo del sistema. En modo protegido todo debe hacerse usando los servicios proveídos por la API (Application Program Interface por sus siglas en inglés) del sistema en cuestión, el cual a su vez se comunicaría con los dispositivos físicos a través de los controladores de dispositivo correspondientes.

Nota: El hecho de ser MSDOS un sistema operativo en modo real, y por eso permitir acceder al hardware directamente, fue la causa fundamental de utilizarlo para hacer este experimento, además, claro está, de ser un sistema operativo monotarea y no disponer de una interfaz gráfica propia; la idea es poder estudiar cómo se puede programar el hardware de video por cuenta propia cuando se nos presente la oportunidad, aun si en un

comienzo intentaré centrarnos más en el gestor de ventanas propiamente dicho, puesto en un sistema real lo más probable es vernos obligados a utilizar los servicios proveídos por el mismo.

En este caso particular, y debido a las características del tutorial comentadas, utilizaré las primitivas del propio QBasic, o por lo menos la dedicada al trazado de líneas en la pantalla.

De todas formas, como de pronto nos podemos encontrar con una limitación insalvable, enseguida veremos también cómo podemos cubrirnos las espaldas.

En un BASIC, por lo común se utiliza una subrutina como LINE para trazar las líneas, a pesar de servir ésta también para dibujar los bordes de una región rectangular, e incluso una región rectangular con un relleno de un color determinado. En este caso utilizaré LINE para todo esto, o sea, la usaré para las líneas, y para los rectángulos con relleno, dado como imaginarán, tanto las ventanas como muchos otros controles gráficos se trazan usando esas primitivas. Pero para no depender en demasía del uso de esta subrutina más propia del BASIC, y después poder usar un método diferente sin vernos en la necesidad de cambiar todo el código en donde se las ha llamado, vamos a crear las siguientes subrutinas:

```
SUB GLine (x1 AS INTEGER, y1 AS INTEGER, x2 AS INTEGER, y2 AS INTEGER, ColorCode AS INTEGER)
  LINE (x1, y1)-(x2, y2), ColorCode ‘Llamada a la subrutina de BASIC.
END SUB

SUB GLineBF (x1 AS INTEGER, y1 AS INTEGER, x2 AS INTEGER, y2 AS INTEGER, ColorCode AS INTEGER)
   LINE (x1, y1)-(x2, y2), ColorCode, BF ‘Llamada a la subrutina de BASIC.
END SUB
```

El truco anterior nos servirá bastante bien como un mecanismo de caja negra, y así las subrutinas de más alto nivel usarán GLine o GLineBF sin preocuparse para nada del cómo estas subrutinas hacen la tarea demandada; eso significa que si en un momento determinado cambio la forma como se trazan las líneas sobre la pantalla, y utilizo otro método, la subrutina encargada de dibujar una ventana u otro control continuará funcionando como si nada.

Las subrutinas GLine y GLineBF son bastante simples y por eso no necesitan ser explicadas, no obstante, sólo comentaré la primera dibuja una línea entre los puntos dados por las coordenadas (x1, y1) y (x2, y2), y la segunda nos servirá para trazar los rectángulos con relleno de color especificado usando un sistema de coordenadas parecido, salvo por ser ahora un punto el vértice superior izquierdo del rectángulo, y el otro el vértice inferior derecho del mismo.

En todo caso, como para pintar una línea necesitaremos también un color tal como acabo de decir, y como nos resultaría conveniente reunir todos los colores utilizados por el sistema de ventanas y por los controles de la GUI en un solo sitio, en la sección de declaración de las variables deberíamos de crear un arreglo para contener estos valores:

```
DIM SHARED SystemColor(15) AS INTEGER
```

La palabra clave SHARED, por si alguno no lo recuerda, nos permitirá usar el arreglo SystemColor desde dentro de las subrutinas y las funciones sin necesidad de pasarlo como parámetro, o sea, se declara el arreglo SystemColor como un arreglo global a todo el modulo del programa.

