CURSO IBM

PROGRAMAR es fácil

9

Borland Delphi

Programación Visual

Multimedia Ediciones, S.A.

Conceptos de programación (9)

Toma de decisiones

La toma de decisiones es una de las tareas cruciales a las que se tiene que enfrentar siempre un programa. Por este motivos, un buen programa debe contemplar, por lo tanto, soluciones para cualquier eventualidad que pueda suceder (o al menos para las previsibles).

La utilización de pseudocódigo puede ser de gran utilidad para crear unos bocetos precisos de lo que deben hacer los programas. A su vez puede estar, en muchos casos, basado fundamentalmente en diagramas de flujo previos. El pseudocódigo, por sus características, es aplicable a cualquier lenguaje de programación, siendo además su uso muy intuitivo para cualquier programador.

MÁS PSEUDOCÓDIGO

En las anteriores unidades nos hemos ocupado de una de las técnicas fundamentales del diseño: el pseudocódigo. Al igual que un arquitecto dibuja planos de sus edificios antes de construirlos, el pseudocódigo permite esquematizar el funcionamiento de un programa expresando sus distintas secciones en términos sencillos, para, posteriormente, ir profundizando gradualmente en los detalles.

Por ejemplo, un diseño en pseudocódigo de un programa de juegos podría ser parecido a esto (se utilizan algunas de las reglas descritas en las unidades anteriores):

```
Empezar
  Preguntar «¿Partida nueva?»
  Si Jugador responde «NO»
    Terminar
  En caso contrario
    Jugar
Terminar
```

Fijémonos en que, como es habitual en esta sección, el pseudocódigo no está escrito en ningún lenguaje de programación en particular, sino que es una idea escrita en lenguaje natural, pero estructurada de forma similar a cómo se hará en el programa final. Este "pseudoprograma" permite analizar la lógica de funcionamiento, ver que vamos a necesitar una serie de elementos (una variable y una instrucción para comparar dos valores, por ejemplo) y, en definitiva, plantear de manera ordenada y coherente cómo vamos a programar. Evidentemente, una descripción tan somera como la anterior, sólo nos hace pasar un primer nivel en las tareas de la programación, pero ya nos clarifica una serie de tareas que tendremos que realizar: por ejemplo, preguntar si se desea jugar una partida nueva, calcular el movimiento del jugador, etc. A partir de este primer pseudocódigo podemos empezar a pensar en los siguientes niveles que debemos acometer. Por ejemplo, vamos a desarrollar el pseudocódigo del bloque que determina si el jugador desea iniciar una partida nueva. Pensando en la reutilización del código, sería interesante que esta sección la codificásemos en forma de función o procedimiento, ya que así podríamos usarla de nuevo en otros programas de juegos parecidos. Como lo que ha

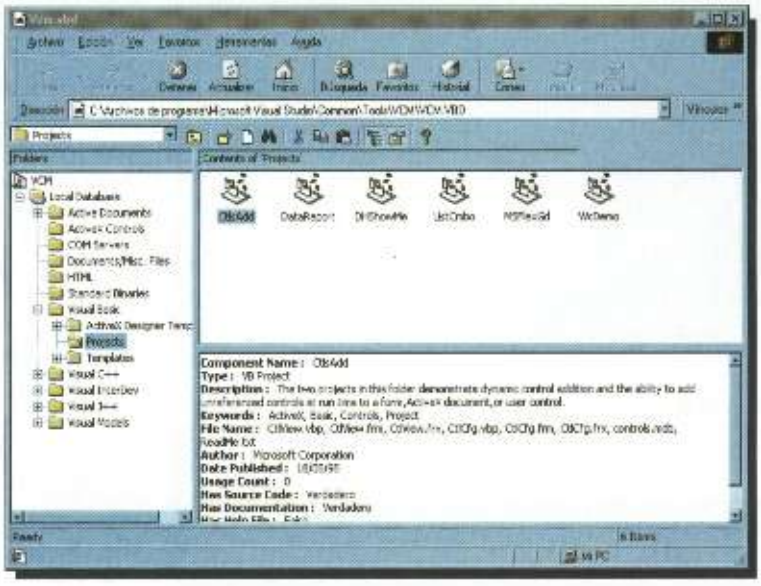


FIG 1. El pseudocódigo debe incluir información lo más detallada posible acerca de cada uno de los módulos a desarrollar.

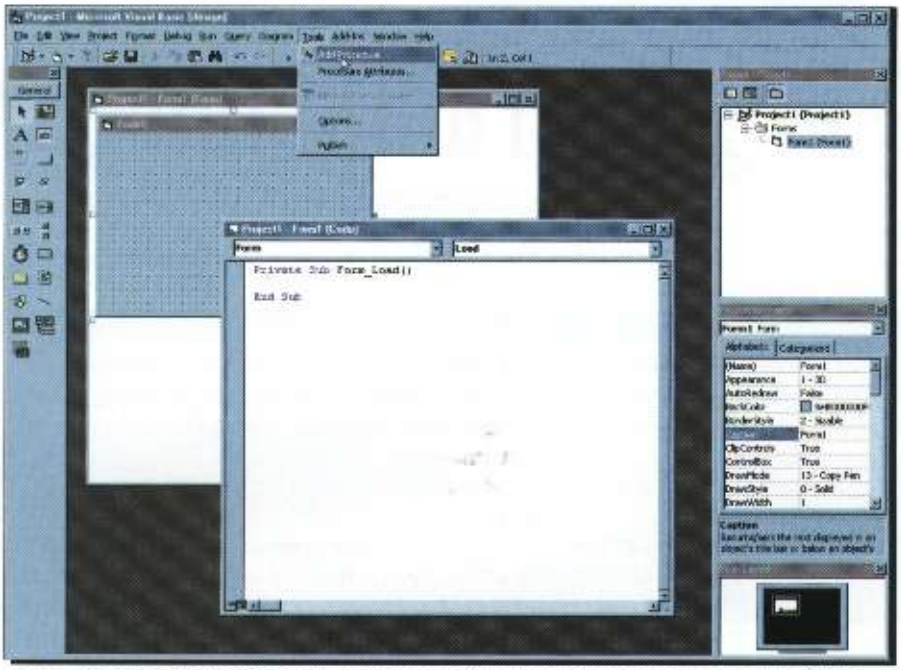


FIG 2. Con los lenguajes de programación visual, pasar de pseudocódigo a código fuente es realmente sencillo.

de hacerse es obtener información del usuario y decirle algo al programa principal, es lógico que desarrollemos una función que devuelva un valor (por ejemplo un número) diciendo sí o no. Es el momento de tomar una decisión de diseño: hay que determinar, en principio arbitrariamente, el nombre de la función, qué tipo de valor ha de devolver y qué sentido debemos dar a cada valor. No existe una metodología para tomar este tipo de decisiones. Son la pericia y la experiencia del diseñador los que determinarán los resultados finales. A veces las opciones elegidas en la fase de diseño se muestran erróneas en fases avanzadas del proyecto, lo que obliga a tirar por la borda el trabajo realizado, corregir una decisión y volver a empezar esta parte. Si esto le ocurre alguna vez, no se preocupe, es normal; hasta el profesional más experimentado no puede evitar que esto le suceda de vez en cuando.

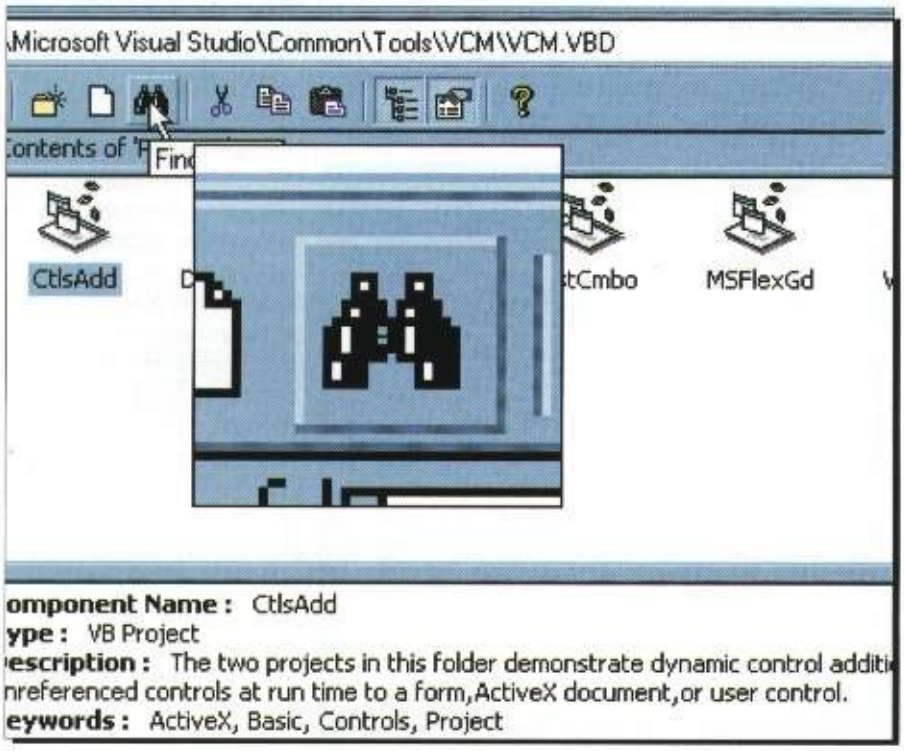


FIG 3. El proceso de definir un programa es iterativo; iremos revelando más detalles al respecto en cada nivel de desarrollo.

Sabiendo cómo funciona nuestro lenguaje de programación particular (por ejemplo, Pascal), el resultado de las decisiones que hemos tomado en la fase de diseño es a menudo obvio. En nuestro caso, optamos por que la función devolverá un número entero, y que el valor 0 significará que el jugador ha contestado que no. Cualquier otro valor significará una respuesta afirmativa. Así, el pseudocódigo para la función podría quedar del siguiente modo:

```
Función
JuegoNuevo(respuesta)
[devuelve un entero]
  Respuesta = 1

Mostrar Texto «¿Empezar la
partida?», con botones «Sí» y
«No»
  Si pulsa «No»
    Respuesta = 0
Fin de Función
```

En el caso de que aún quedasen algunos puntos por detallar, volveríamos a ampliar en un nuevo nivel los bloques que lo requirieran.

El proceso se repite hasta que el programa queda claramente definido y se puede ir ya sin demasiadas dudas al ordenador con el fin de empezar a programar.

En este caso estas instrucciones son lo bastante detalladas como para poder ser traducidas directamente a nuestro lenguaje de programación; fijémonos en que, en este caso intencionadamente, no hemos dicho, por ejemplo, cómo se dibuja un cuadro de diálogo en la pantalla, dejando este detalle para el momento de programar. El conocimiento del lenguaje de programación que manejamos y de las posibilidades y herramientas que ofrece nos indicarán hasta dónde hemos de llegar en la especificación del pseudocódigo.

DECISIONES EQUIVOCADAS

Es fácil cometer errores, hasta en los entornos más cualificados y profesionales. Como muestra, he aquí la siguiente anécdota.

El conocido avión de combate F-16 fue el primero en disponer del llamado sistema de control por cable. En éste, los mandos del piloto van conectados a un ordenador, que interpreta sus instrucciones y genera los movimientos necesarios en los

FIG 4. La aceleración de Coriolis es responsable del giro de las masas de aire en la atmósfera.

elementos de control (alerones, timón, etc.) para completar esta maniobra. Además, el ordenador calcula otras maniobras adicionales de manera automática con el fin de simplificar el trabajo del piloto.

Durante el desarrollo del software de control se hicieron una serie de pruebas en simuladores, con el fin de determinar su correcto funcionamiento. Tras varias pruebas, todo iba perfectamente y sin problemas. Se estaba a punto de pasar el software al primer prototipo real del avión cuando, de pronto, ocurrió algo.

En una sesión de simulación, el piloto del F-16 virtual se encontraba volando al norte del ecuador, supuestamente sobre el mar.

En el curso del vuelo cruzó el ecuador hacia el Sur y, de pronto, el avión empezó a girar de manera descontrolada.

No pudo hacerse nada para recuperar la estabilidad y, si el vuelo hubiese sido real, el piloto sólo habría podido salvarse saltando en paracaídas.

Analizando lo que había ocurrido pronto se encontró la causa.

El ordenador efectuaba automáticamente una compensación de la llamada "aceleración de Coriolis", que hace que un objeto en movimiento sobre la Tierra tienda a desviarse de su trayectoria recta; esta aceleración es la que ocasiona, por ejemplo, el que al vaciar la bañera el agua que cae por el desagüe esta se ponga a girar. La aceleración de Coriolis depende de la latitud (esto es, de cuánto nos encontremos hacia el norte o el sur) y cambia de sentido: en el hemisferio norte este giro se realiza en un sentido, en el sur en el sentido opuesto.

Los ingenieros previeron este efecto e incluyeron una corrección automática en el ordenador, que compensaba el efecto Coriolis y permitiría al avión volar recto.

Pero tomaron una decisión de diseño equivocada: las latitudes se representaron con variables de tipo "sin signo", cuando, en realidad, la latitud, en términos matemáticos, sí tiene signo (positivo hacia el norte, negativo hacia el sur). El resultado fue que en el hemisferio norte todo iba bien (un número sin signo se interpreta como positivo), pero en el sur la aceleración de Coriolis calculada por el ordenador era de signo contrario a la real, por lo que la corrección, en lugar de compensar el efecto Coriolis, lo aumentaba. Como resultado, el piloto sentía girar al avión, lo intentaba compensar con los mandos, se calculaba la nueva fuerza, aumentaba ésta y se aumentaba aún más con la compensación incorrecta, lo que hacía girar más el avión.

El defecto se detectó y corrigió cuando aún se estaba en fase de simulación, pero podría perfectamente haber pasado inadvertido y ocasionar un desastre algún tiempo después.

La mensaje final a tener en cuenta es que, por muy experto que se sea y por muy cualificado que esté nuestro equipo de desarrollo, las decisiones toma-

FIG 5. Hasta en el software más avanzado, como el de aviónica militar, debe verificarse cuidadosamente el diseño mediante la realización de pruebas exhaustivas.

das durante el diseño de un programa pueden ser tan incorrectas como el peor error de software. Pero son más peligrosas, ya que pueden pasar inadvertidas hasta que sea demasiado tarde.

METODOLOGÍAS Y HERRAMIENTAS CASE

Los dos métodos mencionados, organigramas y pseudocódigo, son dos de las herramientas más simples para iniciarse en el diseño de programas, pero no son ni mucho menos las únicas herramientas de que disponemos para ayudarnos en nuestra tarea. Existe toda una serie de metodologías de diseño que persiguen precisamente eso, dar unas pautas más o menos fáciles de seguir que permitan diseñar un programa más fácilmente que por el viejo método de "sentarse y pensar".

No podemos entrar en esta obra a analizar con una cierta profundidad todas las metodologías existentes, ni tan siquiera una; cada una de ellas es tan amplia que casi exige un libro para ella sola, pues incluye multitud de reglas a aplicar en cada caso. Lo que sí vamos a hacer es presentar la idea que persiguen algunas de las más importantes, y con qué herramientas dotan al diseñador de software para cumplir con sus objetivos.

Unas metodologías se dedican, más o menos específicamente, a ciertos tipos de aplicaciones. Por ejemplo, para las aplicaciones de manejo de *bases de datos* disponemos, entre otras, de la metodología *CASE* (no confundir con las herramientas CASE, que veremos en breve). Otras constituyen más una filosofía general, que puede emplearse para cualquier tipo de programación, como pueden ser la programación estructurada o la programación orientada a objetos. Dentro de éstas, diversas metodologías específicas intentan formalizar el proceso de diseño; por ejemplo, las metodologías *Rumbaugh, Booch, Fusion, Coad/Yourdon* o *Shlaer/Mellor*. El objetivo principal de las metodologías específicas es crear un estilo propio de programación, es decir, hacer más homogéneo el código escrito por diferentes programadores para un mismo caso.

Incluyen instrucciones sobre cómo plantear los problemas a resolver, cómo crear gráficos que representen exactamente las intenciones del diseñador y cómo interpretar todo eso.

Para ayudar en su elaboración se cuenta a menudo con diversos programas que permiten la generación de los gráficos y esquemas recogidos en la metodología: las herramientas CASE (nombre que procede del acrónimo inglés *Computer Assisted Software Engineering*, ingeniería de software asistida por ordenador). Este tipo de herramientas, además de facilitar el uso de una o varias metodologías, permiten generar automáticamente código directamente en un lenguaje de alto nivel, por ejemplo COBOL, C++ o Java. De esta manera el diseñador define con detalle la funcionalidad del programa, y el sistema genera a partir de dichas definiciones el código necesario. Esta generación no es completa, sino que suele crearse únicamente la estructura general del programa y se deja al programador la implementación de la funcionalidad concreta; así, por ejemplo, se generan las definiciones de los procedimientos y las funciones a usar en el programa, pero el código que va dentro del programa ha de ser generado manualmente.

El uso de estas herramientas tiene una ventaja adicional; en los mejores modelos, en cualquier momento del diseño podemos pasar a generar código de manera más o menos automática. Aunque la mayor parte de la lógica del programa quedará para ser escrita por el programador (cómo no), el simple hecho de disponer de esta facilidad permite aumentar la calidad del código, ya que muchos detalles (interfaces, tipos de variables, coherencia de los parámetros, etc.) son controlados por la propia herramienta. Como el código es generado por la herramienta CASE, puede reorganizarse fácilmente el diseño sin tener que reescribir una gran porción del código afectado. Es la diferencia entre mirar el paisaje y tratar de averiguar cómo se llama aquella montaña y tener un mapa para observarlo desde cualquier punto. La herramienta CASE es el mapa que nos permite movernos por nuestro software.

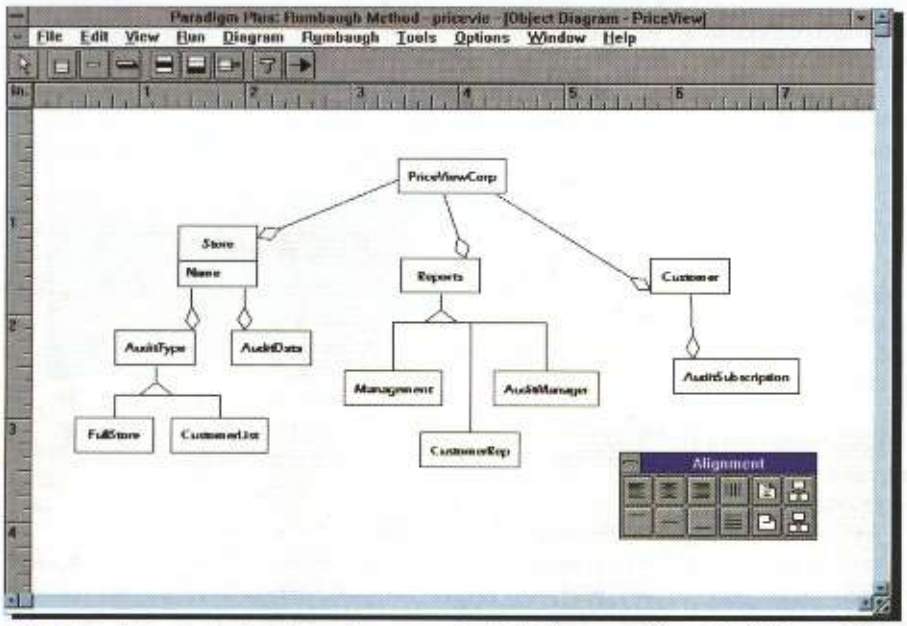


FIG 6. Las herramientas basadas en metodologías permiten simplificar el desarrollo de los esquemas y gráficos necesarios para la programación.

EN DEFINITIVA

El resumen de todo lo visto es muy claro: antes de programar hemos de diseñar la aplicación tal como va a ser. Para ello disponemos de algunos métodos sencillos, como los organigramas o el pseudocódigo, y de otros más complejos, como las metodologías y las herramientas CASE. A pesar de toda la ayuda que esto pueda suponer, la programación exige un esfuerzo mental importante, ya que ninguna de estas técnicas nos va a quitar el trabajo de pensar. A pesar de que los métodos y técnicas utilizados en la ingeniería siguen intentando introducirse en la informática, la programación aún sigue siendo una de las áreas tecnológicas, que siguen manteniendo características artesanales diferenciales importantes.

Soluciones prácticas (8)

Radiate Go!Zilla: Descargas automatizadas (I)

Descargar archivos desde Internet no tiene por qué ser la ardua tarea que sufren a diario cientos de miles de usuarios a lo largo y ancho de todo el globo. Existen aplicaciones como Radiate Go!Zilla capaces de hacer de este proceso algo rápido, cómodo y automático.

Descargar un archivo desde Internet es, en el mejor de los casos, una cuestión de (mucho) tiempo. En el peor, puede convertirse en una auténtica odisea en la que los cortes de conexión se turnan con las bajadas en picado del rendimiento para exasperar al más paciente. No obstante, existen aplicaciones especialmente diseñadas para facilitar la vida al internauta en este sentido, y una de las mejores es sin duda Radiate Go!Zilla. Su nombre, similar (y no por casualidad) al del monstruo cinematográfico, es sinónimo de potencia, eficacia y rapidez.

En resumen, Radiate Go!Zilla es una herramienta destinada a gestionar las descargas desde Internet, bien sean las que se realicen a través de FTP, o, bien las que se efectúen a través de HTTP. La aplicación se integra a las mil maravillas con el navegador de Internet e intercepta todas las peticiones de descarga. Cuando detecta una, entra en funcionamiento para controlar que el proceso de descarga se realice sin problemas.

Entre sus virtudes más destacadas se encuentra la de permitir la recuperación de una descarga incompleta, eso sí, si el servidor de Internet ofrece esta característica (una amplia mayoría de ellos es compatible con ella). Además, ofrece toda la información sobre tamaño, velocidad de conexión y tiempo estimado de descarga se refiere, además de incluir una lista con servidores alternativos de los que descargar el archivo. Y para finalizar con este comentario de introducción, permite organizar las descargas en el tiempo, de forma que podrá definir cuándo descargar un archivo automáticamente. Antes de continuar, tenga presente en el momento de la instalación de Radiate Go!Zilla que no tiene por qué instalar Gator y OfferCompanion. De igual forma, tampoco tiene que instalar QuickGuide si no lo desea.

EJEMPLO DE FUNCIONAMIENTO

En la primera toma de contacto con esta aplicación va a poder apreciar lo sencillo que resulta realizar una descarga con Radiate Go!Zilla. Para la realización de los siguientes pasos daremos por supuesto que el programa se encuentra instalado correctamente y que el ordenador dispone de una conexión a Internet válida, con todo lo que ello implica: la existencia de una línea tele-

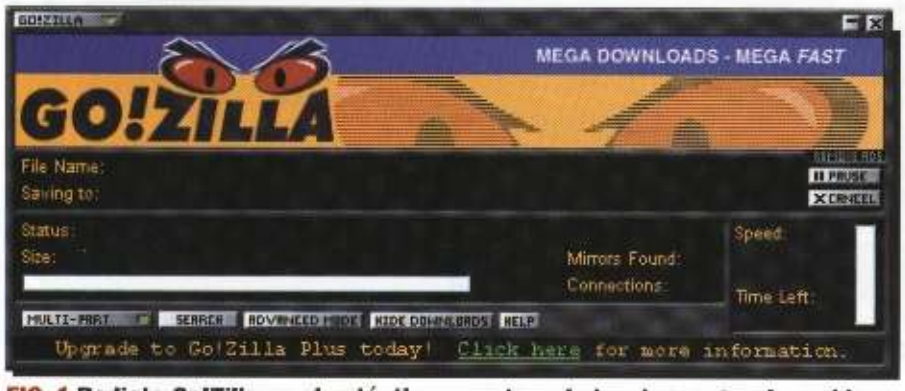


FIG. 1 Radiate Go!Zilla es el auténtico monstruo de las descargas de archivos desde Internet.

FIG. 2 Esta aplicación se integra con el navegador e intercepta las peticiones de descarga de archivos.

FIG. 3 El programa ofrece todo tipo de información: tamaño de descarga, tiempo que va a durar el proceso, velocidad del proceso.

fónica, un módem perfectamente configurado y conectado, y un Acceso telefónico a redes perfectamente configurado. Antes de empezar, y dado lo incómoda de manejar que resulta la interfaz que se muestra por omisión (es demasiado grande), haga clic sobre el botón **Search** para esconder esta ventana. Recuerde que más tarde podrá acceder a esta función de búsqueda de Radiate Go!Zilla usando este mismo botón. Vamos a suponer que ya se ha conectado a Internet, usando Internet Explorer, y que lo ha hecho a una página Web cualquiera para descargar un archivo de la forma habitual. Cuando haga clic sobre el enlace que dará inicio a la descarga no aparecerá la pantalla a tal efecto de Internet Explorer, sino que se mostrará una especial de Radiate Go!Zilla en la que el programa le informará de que ha capturado la petición de descarga.

En el momento en que visualice la pantalla puede hacer cuatro cosas distintas: comenzar la descarga que ha pedido (**Download Now**), descargar el archivo más tarde (**Download Later**), inhabilitar el funcionamiento de Radiate Go!Zilla (**Bypass Go!Zilla**) y olvidarse de la descarga (**Cancel Download**). Normalmente la opción que escogerá es **Download Now**, aunque si en ese preciso momento no quiere descargar el archivo porque, por ejemplo, la tarifa telefónica del tramo horario es demasiado cara, puede pulsar sobre **Download Later**; de este modo, Radiate Go!Zilla mantendrá una copia de la dirección de descarga del archivo para que pueda descargarlo cuando desee sin necesidad de buscarlo de nuevo.

INFORMACIÓN

En el momento de la descarga, Radiate Go!Zilla ofrece todo tipo de información relacionada con el archivo y con la conexión. Habrá observado que, tras hacer clic sobre el enlace de la descarga, apareció una pequeña ventana en la que podía ver el nombre del archivo (**File Name**), la ubicación donde se estaba copiando (**Saving to**), su tamaño (**Size**, parcial, lo que se descar-

RADIATE GO!ZILLA ES FREEWARE

Radiate Go!Zilla es una aplicación que, como verá a lo largo de esta unidad, resulta tremendamente útil, además de mostrarse muy potente en cuanto a sus funciones se refiere. Es por ello que probablemente se preguntará por qué no le costará ni un céntimo usarla. La explicación es muy sencilla: mientras permanezca conectado a Internet recibirá anuncios publicitarios que prácticamente no afectarán al rendimiento de la conexión dado su pequeño tamaño; créanos: está más que comprobado.
No obstante, si desea usar este programa sin recibir publicidad de ningún tipo, puede comprarlo por un precio realmente módico. Para obtener más información sobre esta aplicación, conéctese a la página Web www.gozilla.com.