En cuanto a los límites del arreglo, se han declarado nada más 16 colores con los índices desde el 0 hasta el 15, dado QBasic nada más soporta 16 colores en un modo de video de “alta resolución” como lo es el modo 12, en el cual dispondremos de unas dimensiones de la pantalla de 640 x 480 pixels (no mucho para los parámetros actuales, mas suficientes para este experimento básico).

Nota: En las primeras Macintosh lanzadas al mercado el 24 de enero de 1984, se disponía de una resolución de 512 x 342 pixels nada más en blanco y negro, y a pesar de todo fueron las primeras microcomputadoras con una interfaz gráfica de usuario en ser comercializadas con éxito, por eso nosotros no podemos estar inconformes con nuestras limitaciones.

En realidad, en los elementos del arreglo SystemColor, no se guardarán tampoco los valores de color propiamente dichos, sino más bien los números de atributos de color; esto es algo así como los índices de la paleta de colores de QBasic, o más bien de las tarjetas de video VGA usada con éste, en donde sí están definidos los valores reales de los colores por sus componentes RGB (Red-Green-Blue según sus siglas en inglés).

En fin, en este momento no vamos a profundizar mucho en este tema, puesto para eso necesitaríamos todo un artículo, nada más lo menciono para que lo tengan en cuenta, y tal vez más adelante si nos metamos de cabeza en ello al vernos obligados a manipular imágenes.

Por último, para terminar con esto de los colores, en la sección de inicialización de las variables pondremos un código de programa como el siguiente para inicializar los elementos del arreglo de colores del sistema antes creado:

```
SystemColor(1) = 0 'El color negro
SystemColor(2) = 7 'El color blanco, o más bien gris claro
SystemColor(3) = 4 'El color rojo
SystemColor(4) = 15 'El color blanco brillante
SystemColor(5) = 8 'El color gris oscuro
SystemColor(6) = 8
```

En el código de programa anterior podemos ver como el código de color 2 de nuestro sistema se refiere en realidad a un atributo de color 7 (color blanco por defecto) de la paleta de colores del QBasic, y como el código de color 3 de nuestro sistema se refiere al atributo de color 4 (color rojo por defecto) de la misma.

Nota: En realidad más adelante utilizaremos constantes en lugar de los valores literales para representar los colores, sin embargo, por el momento no vamos a preocuparnos por eso.

En caso de tener necesidad de ver una representación en pantalla de dicha paleta usada por defecto por QBasic, pueden correr el programa Palette.bas incluido en unión del texto fuente del programa de esta parte del tutorial, cuya salida en pantalla se muestra en la Figura 1.1 (la primera banda de color no se ve puesto es negra como el fondo).