FIG. 4 Acceda a la página Web www.gozilla.com para obtener más información sobre Radiate Go!Zilla en general y sobre la compra del programa en particular.

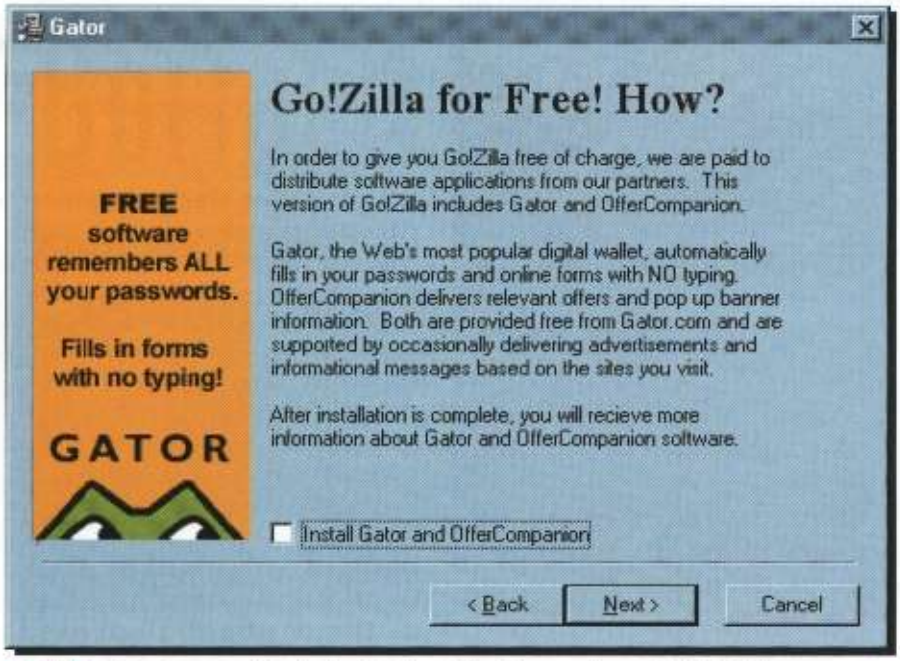


FIG. 5 No tiene por qué instalar Gator, OfferCompanion y QuickGuide.

gó hasta el momento, y total, la cantidad a descargar), el tiempo restante de descarga a la velocidad de ese momento (**Time Left**), el estado de la transmisión (**Status**), la cantidad de servidores "mirrors" que se han encontrado (**Mirrors Found**) y el número de conexiones (**Conections**). Pero eso es sólo en lo que se refiere al lado izquierdo de la ventana de descarga. Si se fija en la parte derecha, obtendrá todavía más datos de la descarga del archivo: la velocidad en Kb/s (**Speed**), una apreciación más sencilla de la velocidad (por ejemplo, **slow**, es decir, *lento*), el tanto por ciento descargado del total del archivo y una medida gráfica de la velocidad de la descarga respecto al tiempo transcurrido. Además de esta información, encontrará otras cosas. Existen dos botones, **Pause** y **Cancel** que le permitirán, respectivamente, hacer una pausa en la descarga o cancelarla. En la gráfica que muestra la evolución de la velocidad de la descarga respecto al tiempo existe una línea de color verde y, a su derecha, una fecha. Con este control tan sencillo podrá limitar el número de Kb/s al que se descargará el archivo; reduciéndolo conseguirá más ancho de banda para realizar otras operaciones, como consultar el buzón de correo electrónico o conectarse a una página Web.

FIG. 6 Supondremos que el módem está perfectamente instalado y que existe un Acceso telefónico a redes válido.

Podrá encontrar un control igual que el anterior en la ventana **Go!Zilla**, es decir, la ventana principal de Radiate Go!Zilla, concretamente en su parte izquierda. No obstante, mientras que la flecha anterior controla la tasa de descarga de un archivo en particular, aquí puede determinar la velocidad de obtención de todos los archivos que se están descargando en un momento

EVITAR LA CAPTURA

Siempre querrá que sea Radiate Go!Zilla quien realice las descargas, aunque pueden existir ocasiones especiales en las que no desee que el programa interfiera en la pulsación de un enlace (por ejemplo, puede suceder que el programa interprete por error que un enlace desemboca en una descarga y se empeñe en activarse). Si se encuentra alguna vez en la necesidad de obviar la ayuda de Radiate Go!Zilla, mantenga pulsada la tecla Alt mientras hace clic sobre el enlace.

determinado. De igual forma, existen dos botones llamados **Pause All** y **Cancel All** que cumplen una función exactamente igual que los anteriores, pero cuyo efecto es global sobre la totalidad de los archivos que está descargando. Si en algún momento se corta la descarga por cualquier motivo (por ejemplo, el servidor ha dejado de enviar el archivo), podrá volver a recuperar la descarga desde el punto donde la dejó con total seguridad si el servidor soporta esta característica. Tan sólo debe reiniciar la conexión con Internet si y sólo si se ha interrumpido, iniciar Radiate Go!Zilla si se ha cerrado, seleccionar en la ventana el archivo cuya descarga ha quedado interrumpida y hacer clic sobre **Download**, una opción que encontrará en la columna **Options** de la ventana **File Manager**. Radiate Go!Zilla intentará restablecer la conexión y, en caso de conseguirlo, retomará la descarga del archivo seleccionado sin que sea necesario volver a descargarlo desde el principio. De igual forma, también podrá iniciar la descarga accediendo al menú contextual del archivo en cuestión haciendo clic con el botón derecho del ratón sobre su superficie y seleccionando **Download**.

FIG. 7 Si oculta la ventana Search, la interfaz de Radiate Go!Zilla será más cómoda de manejar.

DESCARGA COMPLETA

Cuando el archivo se haya descargado satisfactoriamente, podrá ver que en la ventana **File Manager** aparece el texto **Success** en el apartado **Status**. También podrá comprobar que en **Done** aparece la cantidad **100.0%**, lo que indica que se ha descargado en su totalidad.

Para obtener más información, en la ventana **File Manager**, deslice la barra de desplazamiento hacia la derecha. Además de la fecha y la hora de la descarga (**Time Downloaded**), de la velocidad media del proceso (**Speed**) y del tiempo que resta hasta la descarga completa (**Time Left**, en este caso **00:00:00** porque ya se ha descargado el archivo), se mostrará la carpeta **C:\WINDOWS\Escritorio\Downloads**, donde reside el archivo.

En efecto, si accede al Escritorio descubrirá que existe una nueva carpeta llamada **Downloads**. Ábrala haciendo doble clic sobre ella y descubrirá que contiene el archivo que acaba de descargar. Ahora podrá hacer con él lo que le plazca.

EN DEFINITIVA...

En esta primera toma de contacto ha aprendido una cosa: Radiate Go!Zilla es una excepcional aplicación, con un potencial para la descarga automática realmente increíble. Y esto es algo que comprobaremos sobradamente en la siguiente unidad.

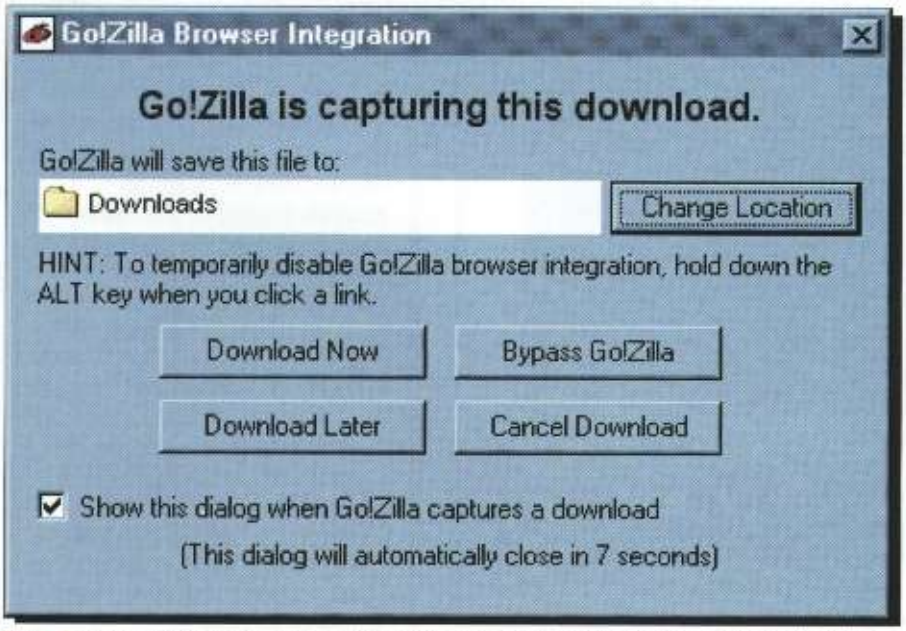


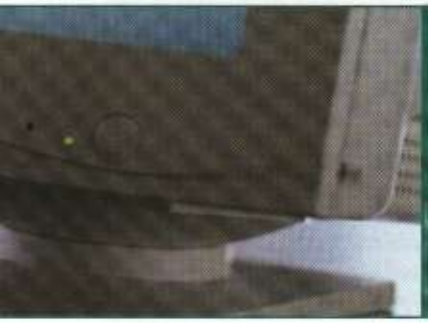
FIG. 8 Esta ventana le informará de que la aplicación ha capturado la petición de descarga.

PUBLICIDAD

Seguro que no le ha pasado por alto el hecho de que Radiate Go!Zilla muestra en la parte superior de su ventana principal banners publicitarios que cambian a medida que transcurre el tiempo. Como ya hemos dicho anteriormente, no debe preocuparse en absoluto, ya que el tiempo que dedica el programa a su descarga es realmente ínfimo y no afecta prácticamente en nada al rendimiento de la descarga de los archivos. Tenga en cuenta que el precio que está pagando por el uso del programa es prácticamente nulo, dado el escasísimo volumen de publicidad que hay que descargar.

Borland Delphi (3)

Componentes



Ofreceremos en el presente capítulo información sobre las fichas y componentes de Borland Delphi, directrices para elegir el componente más adecuado en cada caso e instrucciones para personalizar la barra de componentes. En esta unidad comenzaremos el desarrollo de una aplicación concreta, un juego, Tetris.

Las herramientas de diseño de Borland Delphi permiten crear cómodamente los distintos componentes en que se basan las interfaces de usuario de las aplicaciones para los entornos Windows, generando automáticamente el código Object Pascal básico de los eventos asociados a los mismos. Esta metodología consigue ahorrar tiempo de desarrollo, dada la frecuencia con que se recurre a determinado tipo de componentes. Cuatro son los pasos a dar para desarrollar la interfaz de usuario: crear las fichas, colocar los componentes, definir sus propiedades, y asociar el código a los sucesos de estos.

CREACIÓN DEL PROYECTO DE EJEMPLO

Crear un nuevo proyecto en blanco es sencillo, basta con acceder a Borland Delphi desde el menú de inicio de Windows. Al hacerlo se mostrará un nuevo proyecto en blanco con la ficha principal y su módulo de código asociado sobre el área de trabajo del programa. El siguiente paso a seguir será almacenar este proyecto en disco al pulsar el botón **Save All** (Guardar Todo) de la **Speedbar** o seguir la secuencia de comandos **File | Save All** (Archivo | Guardar Todo). Tras elegir la carpeta de destino dentro del cuadro de diálogo, asignaremos al archivo de código Object Pascal (de extensión PAS) asociado a la ficha principal el nombre *Tetris1* validando al hacer clic sobre el botón **Guardar**, y a continuación el nombre *Tetris* al archivo de proyecto (de extensión DPR).

Con esto se habrán creado en disco los archivos **Tetris1.dfm** y **Tetris1.pas**, correspondientes, respectivamente, al código de la ficha principal y del módulo Object Pascal asociado a la misma, así como los archivos **Tetris.dof**, **Tetris.dpr** y **Tetris.res**, de opcio-

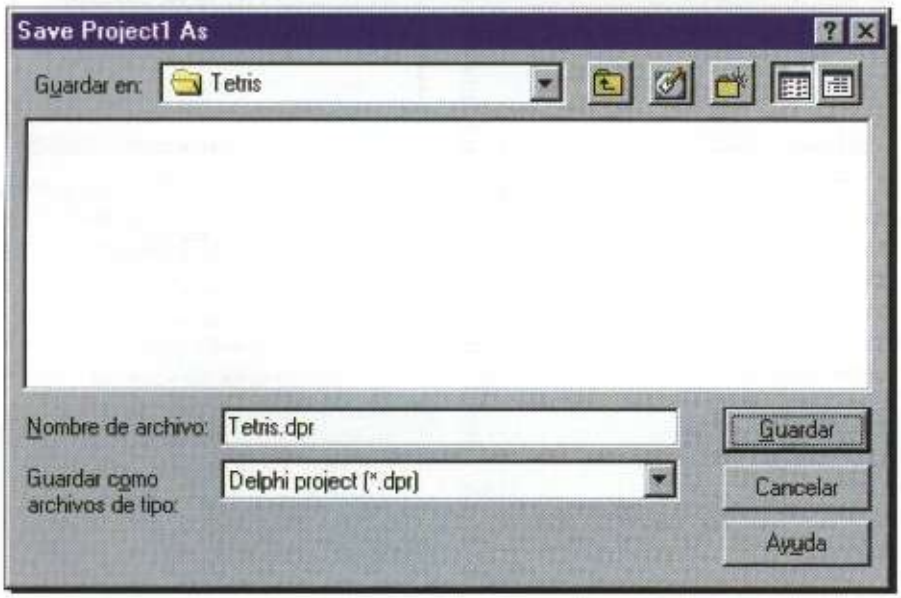


FIG. 1 Tras dar nombre al archivo de código, se debe nombrar igualmente el archivo de proyecto de ejemplo.