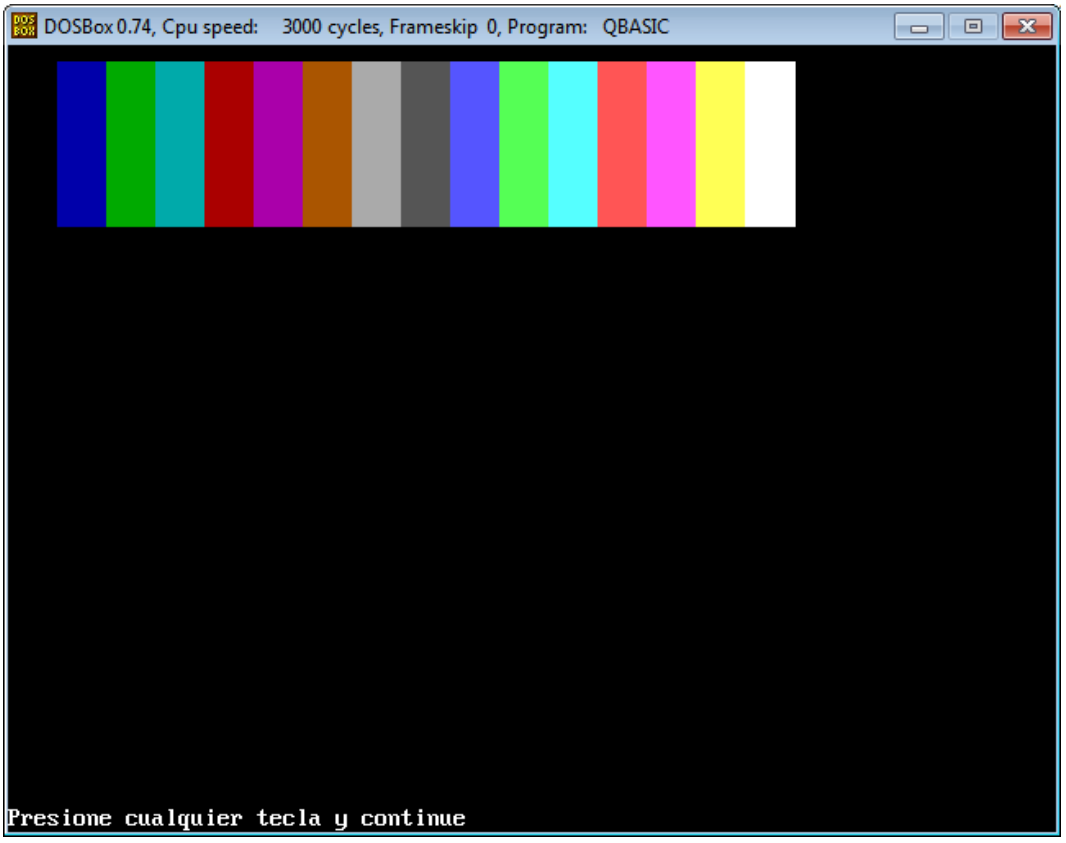

Figura 1.1: La paleta de colores por defecto de VGA (la primera banda no se distingue por ser negra).

Por el momento, como se pueden dar cuenta, no necesitaremos establecer más colores para poder poner una ventana en la pantalla, y por eso nada más se han establecido un subconjunto de colores del gestor de ventanas; incluso no necesitaremos el rojo, y se está utilizando la paleta de colores inicializada por QBasic sin hacerle ninguna modificación, aun cuando esto es posible sin mucho esfuerzo y se podría hacer de tener esa necesidad en algún momento.

En este instante tenemos casi todo lo necesario para poder mostrar una ventana en pantalla, o para mostrar un conjunto de ellas. En apariencia nada más nos iría faltando escribir la función destinada a trazar dichos controles gráficos y listo, puesto tenemos las primitivas gráficas y también hemos establecido los colores del sistema necesarios para ello. Pero todavía nos falta por tocar un punto importante para poder aun si fuera mostrar un par de ventanitas, no digamos para interactuar exitosamente con ellas.

En efecto, por ser las ventanas nada más unos dibujos en la pantalla, se necesita disponer de alguna estructura interna de la GUI para mantener un seguimiento de su estado y de sus propiedades en todo momento; como propiedades comunes podríamos tomar la posición en pantalla de la ventana o de un control gráfico, sus dimensiones, o su color de fondo; y como estado si ésta necesita ser redibujada por un motivo o por otro, o por el contrario esto no es necesario, o si es visible o está oculta a la vista.

La estructura de base para declarar a partir de ella las ventanas podría ser algo como lo siguiente:

```
TYPE WindowType
  Handle AS LONG ‘El manipulador de la ventana
  WindowRect AS RectType ‘El recuadro de la ventana (posición y dimensiones)
  NeedPaint AS INTEGER ‘El estado de dibujado de la ventana
END TYPE
```

En realidad, como se podrán imaginar, esto es una representación bastante simplificada de la estructura encargada de mantener los datos de una ventana de la interfaz gráfica; no obstante, tal y como está nos va a servir bien para la primera aproximación, y más tarde le podremos ir agregando más elementos a medida los necesitemos.

En la estructura anterior podemos ver un elemento de tipo entero largo de nombre Handle, destinado a retener el manipulador de la ventana; este manipulador es nada más un número único para poder identificar en cada momento y sin ambigüedades cada una de las ventanas.