IMPORTANCIA DE LAS FICHAS

Las fichas son los componentes básicos del diseño visual, dando lugar a las ventanas y cuadros de diálogo de las aplicaciones. Sobre las mismas pueden disponerse componentes visibles, como los controles para que el usuario pueda interactuar con la aplicación, e invisibles, como los de explotación de recursos de acceso a bases de datos.

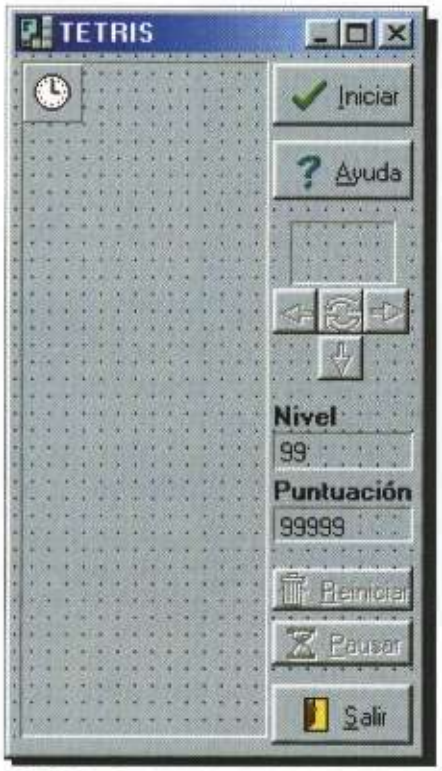


FIG. 2 La ficha principal de la aplicación de ejemplo aloja los componentes que permitirán controlar el desarrollo del juego.

nes, descripción y recursos de proyecto, respectivamente.

COMPONENTES

Como ya se ha comentado anteriormente, los componentes son objetos y, como tales, encapsulan conjuntos de funciones, heredan datos y comportamientos de objetos antecesores, y operan de manera intercambiable con objetos derivados antecesores comunes gracias al *polimorfismo*. Además de sus características distintivas, los objetos heredan otras comunes de su antecesor **TComponent**.

Los componentes contienen tres tipos de información: información de estado, de acción y de respuesta.

- **Estado.**

Se denomina propiedad. Se trata de atributos editables, como **Width** (Anchura) y **Color**.

- **Acción.**

Los componentes pueden disponer de métodos para instruir al componente de su sometido, que permiten desarrollar determinadas acciones, y que se subdividen a su vez en procedimientos y funciones, como **Refresh** (Refrescar) y **Hide** (Ocultar).

- **Respuesta.**

Pueden asociarse sucesos a los componentes mediante código Object

PREPARACIÓN DE LA FICHA PRINCIPAL DEL EJEMPLO

Desde el Object Inspector (Inspector de Objetos) dimensionaremos la ficha principal otorgando a las propiedades ClientHeight (Altura cliente) y ClientWidth (Anchura cliente) los valores *327* y *187*, respectivamente, para establecer así un tamaño con espacio suficiente para alojar la ventana de juego propiamente dicha y los controles correspondientes. Desactivaremos la característica de autoscroll asignando el valor False (Falso) a la propiedad AutoScroll. Para dar nombre a la ficha basta con teclear el texto *TETRIS* para la propiedad Caption (Rótulo), con lo cual pasará a mostrarse el nombre en la barra de título de la ventana de la aplicación.
El siguiente paso será asociar el icono de la aplicación. Para ello se debe hacer clic sobre el botón asociado a la propiedad Icon (Icono) para mostrar el cuadro de diálogo Picture Editor (Editor de Imágenes). Al hacer clic sobre el botón Load (Cargar) puede seleccionarse el icono con el nombre de la aplicación y extensión ".ICO" al hacer doble clic sobre el mismo. Ya para terminar, asignaremos a la propiedad Visible el valor True, tras lo cual puede volverse a almacenar en disco la aplicación por el procedimiento habitual. Éste es el código Object Pascal generado automáticamente como consecuencia de estas definiciones:

```
unit Tetris1;
interface
uses
Windows, Messages, SysUtils, Classes, Graphics, Controls,
Forms, Dialogs;
type
TForm1 = class(TForm)
private
{ Private declarations }
public
{ Public declarations }
end;
var
Form1: TForm1;
implementation
{$R *.DFM}
end.
```

FIG. 3 La paleta Component dispone de una serie de fichas, accesibles mediante pestañas, en las cuales alojar ordenadamente los distintos componentes.

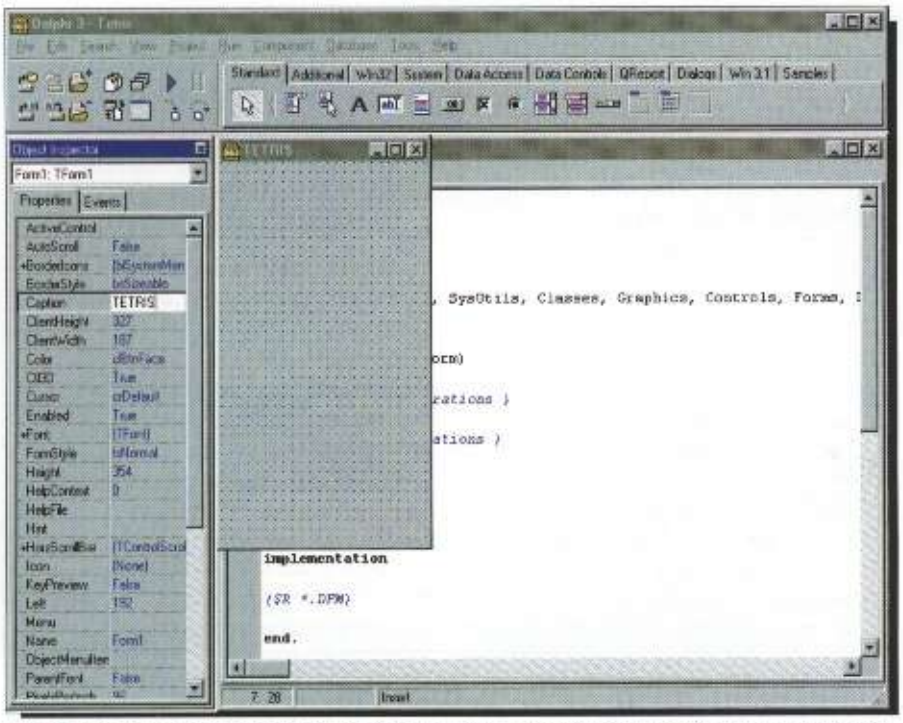


FIG. 4 Aspecto de la ficha principal tras dar nombre a la ventana de la aplicación mediante la propiedad Caption.

Pascal para hacerlos interactivos, como **OnClick** (Al pulsar botón) y **OnKeyDown** (Al pulsar tecla).

AGRUPACIÓN DE COMPONENTES

Continuando con los aspectos teóricos, la paleta **Component** (Componentes) recoge los componentes estándar de Borland Delphi clasificados en fichas atendiendo a su función. Al ser personalizable permite tanto añadir como eliminar elementos, e incluso crear plantillas de componentes.

Los componentes pueden agruparse tanto de forma jerárquica como funcional, dividiendo la primera los componentes por su naturaleza, mientras que la segunda atendiendo a su función propiamente dicha.

■ **Jerarquía.**

A efectos de jerarquía los componentes pueden dividirse en no visuales y visuales (controles), subdividiéndose estos últimos a su vez en controles de ventana y gráficos.

• **Componentes no visuales.**

Son los elementos de programa con los cuales no puede interactuar directamente el usuario, como enlaces con cuadros de diálogo y temporizadores, y que se encuentran accesibles en la fase de diseño en forma de iconos para asignarles propiedades y sucesos. Algunos de estos componentes, sin embargo, sí se muestran visualmente en el momento de ejecutarse la aplicación, como es el caso de la barra de menús principal o un cuadro de diálogo estándar de Windows.

• **Componentes visuales.**

Por contra, los controles son elementos visibles sobre los cuales puede interactuar el usuario y de aspecto similar tanto en la fase de diseño como en la de ejecución, como pueden ser los botones y cuadros de lista. Los controles pueden agruparse, a su vez, en controles de "ventana" y "gráficos". Los primeros pueden recibir el foco de entrada de datos, como es el caso de la mayoría de los controles estándar de Windows.

Los segundos, también denominados de "no ventana", no pueden recibir el foco, aunque consumen menos recursos del sistema.

■ **Funcionalidad.**

El amplio repertorio de componentes hace dudar en ocasiones sobre la idoneidad de cuál elegir en cada caso, ya que puede utilizarse con frecuencia más de uno para un cometido similar. En estos casos la decisión se tomará atendiendo a la capacidad o menor consumo de recursos del sistema.

ELEMENTOS COMUNES Y ELECCIÓN DE COMPONENTES

Las propiedades de los componentes pueden ser en parte propias y en otra heredadas de antecesores, denominándose en este caso elementos *comunes* como es, por ejemplo, la propiedad **Height** (Altura) de los controles. Las propiedades exclusivas se denominan, por su parte, elementos *clave*, como es el caso de la propiedad **Checked** (Activada) de las casillas de verificación, por ejemplo. La propiedad

FICHAS DE LA PALETA COMPONENT

Standard	**Controles y menús estándares de Windows.**
Additional	**Controles personalizados.**
System	**Componentes y controles de sistema, incluyendo temporizadores, y elementos multimedia.**
Win32	**Controles estándares de Windows 95/98/ME y NT.**
Dialogs	**Cuadros de diálogo estándares de Windows.**
Data Access	**Componentes no visuales para acceso a bases de datos, tablas, consultas e informes.**
Data Controls	**Controles visuales enlazados a datos.**
Win 3.1	**Componentes para compatibilidad con la versión 3.1 de Windows.**
Internet	**Componentes para gestionar aplicaciones cliente / servidor y protocolo TCP/IP.**
Samples	**Componentes personalizados de ejemplo.**
QReport	**Componentes para crear informes incrustados.**
Decision Cube	**Controles para mostrar bases de datos.**
ActiveX	**Controles ActiveX de ejemplo.**

común a todos los componentes más importante es **Name** (Nombre) y está encargada de dar un nombre identificativo y exclusivo al componente, según las normas de codificación del lenguaje.

Algunos componentes tienen funciones similares, aunque las desempeñan de forma especializada, razón por la cual conviene conocer las diferencias existentes entre los diferentes componentes de un mismo grupo.

■ **Propiedades comunes.**

Las propiedades comunes a todos los componentes y que son las que mejor los definen se dividen en categorías: tamaño y posición, visualización, padre, desplazamiento y arrastrar y soltar.

• **Tamaño y posición.**

Son propiedades vinculadas a los componentes visuales y, en general, se establecen al emplazar y arrastrar los objetos sobre las fichas, aunque pueden asignarse igualmente en tiempo de ejecución mediante código de programa: **Height** define la altura, **Width** la anchura, **Top** (Superior) el desplazamiento desde el margen superior y **Left** (Izquierda) el desplazamiento desde el margen izquierdo.

• **Visualización.**

Estas propiedades se refieren a la presentación de objetos: **BorderStyle** (Estilo de borde) determina el contorno, **Color** el color de fondo, **Ctrl3D** (Control tridimensional) el aspecto tridimensional o plano, y **Font** (Fuente) las características tipográficas de fuente, tamaño, color y estilo.

• **Padre.**

Este conjunto de propiedades garantiza la homogeneidad de aspecto de los componentes de la aplicación sobre la base de su dependencia de la jerarquía superior. Para ello se puede asignar el valor **True** (Verdadero) a las propiedades **ParentColor** (Color heredado), **ParentFont** (Fuente heredada) y **ParentCtrl3D** (Control tridimensional heredado), para el color, tipografía y aspecto tridimensional, respectivamente, según se desee.