En adición nos encontramos con un elemento de nombre WindowRect y tipo RectType encargado de mantener la posición y las dimensiones de cada ventana; el tipo RectType también deberemos de declararlo para poder usarlo, y lo haremos de la manera siguiente:

```
TYPE RectType
  Left AS INTEGER ‘La posición de la ventana por las abscisas (x)
  Top AS INTEGER ‘La posición de la ventana por las ordenadas (y)
  Right AS INTEGER ‘El ancho de la ventana (Width)
  Bottom AS INTEGER ‘El alto de la ventana (Height)
END TYPE
```

Por último, tenemos en la estructura de las ventanas un elemento de nombre NeedPaint, aun si en realidad en este caso mantendrá un valor de tipo booleano y no un entero como se ha declarado; este elemento existe para contarle al proceso de presentación en pantalla si la ventana en cuestión necesita actualizarse sobre esta, y será cambiado según nos convenga; es decir, si alguna subrutina interna de la GUI modificara algo en una ventana, o un usuario interactuara con ella, el código de programa involucrado debería de poner NeedPaint a True.

La estructura WindowType nos servirá ahora para declarar un arreglo WindowsList en la sección de declaración de las variables de manera podamos usarlo como lista de ventanas para retener la información referente a cada ventana existente en el sistema de gestión de la interfaz gráfica:

```
DIM SHARED WindowsList(MaxWindows) AS WindowType
```

En un entorno distinto podríamos haber utilizado una lista doblemente enlazada para hacer esto, no obstante, como implementar una lista enlazada en QBasic nos podría resultar más costoso a la hora de manipularla, por ahora será más conveniente conformarse con un arreglo.

En todo caso, y como podemos ver en la línea de código de programa mostrada, también necesitaremos una constante MaxWindows y deberemos declararla en la sección dedicada a las constantes como sigue:

```
CONST MaxWindows% = 20
```

La declaración anterior nos limitará a tener sólo un total de 21 ventanas a la vez (los índices del arreglo WindowsList van desde 0 hasta 20), mas por el momento esto también nos será suficiente.

Por último, necesitaremos una variable para poder hacer un seguimiento de los manipuladores de las ventanas, y no asignar un mismo número a dos de ellas; para esto declararemos una variable más en la sección correspondiente como se muestra en la siguiente línea:

```
DIM SHARED WindowsNumber AS LONG
```

Las últimas declaraciones de variables hacen necesario ir a la sección de inicialización de variables del texto fuente del programa e introducir las líneas de código siguientes para inicializarlas:

```
WindowsNumber = 1

DIM i AS INTEGER
FOR i = 0 TO MaxWindows
  WindowsList(i).Handle = -1 'La ventana se declara inoperativa con -1 en su manipulador o Handle
NEXT i
```

Por fin, habiendo hecho todo lo anterior, estamos en condiciones de crear la función CreateWindow, como su nombre lo indica, encargada de crear una ventana:

```
FUNCTION CreateWindow& (Rect AS RectType)
  DIM FreeSlot AS INTEGER
  CreateWindow = -1 'Por defecto CreateWindow devuelve -1, o sea, fracaso
  FreeSlot = LocateFreeSlot 'La función LocateFreeSlot encuentra una posición disponible en el arreglo de ventanas
  IF FreeSlot <> -1 THEN '¿La llamada a LocateFreeSlot resultó exitosa?
    'Las líneas siguientes asignan los valores en la posición de la lista correspondiente a la nueva ventana creada
    WindowsList(FreeSlot).Handle = WindowsNumber
    WindowsList(FreeSlot).WindowRect = Rect
    WindowsList(FreeSlot).NeedPaint = True
    CreateWindow = WindowsNumber
    WindowsNumber = WindowsNumber + 1
  END IF
END FUNCTION
```

La función CreateWindow, como pueden ver, nada más se limitará a llenar un elemento del arreglo WidowsList, usado como lista de ventanas, con la información de la ventana pasados a ella, y luego marcará a la ventana creada como que necesita dibujarse y devolverá un valor entero; este valor puede ser un número consecutivo almacenado en la variable WindowsNumber, y utilizado como el manipulador de la ventana recién creada si la llamada tuvo éxito, o podría resultar en un -1 si el proceso terminó en fracaso (es posible sea más conveniente usar 0 para el fracaso puesto un -1 será tomado como True si ponemos la llamada en una estructura condicional).

Por lo demás, CreateWindow utiliza una función auxiliar de nombre LocateFreeSlot para encontrar una ranura libre dentro del arreglo de la lista de ventanas WindowsList; para hacer esto, dicha función recorre los elementos de WindowsList hasta encontrar uno en donde su propiedad Handle sea igual a -1, lo cual significa ese elemento no se está utilizando (la ventana es inoperativa):

```
FUNCTION LocateFreeSlot%
  DIM i AS INTEGER
  LocateFreeSlot = -1 'La función LocateFreeSlot devuelve -1 por defecto, o sea, fracaso
  FOR i = MaxWindows TO 0 STEP -1 'La lista de ventanas es recorrida en orden inverso
    IF WindowsList(i).Handle = -1 THEN ¿La ventana en la posición "i" está inoperativa?
      LocateFreeSlot = i 'El índice "i" es devuelto por considerarse libre
```

```
    EXIT FOR 'El bucle For es interrumpido puesto no es necesario continuar recorriendo WindowsList
   END IF
  NEXT i
END FUNCTION
```

En la función anterior seguro notaron como el bucle For recorre la lista de ventanas desde el índice más grande hasta el más pequeño, esto es, desde 20 hasta 0 con el valor actual de MaxWindows; esto ahora mismo nos puede causar un cierto problema con el orden de presentación de las ventanas (prueben a crear más de una cuando descarguen el fuente); pero de todas formas, más tarde, cuando agreguemos las correcciones necesarias, nos será de utilidad para esto mismo.

El hecho de haber creado una ventana usando la función CreateWindow, sin embargo, no significa esta se va a ver en pantalla; para lograr esto último debemos mandar a dibujarla, y CreateWindow nada más crea su estructura interna simplificada y marca la ventana como lista para ser trazada.

La siguiente subrutina de nombre DrawWindow es la realmente encargada de realizar el dibujado de la ventana según los colores del sistema establecidos antes, y para esto usa las primitivas para el trazado de líneas y rectángulos:

```
SUB DrawWindow (x1 AS INTEGER, y1 AS INTEGER, x2 AS INTEGER, y2 AS INTEGER)
  GLineBF x1, y1, x2, y2, SystemColor(2)
  GLineBF x1 + 23, y1 + 3, x2 - 3, y1 + 11, SystemColor(4)
  GLine x1 + 1, y1 + 1, x1 + 1, y2 - 2, SystemColor(4)
  GLine x1 + 1, y1 + 1, x2 - 2, y1 + 1, SystemColor(4)
  GLine x1, y2, x2, y2, SystemColor(1)
  GLine x2, y1, x2, y2, SystemColor(1)
  GLine x1 + 1, y2 - 1, x2 - 1, y2 - 1, SystemColor(5)
  GLine x2 - 1, y1 + 1, x2 - 1, y2 - 1, SystemColor(5)
  FOR y% = 3 TO 11 STEP 2
    GLine x1 + 23, y1 + y%, x2 - 3, y1 + y%, SystemColor(5)
  NEXT
  GLine x1 + 3, y1 + 11, x1 + 11, y1 + 11, SystemColor(4)
  GLine x1 + 11, y1 + 3, x1 + 11, y1 + 11, SystemColor(4)
  GLine x1 + 3, y1 + 3, x1 + 11, y1 + 3, SystemColor(5)
  GLine x1 + 3, y1 + 3, x1 + 3, y1 + 11, SystemColor(5)
  GLine x1 + 4, y1 + 4, x1 + 10, y1 + 4, SystemColor(4)
  GLine x1 + 4, y1 + 4, x1 + 4, y1 + 10, SystemColor(4)
  GLine x1 + 4, y1 + 10, x1 + 10, y1 + 10, SystemColor(5)
  GLine x1 + 10, y1 + 4, x1 + 10, y1 + 10, SystemColor(5)
  GLine x1 + 13, y1 + 11, x1 + 21, y1 + 11, SystemColor(4)
  GLine x1 + 21, y1 + 3, x1 + 21, y1 + 11, SystemColor(4)
  GLine x1 + 13, y1 + 3, x1 + 21, y1 + 3, SystemColor(5)
  GLine x1 + 13, y1 + 3, x1 + 13, y1 + 11, SystemColor(5)
  GLine x1 + 14, y1 + 7, x1 + 17, y1 + 7, SystemColor(5)
  GLine x1 + 17, y1 + 4, x1 + 17, y1 + 7, SystemColor(5)
  GLine x1 + 14, y1 + 4, x1 + 17, y1 + 4, SystemColor(4)
  GLine x1 + 14, y1 + 4, x1 + 14, y1 + 7, SystemColor(4)
END SUB
```