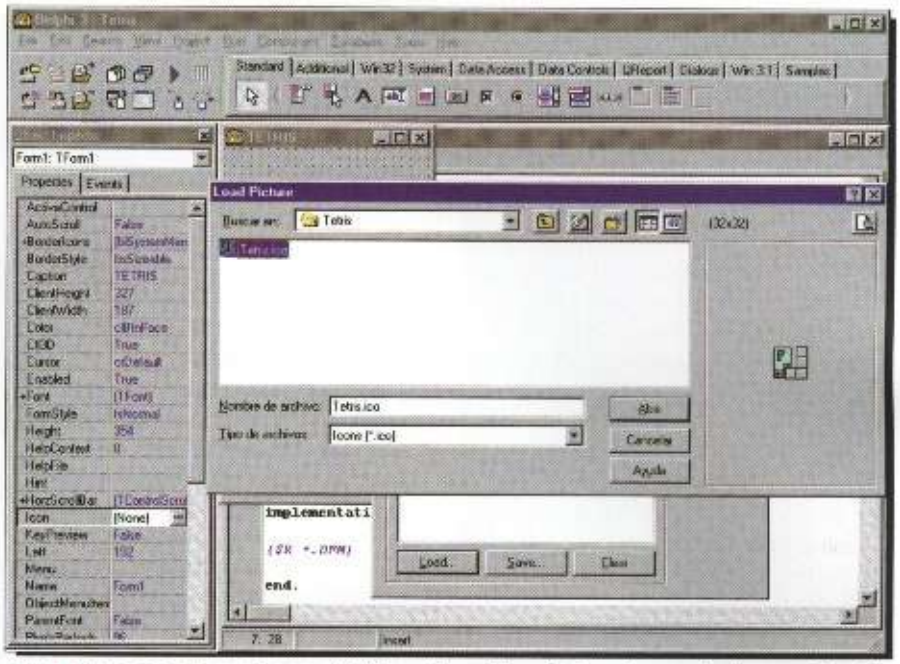


FIG. 5 Para asociar un icono gráfico a la aplicación basta con indicar su localización en disco como valor de la propiedad Icon.

• **Desplazamiento.**

Son propiedades que afectan a la identificación y desplazamiento del usuario por los componentes de las fichas: **Caption** (Etiqueta) asigna rótulo y tecla abreviada al componente, y **TabOrder** (Orden de tabulación) establece el orden de tabulación para pasar de un campo a otro al pulsar la tecla **Tab** cuando se activa la propiedad **TabStop** (Posición de tabulación).

• **Arrastrar y soltar.**

El comportamiento para la acción de arrastrar y soltar lo definen dos propiedades: **DragMode** (Modalidad de arrastre) para indicar la modalidad de arrastre y **DragCursor** (Cursor de arrastre) para determinar el aspecto del puntero del ratón durante la operación.

TIPOS DE COMPONENTES BUTTON

Button	**Botones de comando con etiqueta de texto.**
BitBtn	**Botones de comando con icono y etiqueta de texto.**
SpeedButton	**Botones de barra de herramientas.**
CheckBox	**Casillas de verificación.**
RadioButton	**Casillas de selección.**
ToolBar	**Barra de herramientas configurable.**
CoolBar	**Controles de ventana redimensionable.**

CONTROLES DE TEXTO

Existen varios componentes para mostrar y editar texto, según necesidades: **Edit** (Edición) para una sola línea, **Memo** (Memo) para varias líneas, **MaskEdit** (Máscara de edición) para forzar la entrada a un formato dado o limitar el uso de determinados caracteres, y **RichEdit** (Edición de texto enriquecido) para un número ilimitado de líneas o texto con formato.

■ **Propiedades comunes a controles de texto.**

Existen una serie de propiedades comunes a los controles de texto: **Text** (Texto) establece el texto inicial que se muestra, **CharCase** (Caja de carácter) limita las combinaciones de mayúsculas y minúsculas, **ReadOnly** (Sólo lectura) permite o no la edición de texto, **MaxLength** (Longitud máxima) limita el número máximo de caracteres, **PasswordChar** (Carácter clave) oculta el texto introducido y **HideSelection** (Ocultar selección) controla el resalte de texto cuando el objeto no tiene el foco.

■ **Propiedades comunes a objetos memo y de texto con formato.**

Además de las propiedades comunes comentadas, los componentes que admiten varias líneas de texto tienen otras específicas: **Alignment** (Alineación) indica la alineación (izquierda,

derecha o centrado), **WordWrap** (Ajuste de palabras) lleva automáticamente a la siguiente la palabra que no cabe completa en la línea, y **WantReturns** (Retorno requerido) y **WantTabs** (Tabulador requerido) deciden cómo deben tratarse en el texto las pulsaciones de estas teclas especiales.

- **Propiedades de objetos memo.**

Éstas son sus propiedades específicas: **AutoSelect** (Selección automática) para mostrar el texto a editar resaltado, **SelText** (Texto seleccionado) para comprobar o sustituir el texto seleccionado por el usuario, y **SelStart** (Seleccionar comienzo) y **SelLength** (Seleccionar longitud) para comprobar y resaltar bloques de texto.

BOTONES Y SIMILARES

Los botones y controles similares son el medio más común de ejecutar comandos, a excepción de las propias opciones de menú. El botón estándar admite variaciones para adaptarse a cada necesidad.

- **Botones de texto.**

Es el tipo de control más general, y puede ubicarse y dimensionarse sobre las fichas.

Sus propiedades permiten asociarle una etiqueta de texto y se le puede vincular una acción mediante la propiedad **OnClick** (Al pulsar).

Se puede activar su propiedad **Cancel** (Cancelar) para que se ejecute la acción asociada cuando el usuario pulse la tecla **Esc** cuando el botón tenga el foco, o bien activar la propiedad **Default** (Omisión) para que la pulsación de **Intro** ejecute también la acción asociada, incluso aunque el control no tenga el foco.

- **Botones con icono.**

Son similares a los anteriores, pero permiten incluir un icono descriptivo de su cometido, a elegir dentro de la biblioteca del programa o definido por el propio usuario. La propiedad **Glyph** (Glifo) permite añadir el icono, **Kind** (Clase) elegir una de las modalidades de botones estándar con icono, **Layout** (Perfil) escoger la situación relativa del icono, **Margin** (Margen) definir el margen del botón al borde de éste, **Spacing** (Espaciado) especificar la distancia entre imagen y texto, y **NumGlyphs** (Número de glifos) indicar el número de estados de los botones (sin pulsar, pulsado y pulsación mantenida).

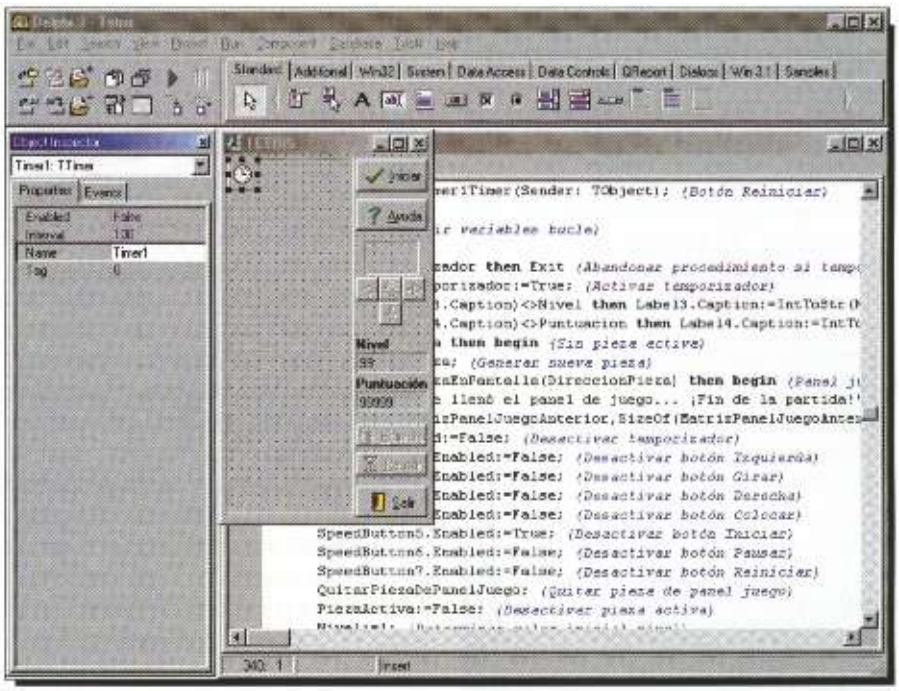


FIG. 6 El componente Timer1 es de tipo no visual y, por lo tanto, no se muestra durante la ejecución de la aplicación de ejemplo.

- **Botones rápidos.**

Éste es un conjunto de botones predefinido con que cuenta el programa, generalmente con icono asociado, para ejecutar comandos o activar modos, dotados a su vez de características que facilitan su empleo como conjunto, razón por la cual se emplazan frecuentemente sobre paneles para crear barras de herramientas. La propiedad **Down** (Abajo) determina si deben mostrarse o no como seleccionados, **AllowAllUp** (Todos arriba) permite mostrar el conjunto como no seleccionado

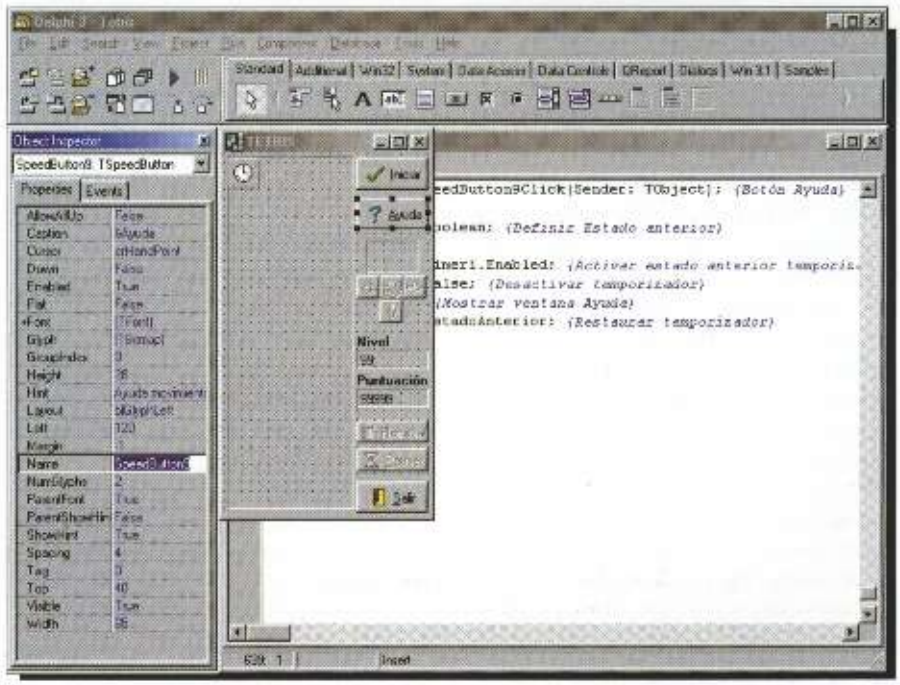


FIG. 7 La propiedad Name es la más común de todas, sirviendo de identificador exclusivo del componente.

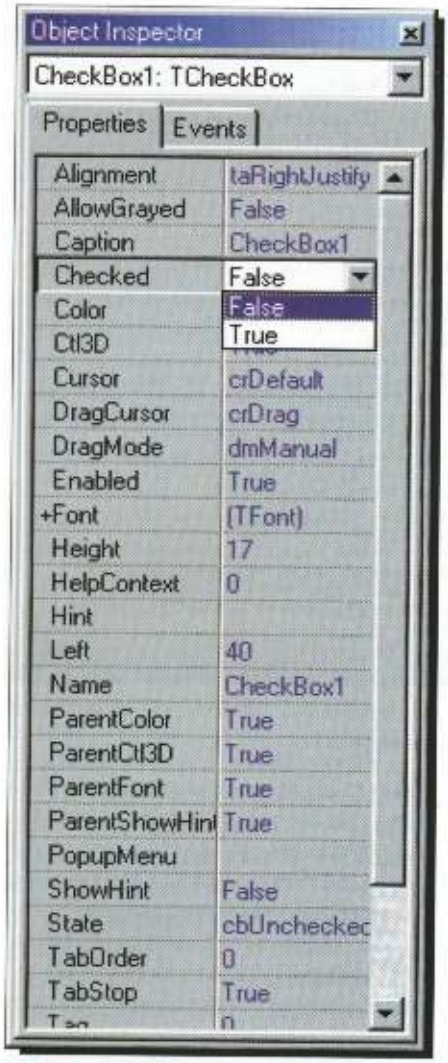


FIG. 8 La propiedad Checked es característica de los controles conocidos como casillas de verificación y casillas de selección.

o como casillas de selección, y **GroupIndex** (Índice de grupo) permite que se comporten como grupo al asignar a todos el mismo valor.

- **Casillas de verificación.**

Son los elementos ideales para la toma de decisiones de dos estados. La propiedad **Checked** indica la selección, **AllowGrayed** (Permitir gris) permite introducir un tercer estado como marca atenuada, y **State** (Estado) determina los distintos estados que puede adoptar el control

- **Casillas de selección.**

Permiten crear conjuntos de decisiones de dos estados excluyentes entre sí, pudiendo colocarse tanto individualmente como en conjunto. En el segundo caso se debe definir una propiedad **Caption** común e identificar los elementos individuales mediante la propiedad **Items** (Elementos). Al igual que en el caso de las casillas de verificación, la propiedad **Checked** indica el estado de activo o inactivo.

- **Barras de herramientas.**

A partir de un componente panel y una serie de botones rápidos puede definirse una barra de botones, así como partiendo de un componente **ToolBar** (Barra de herramientas) y seleccionando **New Button** (Nuevo botón) para cada nuevo botón que se desee crear.

Todos los botones de la barra son del mismo tamaño, así como los demás controles que se sitúen sobre la misma, disponiéndose automáticamente en varias filas de ser necesario. Los botones pueden agruparse mediante espacios extra y separadores, y se pueden añadir bordes y transparencias.

- **Barras flotantes.**

Estos controles alojan otros controles hijos, que pueden desplazarse y redimensionarse individualmente, situándose cada uno de ellos en una banda individual. El usuario coloca los controles arrastrando con el ratón el manejador del margen inferior izquierdo de cada banda.

La propiedad **Bands** (Bandas) incluye una serie de objetos **TcoolBand**; **FixedOrder** (Orden fijo) indica si el usuario puede modificar el orden de las bandas, y **FixedSize** (Tamaño fijo) especifica si se deberá adoptar automáticamente un tamaño predeterminado.

BOTONES DE CONTROL DE LA APLICACIÓN DE EJEMPLO

La aplicación se controla por medio de cinco botones de funciones, más cuatro de desplazamiento, mostrando los primeros texto e icono y los segundos únicamente icono, todos ellos procedentes de la biblioteca de Borland Delphi.

Por otro lado, como propiedad común, van a contar con texto de ayuda emergente.

Para colocar el primer botón de control de movimiento seleccionaremos el componente **SpeedButton** (Botón rápido) dentro de la ficha **Additional** (Adicional) de la paleta **Component** para, a continuación, arrastrar con el ratón para trazar un pequeño botón cuadrado ubicado hacia la mitad del margen derecho de la ficha.

Para emplazarlo de manera precisa basta con introducir los valores *120* para la propiedad **Left** (Izquierda) y *110* para **Top** (Arriba), y para dimensionarlo asignar el valor *21* a las propiedades **Height** (Alto) y **Width** (Ancho).

A continuación, asociaremos un gráfico al abrir el cuadro de diálogo de la propiedad **Glyph** (Glifo) y localizar la carpeta **\Delphi3m\Images\BUTTONS** para hacer doble clic sobre el archivo **Arrow11.bmp**, validando al pulsar **OK** (Aceptar). Cambiaremos también el puntero del ratón para que se muestre

FIG. 9 La propiedad Height establece la altura del componente, siendo una de las propiedades comunes a casi todos los controles.

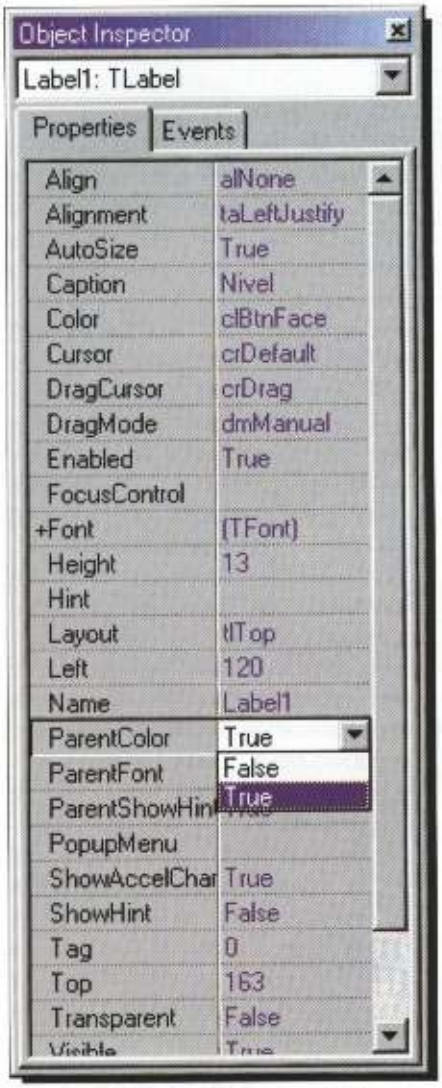


FIG. 10 La propiedad ParentColor, que puede adoptar los valores True y False, controla la herencia de color del componente de la jerarquía superior.

como una mano al situarse sobre el botón.

Para ello asignaremos el valor **crHandPoint** (Cursor de mano) a la propiedad **Cursor**. Daremos también el valor **False** a la propiedad **Enable** (Activar) para que el botón se muestre como no activo por omisión (con su icono en gris y sin relieve). Nos ocuparemos ahora de la información de ayuda emergente del botón, al introducir para la propiedad **Hint** (Ayuda) el valor **Mover izquierda** y activar la propiedad **ShowHint** (Mostrar ayuda) otorgándole el valor **True**. El código generado automáticamente por Borland Delphi para este primer botón será:

```
procedure
TForm1.SpeedButton1Click
(Sender: TObject);
begin
end;
end.
```

Tras crear el primero de los botones de movimiento, generar los restantes es tarea fácil. Tras seleccionar el botón al hacer clic sobre el mismo, lo copiaremos en el Portapapeles mediante la secuencia de comandos **Edit | Copy** (Editar | Copiar) o la combinación de teclas **Control + C**, para pegarlo seguidamente mediante la secuencia de comandos **Edit | Paste** (Editar | Pegar) o la combinación de teclas **Control + V**, situándolo inmediatamente a la derecha del original. Para efectuar el ajuste preciso basta con recurrir al **Object Inspector** e introducir los valores *141* y *110* para las propiedades **Left** y **Top**, respectivamente. Cambiaremos seguidamente el icono del botón por el procedimiento descrito anteriormente, para elegir en este caso el archivo **Retry.bmp** del clip-art. Del mismo modo, nos será fácil pegar nuevamente el primero de los botones inmediatamente a la derecha, en la posición *162* para **Left** y *110* para **Top**, y asociarle el gráfico **Arrow1r.bmp**. Ya sólo resta añadir un tercer botón para situarlo justo debajo del central del grupo de tres, para lo cual se puede pegar inicialmente el componente y asignarle seguidamente valores. En este caso la posición precisa sería de *141* para la propiedad **Left** y *132* para la propiedad **Top**, y el gráfico a asociar **Arrow1d.bmp**.

Falta el pequeño detalle, sin embargo, de los textos de ayuda emergente para estos botones. Para incluirlo basta con ir seleccionando cada botón y asignar a su propiedad **Hint** los valores *Mover izquierda*, *Girar pieza*, *Mover derecha* y *Colocar pieza*, respectivamente, con lo cual puede darse por concluida la configuración botones de controles de movimiento.

Colocaremos ahora el primero de los botones de función. Como en el caso anterior, seleccionaremos un componente **SpeedButton** dentro de la ficha **Additional** de la paleta **Component** y lo emplazaremos en el margen superior derecho de la ficha, de forma precisa en los valores *120* para **Left** y *4* para **Top**. En cuanto al tamaño se refiere, los valores adecuados serán *29* para **Height** y *65* para **Width**. En este

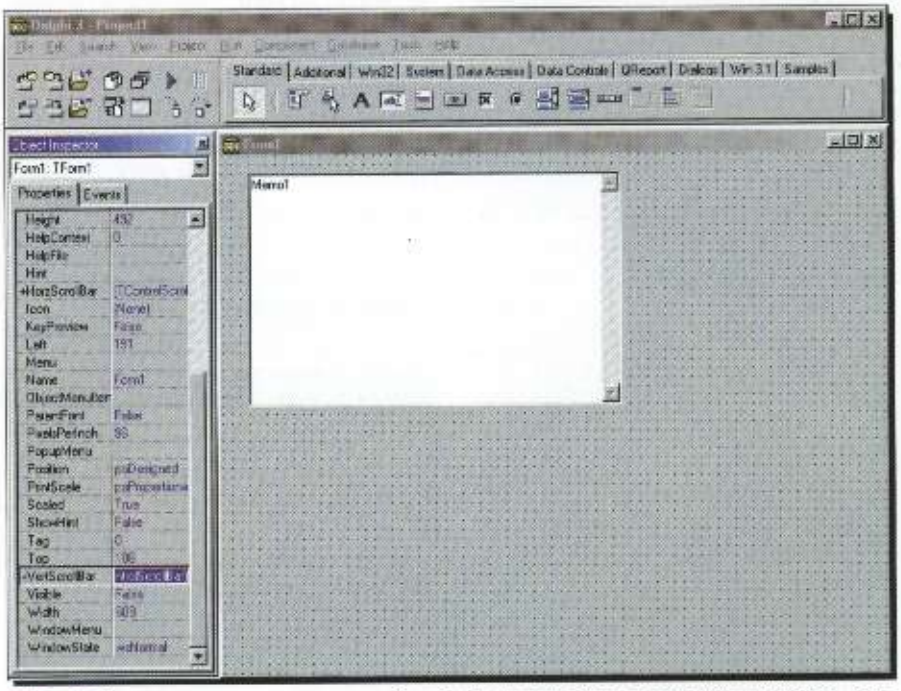


FIG. 11 Los campos memo permiten introducir gran cantidad de texto, con posibilidad de ajuste de palabras, barras de desplazamiento y control de las pulsaciones de las teclas Intro y Tab.

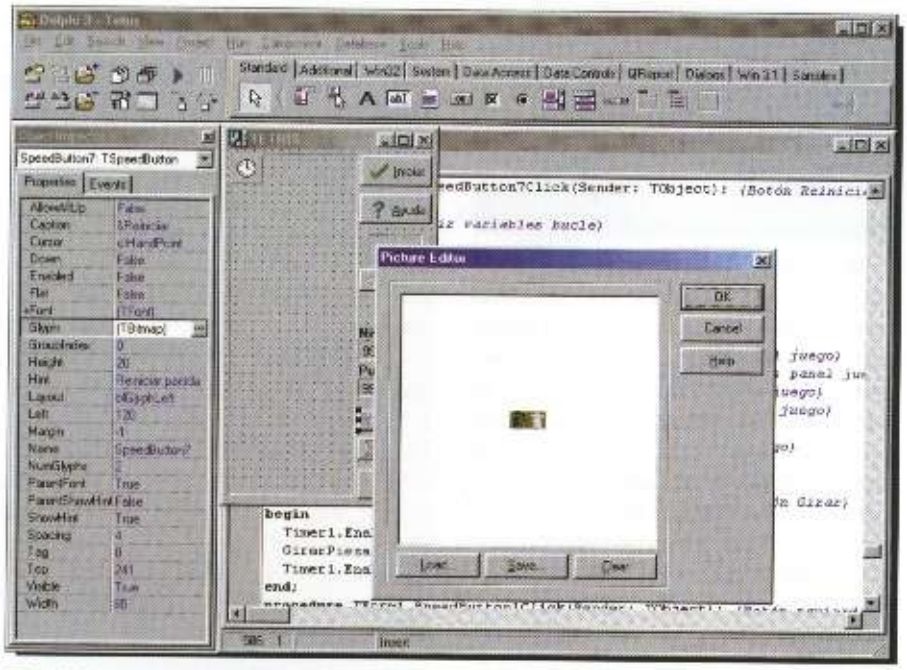


FIG. 12 La propiedad Glyph permite asociar a un botón una imagen de mapa de bits, tomada tanto de la biblioteca del programa como creada por el usuario.

caso teclearemos como rótulo o etiqueta *&Iniciar* para su propiedad **Caption**. Anteponemos el carácter "&" a la inicial para que ésta sirva automáticamente en el momento de la ejecución para activar el botón como alternativa a la pulsación por ratón. Asignaremos como anteriormente **crHandPoint** para la propiedad **Cursor**. En este caso, además del rótulo de texto añadiremos un icono descriptivo a su izquierda, que elegiremos para la propiedad **Glyph** por el procedimiento descrito anteriormente, seleccionando el archivo **Check .bmp**.

Como texto de ayuda emergente teclearemos ahora *Iniciar partida* para la propiedad **Hint**. Dejaremos también

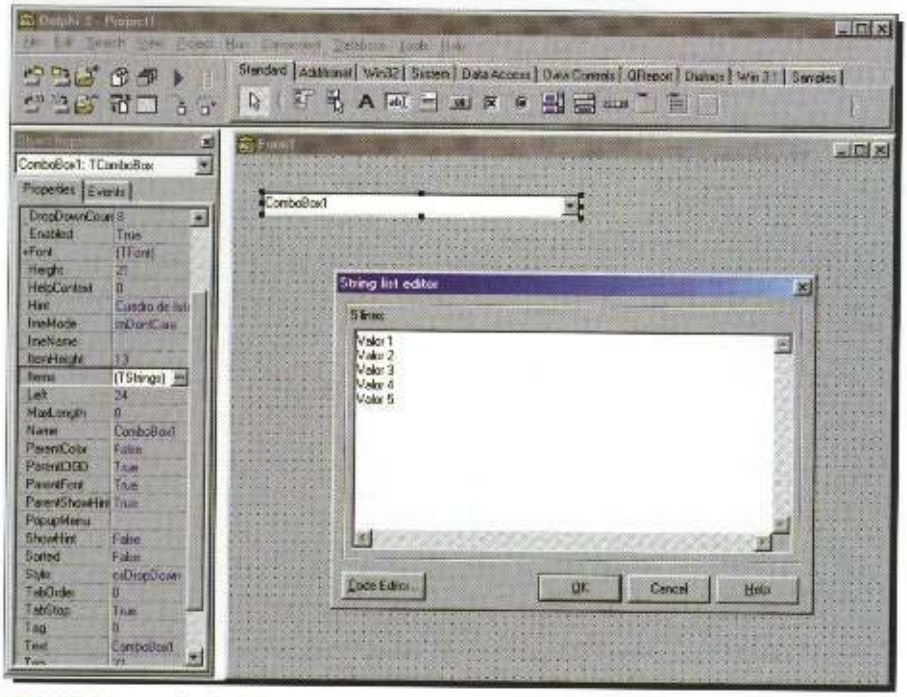


FIG. 13 La propiedad Items permite definir la lista de posibilidades entre las cuales podrá elegir el usuario al ejecutar la aplicación.