La subrutina anterior se limita a dibujar una ventana línea por línea, y puede parecer bastante complicada por tantos numeritos a pesar de su sencillez, sin embargo, no la comento puesto un poco más adelante deberemos modificarla bastante y sería inútil hacerlo.

Por lo demás, DrawWindow a su vez debe ser llamada por alguien o en caso contrario las ventanas definidas nunca se mostrarían en pantalla; esto último lo realiza la subrutina PresentationProcess, o sea, la subrutina encargada del proceso de presentación en pantalla de la interfaz gráfica.

En el listado de código de programa siguiente se expone la subrutina PresentationProcess, y si lo estudian podrán ver como recorre la lista de ventanas (esta vez de principio a fin y no a la inversa como lo hace LocateFreeSlot), y para cada una comprueba sea operativa (Handle <> -1), y en ese caso, si necesita ser dibujada en pantalla (NeedPaint = True):

```
SUB PresentationProcess
  DIM x1 AS INTEGER
  DIM y1 AS INTEGER
  DIM x2 AS INTEGER
  DIM y2 AS INTEGER
```

```
 DIM i AS INTEGER

 FOR i = 0 TO MaxWindows 'La lista de ventanas WindowsList es recorrida
  IF WindowsList(i).Handle <> -1 THEN '¿La ventana en la posición "i" está operativa?
   IF WindowsList(i).NeedPaint THEN '¿La ventana necesita ser dibujada?
    x1 = WindowsList(i).WindowRect.Left 'La posición "x" de la esquina superior izquierda es obtenida
    y1 = WindowsList(i).WindowRect.Top 'La posición "y" de la esquina superior izquierda es obtenida
    x2 = WindowsList(i).WindowRect.Left + WindowsList(i).WindowRect.Right
    y2 = WindowsList(i).WindowRect.Top + WindowsList(i).WindowRect.Bottom
    DrawWindow x1, y1, x2, y2 'La ventana es trazada en la pantalla
    WindowsList(i).NeedPaint = False 'La ventana se marca para no volver a ser dibujada hasta no detectar fue modificada
   END IF
  END IF
 NEXT i
END SUB
```

Por último, la subrutina de presentación es llamada de continuo desde el bucle principal del sistema gestor de ventanas, como se muestra a continuación, dado ella tampoco va a llamarse por sí sola; también se ha utilizado CreateWindow antes de entrar en el bucle para poder crear una ventana de prueba, puesto en caso contrario no se vería nada al correr el programa sino una pantalla oscura:

```
SCREEN 12 'Poner modo de video 12 (640 x 480 pixels con 16 colores)

CONST WindowWidth% = 300 'Ancho de la ventana
CONST WindowHeight% = 200 'Alto de la ventana

DIM Rect AS RectType
DIM PressetKey AS STRING

'Calcula las coordenadas para posicionar la ventana al centro de la pantalla según sus dimensiones y la resolución establecida
Rect.Left = 640 / 2 - WindowWidth / 2
Rect.Top = 480 / 2 - WindowHeight / 2

'Establece el alto y el ancho de la ventana
Rect.Right = WindowWidth
Rect.Bottom = WindowHeight

DIM Handle AS LONG

'Crea la ventana
Handle = CreateWindow(Rect)

DO
 PressetKey = INKEY$
 PresentationProcess
LOOP WHILE PressetKey <> CHR$(27)
```