ParentShowHint en **False** para anular la herencia para mostrar ayuda emergente contextual, con lo cual podemos dar por concluida su definición.

Definiremos seguidamente el botón 6, duplicado inicialmente del 5, que tendrá como **Caption** el valor *&Pausar* y como **Enable** valor **False**, siendo su **Glyph** el archivo **Hourglass.bmp**, su **Hint** el texto *Pausar partida*, y su posición exacta de *120* para **Left** y *266* para **Top**. Duplicaremos de nuevo este último para situar una copia justo encima, que tendrá como **Caption** valor *&Reiniciar*, siendo su **Glyph** el archivo **Trash.bmp**, su **Hint** el texto *Reiniciar partida*, y su posición exacta de *120* para **Left** y *241* para **Top**.

Colocaremos ahora en el margen inferior derecho una réplica del botón del margen superior derecho para, seguidamente, modificar sus valores de partida. Asignaremos como **Caption** el valor *&Salir*, siendo su **Glyph** el archivo **Dooropen.bmp**, su **Hint** el texto *Salir del programa*, y su posición exacta de *120* para **Left** y *296* para **Top**. Pegaremos un nuevo botón modelo debajo del primero. Asignaremos como **Caption** el valor *&Ayuda*, siendo su **Glyph** el archivo **Help.bmp**, su **Hint** el texto *Ayuda movimientos*, y su posición exacta de *120* para **Left** y *40* para **Top**. Con esto terminamos la definición de botones. No obstante, podemos mejorar los iconos de los botones de inicio de partida, ayuda y giro de piezas, sustituyendo los iconos de la biblioteca estándar por los archivos **Iniciar.bmp**, **Ayuda .bmp** y **Girar.bmp**, respectivamente, tomados de la carpeta de los archivos de ejemplo.

LISTAS

Tiene como cometido general mostrar al usuario una serie de opciones entre las cuales elegir, existiendo distintas variantes.

- **Propiedades comunes.**

Existen una serie de propiedades comunes que rigen la presentación de listas.

La propiedad **Columns** (Columnas) define el número de elementos, **IntegralHeight** (Altura integral) muestra las entradas que quepan en el espacio vertical asignado según se desee, **ItemHeight** (Altura de elemento) de-

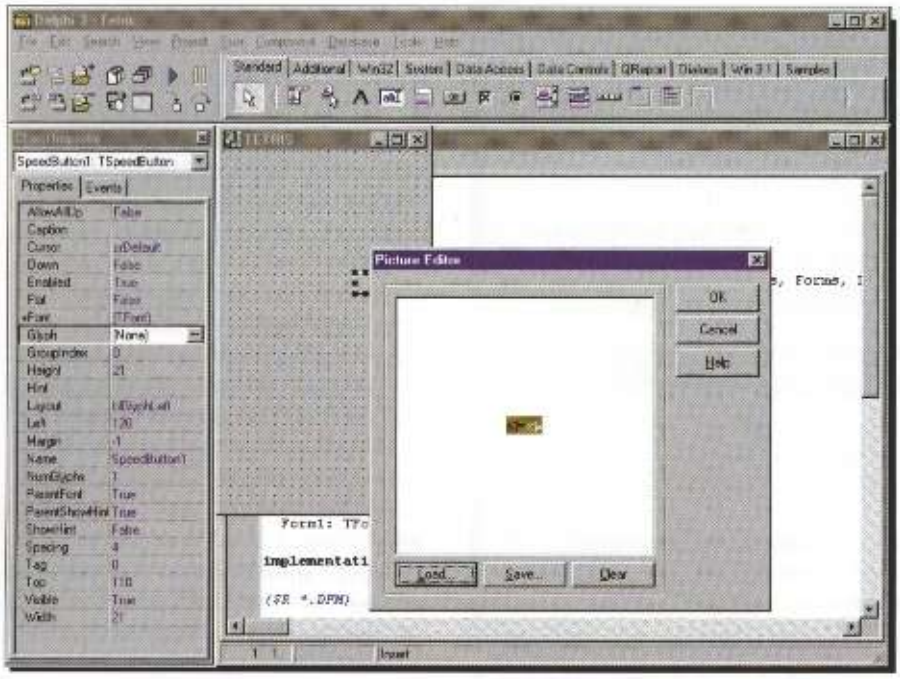


FIG. 14 La biblioteca de imágenes de Borland Delphi es capaz de satisfacer las necesidades más habituales de programación.

fine la altura de los elementos, **Items** define la lista de cadenas de caracteres con los valores posibles, **ItemIndex** (Elemento índice) determina el elemento seleccionado de la lista, **MultiSelect** permite seleccionar más de un elemento con ayuda de las teclas **Control** y **Mayús**, y **Sorted** (Ordenada) impone selectivamente la clasificación alfabética.

- **Cuadros de lista.**

Muestran una lista de opciones de las cuales el usuario puede elegir una o varias. La propiedad **Items** ofrece las modalidades **Add** (Añadir), **Delete** (Eliminar) e **Insert** (Insertar) para gestionar la lista de cadenas, **Multiselect** activa o desactiva la selección múltiple de elementos, **ExtendedSelect** (Selección extendida) permite utilizar las teclas **Control** y **Mayús** para la selección múltiple, y **Style** (Estilo) permite mostrar elementos como texto o gráficos.

- **Cuadros de lista desplegables.**

Este tipo de controles combina el cuadro de texto y el de lista, pudiendo tanto teclearse el contenido como elegir una opción dentro del cuadro de lista. La propiedad **Text** (Texto) se activa al introducirse manualmente texto, **DropDawnCount** (Número desplegado de elementos) indica el número de líneas a mostrar y **Style** permite elegir el estilo del control.

- **Vistas en árbol.**

Son componentes especializados para mostrar información con estructura jerárquica, dotados de botones para ampliar y reducir el nivel de detalle del esquema. Opcionalmente pueden añadirse iconos para los elementos así como casillas de verificación para reflejar información de estado. **Indent** (Sangrado) establece la separación horizontal entre elementos y sus padres, **ShowButtons** (Mostrar botones) permite mostrar u ocultar los botones para ampliar y reducir nivel, **ShowLines** (Mostrar líneas) muestra selectivamente las líneas que relacionan el esquema y **ShowRoot** (Mostrar raíz) suprime las líneas del nivel superior.

- **Cuadros de fecha y hora.**

Este control es un cuadro de lista especial para introducir fechas y horas. Se trata de un componente específico localizado en la ficha **Win32** de la paleta **Component** y denominado **DateTimePicker** (Selector de fechas y horas).

TIPOS DE LISTAS

ListBox	**Muestra una lista de cadenas de texto de las cuales puede elegirse una o varias.**
ComboBox	**Muestra una lista desplegable de cadenas de texto al pulsar un botón.**
TreeView	**Visualiza elementos con estructura jerárquica.**
ListView	**Visualiza gráficamente elementos.**
ImageList	**Gestiona listas de imágenes.**
CheckListBox	**Muestra listas con casillas de verificación precediendo a cada elemento.**
DateTimePicker	**Muestra un cuadro de diálogo especializado para introducir fechas y horas.**

AGRUPACIONES

Borland Delphi permite agrupar la información de distintos modos.

- **Cuadros de agrupación.**

Los componentes más habitualmente utilizados son las casillas de selección, teniendo como propiedad común

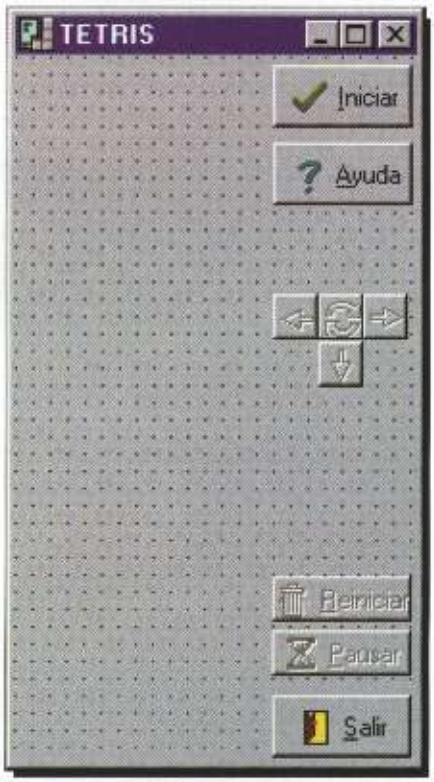


FIG. 15 Aspecto final del área de trabajo de la aplicación tras disponer todos los botones sobre la ficha.

TIPOS DE AGRUPACIONES

GroupBox	**Cuadro de grupo estándar con título.**
RadioGroup	**Grupo sencillo de casillas de selección.**
Panel	**Grupo de controles de disposición variable.**
ScrollBox	**Zona desplegable con controles.**
TabControl	**Conjunto de pestañas excluyentes.**
PageControl	**Conjunto de pestañas excluyentes con controles.**
HeaderControl	**Títulos para las columnas de datos.**

TIPOS DE INFORMACIÓN DE ESTADO

Label	**Cadenas de texto no editables.**
ProgressBar	**Barra de progreso en porcentaje.**
StatusBar	**Barra de estado en el margen inferior de la ventana.**
StaticText	**Texto no editable con manejador de ventana.**
Hint / ShowHint	**Ayuda emergente de herramientas**
HelpContext	**Enlace con el sistema de ayuda.**

COMPONENTES DE VISUALIZACIÓN TABULAR

DrawGrid	**Visualiza una estructura de datos existente.**
StringGrid	**Visualiza cadenas de texto.**
DBGrid	**Visualiza el contenido de un conjunto de datos.**

Caption como texto identificativo del grupo. Tras ubicar en la ficha el componente de agrupación, pueden disponerse sobre el mismo los componentes que se deseen asociar.

- **Casillas de selección.**

La propiedad **Caption** otorga el título al área rectangular. La propiedad **Items** da acceso al editor de listas de cadenas a asociar a las casillas de selección.

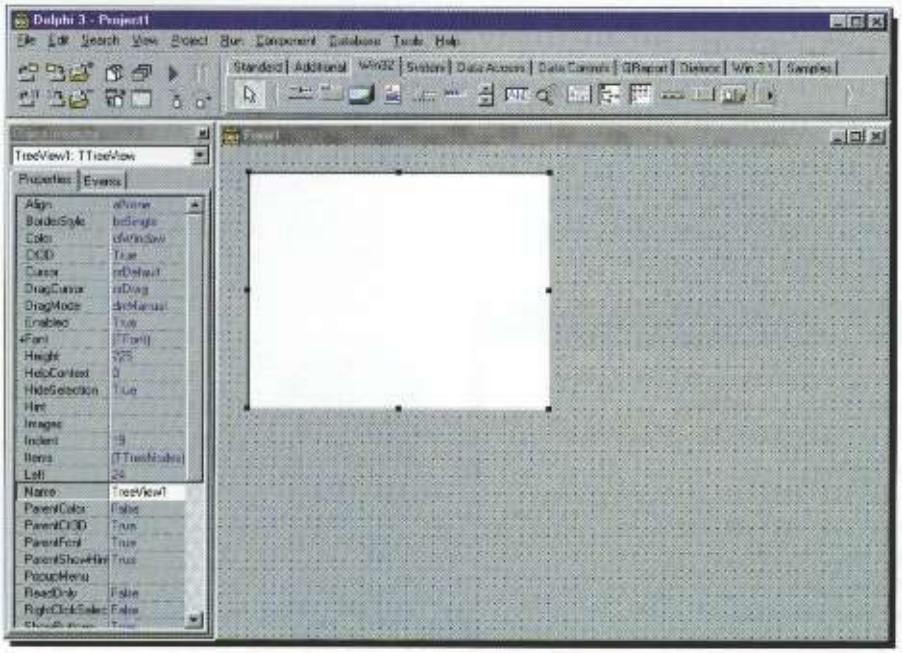


FIG. 16 El componente TreeView (Vista de árbol) permite incluir en las fichas un control capaz de mostrar datos con estructura jerárquica.

- **Paneles.**

Los paneles sirven de soporte físico a los controles que se desee alojar para componer, por ejemplo, barras de herramientas, a diferencia de los biseles, que tienen una misión puramente ornamental. La propiedad **BorderWidth** (Ancho del borde) establece el ancho del borde.

- **Controles de hoja.**

Este componente es la base para crear cuadros de diálogo de varias fichas e inicialmente muestra una hoja. La opción **New Page** (Nueva Página) del menú contextual permite crear nuevas hojas para alojar controles. La propiedad **ActivePage** (Página Activa) especifica el objeto, activo, **Multiline** (Multilínea) si pueden admitirse múltiples líneas, **PageCount** (Contador de página) indica el número de páginas, **TabHeight** (Altura de pestaña) define la altura de las pestañas y **TabWidth** (Anchura de pestaña) la anchura.