El bucle anterior, como pueden ver, continuará en tanto no presionemos la tecla escape en nuestro teclado. En cada ciclo PresentationProcess será llamada, y esta subrutina a su vez recorrerá todo el arreglo de ventanas WindowsList en busca de una ventana operativa. En dicho arreglo debería encontrarse con nuestra ventana recién creada antes de entrar al bucle, y por tanto ésta será mostrada sobre la pantalla. En la Figura 1.2 mostrada a continuación pueden ver el resultado de correr nuestro sencillo programa para mostrar una única ventana en pantalla. Pero si disponen de tiempo pueden escribir más código para crear unas cuantas ventanas más, y así ver cómo se desenvuelven en la pantalla en compañía (deberían notar una cierta anomalía debido a lo comentado antes sobre la forma de recorrer la lista de ventanas en LocateFreeSlot).

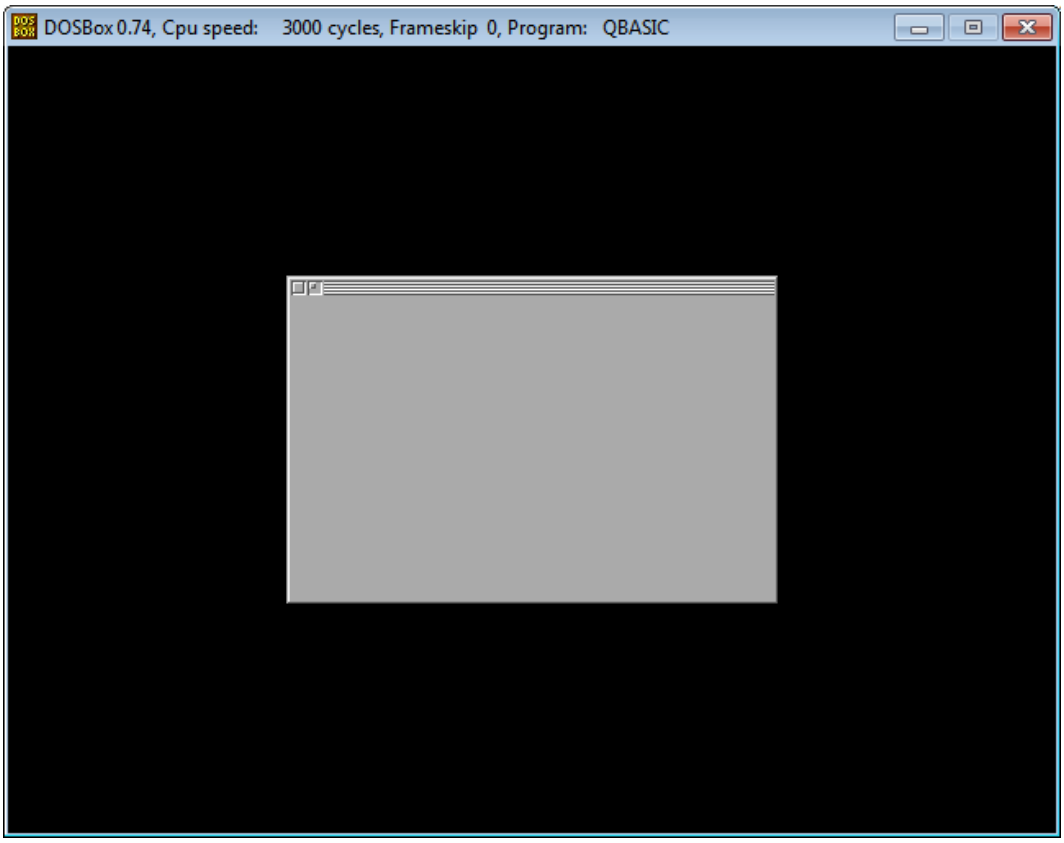

Figura 1.2: El gestor de ventanas mostrando una ventana.

En este momento sólo me resta despedirme de ustedes hasta la próxima, esperando, por supuesto, puedan comentar sus impresiones sobre lo hecho hasta ahora, y ofrecer sus propias ideas, críticas, recomendaciones, etc.

El código fuente de esta entrega del tutorial pueden conseguirlo en Internet con el enlace: Primera.zip

¡Hasta pronto!