- **Controles de pestaña.**

Son los separadores que se emplean en el diseño de cuadros de diálogo de varias fichas.

- **Cuadros de desplazamiento.**

Estos componentes permiten ampliar la zona visible de las fichas cuando se necesita alojar un gran número de componentes, por medio de zonas desplegables que se pueden recorrer con ayuda de cuadros de desplazamiento. La propiedad **Kind** (Clase) determina si debe activarse el desplazamiento horizontal o vertical y **Align** (Alinear) facilita el ajuste a un margen de la ficha.

- **Controles de cabecera.**

Estos componentes permiten redimensionar las cabeceras de secciones

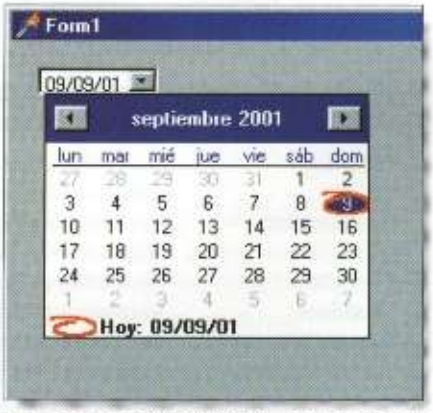


FIG. 17 El control DateTimePicker permite seleccionar fechas y horas mediante un cuadro de diálogo especial.

con el ratón. La propiedad **Sections** (Secciones) determina las secciones de cabecera.

INFORMACIÓN VISUAL

Son componentes específicos para mostrar información de estado referente al funcionamiento de la aplicación.

- **Etiquetas.**

Son etiquetas de texto utilizadas normalmente para identificar campos de datos y otros controles. La propiedad **Caption** contiene el texto de la etiqueta, **FocusControl** (Control del foco) permite redirigir el foco de la aplicación mediante una combinación de teclas asociada a un control y **Transparent** (Transparente) resulta útil para poder mostrar un gráfico con etiqueta superpuesta.

- **Barras de progreso.**

Estos controles muestran gráficamente el progreso porcentual de una tarea larga. La propiedad **Position** (Posición) define la posición por omisión del control, **Max** (Máximo) y **Min** (Mínimo) definen el rango de la barra y **StepBy** (Incremento) determina la tasa de incremento.

- **Barras de estado.**

Usualmente están relegadas al margen izquierdo de la ventana de la aplicación. La propiedad **Align** establece la posición y tamaño, **Panels** (Paneles) permite crear subpaneles y **Width** define el ancho de los mismos.

- **Ayuda emergente.**

Los controles que muestran información de ayuda o sugerencias tienen propiedades comunes. **Hint** (Ayuda) es el texto que se muestra al usuario al mantener el puntero del ratón sobre el control, que se debe activar a su vez mediante la propiedad **ShowHint** (Mostrar ayuda); **ParentShowHint** (Mostrar ayuda padre) resulta útil para activar la propiedad para conjuntos de controles; y **HelpContext** (Ayuda contextual) establece el número de ayuda contextual para el control.

PERSONALIZACIÓN DE PALETAS

El contenido de la paleta Component puede personalizarse. Así, es posible ocultar, reorganizar o cambiar el nombre de los componentes, instalar componentes adicionales e incluso crear nuevas plantillas de componente para añadirlas a la paleta.

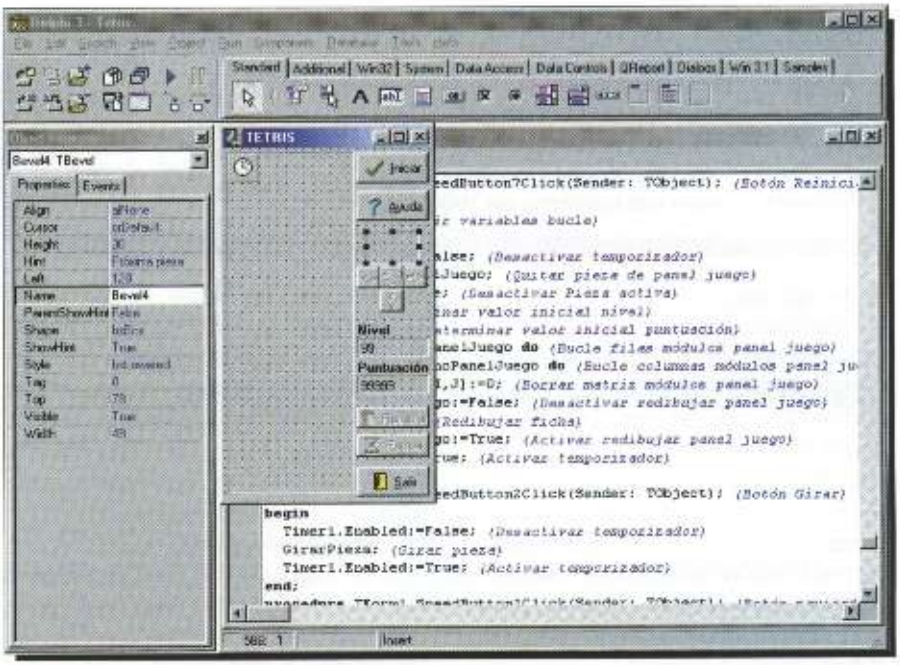


FIG. 18 La ventana de la aplicación de ejemplo sobre la cual se desplazan las piezas es un bisel, así como el área que muestra la próxima pieza y los espacios para mostrar el nivel y puntuación.

VISUALIZACIÓN TABULAR

Los controles denominados *de rejilla* son los responsables de la presentación de información en forma tabular, similar a la característica de una hoja de cálculo, por ejemplo.

- **Rejillas para datos.**

Estos controles son capaces de mostrar información como columnas de datos. La propiedad **DefaultDrawing** (Dibujo por omisión) controla si se dibujará automáticamente el contenido de las celdas o como consecuencia de lo indicado por el usuario mediante el manejador de sucesos; **CellRect** (Recta celda) obtiene el área de dibujo de la celda; **Selection** (Selección) indica la celda seleccionada de la rejilla; **Options** (Opciones) permite modificar el aspecto y comportamiento del control, como por ejemplo, especificando el comportamiento de la tecla **Tab**, o mostrando selectivamente las líneas de separación de la rejilla en horizontal o vertical; **DefaultColWidth** (Ancho de columna por omisión) establece el ancho de columna por omisión; **DefaultRowHeight** (Alto de fila por omisión) establece el alto de fila por omisión; **ColWidths** (Ancho de columna) modifica el ancho de columna; **RowHeights** (Alto de fila) modifica el alto de fila. También existen propiedades para fijar el ancho y alto de columnas, y filas, indicar colores e incluso añadir barras de desplazamiento para permite visualizar datos cuando no pueden mostrarse en el espacio habilitado al efecto en la ficha, disponiendo igualmente de las propiedades, métodos y sucesos propios de los controles de ventana.

- **Rejillas para cadenas de caracteres.**

Este componente es una versión especializada para cadenas de caracteres del control genérico anterior. La propiedad **Cells** (Celdas) puede emplearse para acceder a cualquier celda de la rejilla, así como **Objects** (Objetos) para acceder a los datos almacenados en la posición indicada, **Cols** (Columnas) permite acceder a todos los objetos asociados a una columna, **Rows** (Filas) permite acceder a todos los objetos asociados a una fila, disponiendo igualmente de las propiedades, métodos y sucesos propios de los controles de ventana.

VISUALIZACIÓN GRÁFICA

Borland Delphi cuenta también con un conjunto de controles especializados para mostrar gráficos.

COMPONENTES DE VISUALIZACIÓN GRÁFICA

Image	**Visualiza el contenido de un archivo gráfico.**
Shape	**Visualiza una forma geométrica.**
Bevel	**Visualiza líneas y marcos tridimensionales.**
PaintBox	**Visualiza gráficos programados.**
Animate	**Visualiza archivos en formato AVI.**

- **Imágenes.**

Este componente muestra una imagen de mapa de bits, icono o metaarchivo en la ficha, que puede provenir tanto de la propia biblioteca de Borland Delphi como de cualquier otra fuente elegida por el usuario.

La propiedad **Picture** (Imagen) permite elegir el archivo de imagen, **Center** (Centro) indica la alineación sobre el componente y **Stretch** (Ajustar) vincula la imagen al componente de modo que se amplíe o reduzca su tamaño de modo conjunto.

- **Formas.**

Muestra formas geométricas. La propiedad **Shape** (Forma) indica la forma geométrica, **BorderColor** (Color del borde) el color del borde y **Brush** (Brocha) define el color de la forma.

- **Biseles.**

Este componente permite añadir marcos biselados para dibujar líneas y marcos sobre las fichas. Las propiedades **BevelInner** (Bisel interior) y **BevelOuter** (Bisel exterior) definen las características de los dos biseles, que pueden adoptar las modalidades de relieve, relieve inverso o sin relieve; **BevelWidth** (Ancho de bisel) especifica el espacio de separación entre biseles.

- **Cuadros de dibujo.**

Este componente habilita un área restringida de la ficha en la cual puede dibujarse libremente utilizando código Object Pascal, denominándose **Canvas** a esta área y pudiendo utilizarse el manejador de sucesos **OnPaint**.

- **Control de animación.**

Permite visualizar archivos de video en formato AVI (sin sonido). La propiedad **ResHandle** (Manejador del recurso) es el manejador de Windows para el módulo que contiene el clip AVI como recurso; **ResId** (Identificador de recurso) y **ResName** (Nombre del recurso) permiten especificar qué recurso del módulo indicado es el clip AVI que se desea visualizar, **AutoSize** (Dimensionamiento automático) fuerza al control a ajustar automáticamente el tamaño de la ventana de visualización, **StartFrame** (Fotograma inicial) y **StopFrame** (Fotograma final) determinan los puntos de inicio y fin de la reproducción, **CommonAVI** (AVI estándar) permite visualizar los clips estándar de Windows incluidos en **Shell32.dll**, **Active** (Activo) permite detener y reanudar la animación, **Repetitions** (Repeticiones) fijar el número de iteraciones y **Timers** (Temporizadores) visualizar los marcos mediante un temporizador para permitir sincronizar la secuencia con otras acciones en desarrollo, como por ejemplo la sincronización de la reproducción de la banda sonora.

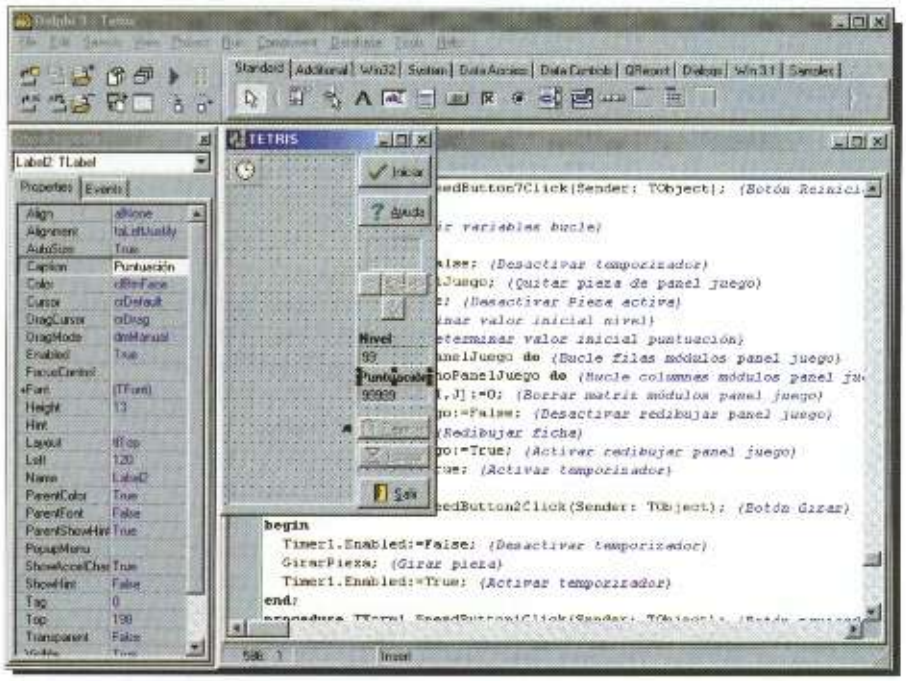


FIG. 19 El componente Label permite mostrar textos sobre las fichas que sirvan de orientación para la introducción de datos o para la interpretación de la información mostrada.

CUADROS DE DIÁLOGO ESTÁNDAR DE WINDOWS

Los componentes propios de los cuadros de diálogo estándar de Windows se agrupan en la ficha **Dialogs** (Diálogos) y permiten mostrar cuadros de diálogo con la interfaz habitual de Windows a petición del método **Execute** (Ejecutar), que ofrece como resultado los valores verdadero o falso según el usuario haya pulsado el botón para aceptar o para cancelar.

RESUMEN

Mediante el inicio del desarrollo de un nuevo proyecto de ejemplo, esta vez con una versión del popular juego del Tetris, profundizamos en esta entrega en el manejo de componentes para su inclusión en las fichas. Se dan también las directrices sobre agrupación de componentes para construir controles complejos. Analizamos, finalmente, los distintos tipos de componentes con que cuenta Borland Delphi, organizados atendiendo a su clase y ámbito de aplicación, con la intención de dar una idea de la funcionalidad con que se puede dotar a los controles alojados en las fichas.