Microsoft Small Basic

Introducción a la programación

Capítulo 1

Introducción

Small Basic y la programación

En informática, se define la programación como el proceso de crear aplicaciones para ordenador utilizando lenguajes de programación. De la misma manera que los humanos hablamos y entendemos inglés, español o francés, los ordenadores entienden programas escritos en ciertos idiomas. Estos se denominan lenguajes de programación. Al principio sólo había unos pocos lenguajes de programación y eran muy fáciles de aprender y entender. Pero a medida que los ordenadores y el software se han vuelto cada vez más sofisticados, los lenguajes de programación evolucionaron rápidamente, volviéndose cada vez más complejos. El resultado es que los más modernos lenguajes de programación y sus conceptos son bastante difíciles de comprender para un principiante. Esto ha desalentado a muchas personas a aprender o intentar programar.

Small Basic es un lenguaje de programación que está diseñado para hacer la programación muy sencilla, accesible y divertida para los principiantes. El objetivo de Small Basic es romper cualquier barrera y servir de piedra angular para el increíble mundo de la programación.

El entorno de Small Basic

Empecemos con una breve introducción al entorno de Small Basic. Cuando lance SB.exe por primera vez, se encontrará con una ventana parecida a la siguiente figura.

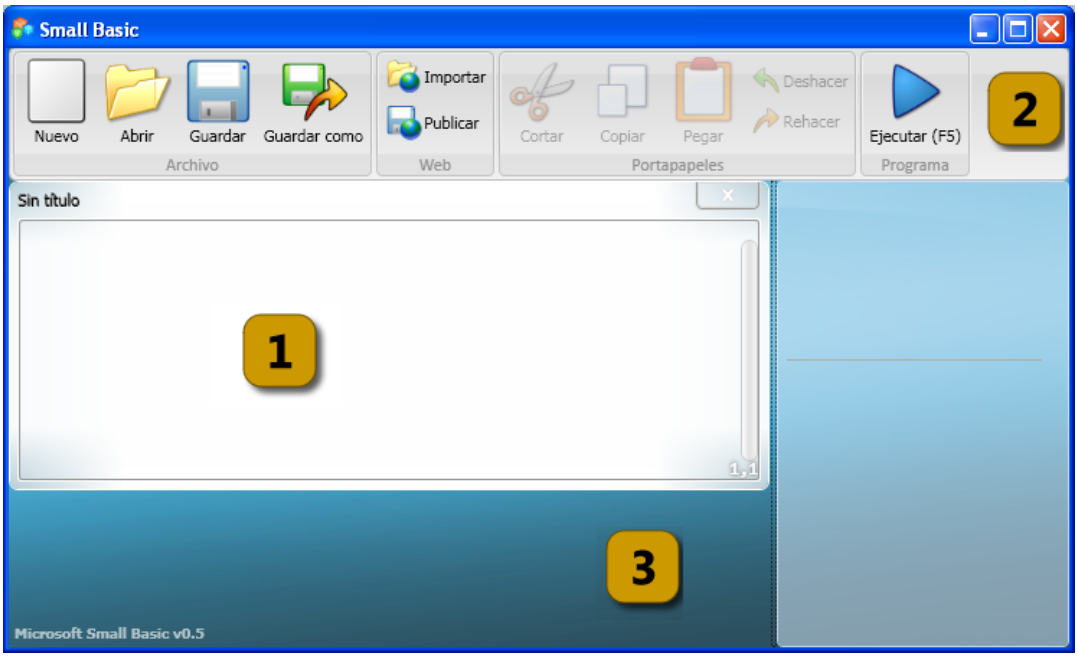

Figura 1 - El entorno de Small Basic

Este es el entorno de Small Basic, donde escribiremos y ejecutaremos los programas de Small Basic. Este entorno contiene varios elementos distintos identificados con números.

El **Editor**, identificado con [1], es el lugar donde escribiremos nuestros programas de Small Basic. Cuando abra un programa de ejemplo u otro guardado previamente, aparecerá en el editor. Una vez abierto, puede modificarlo y guardarlo para usos posteriores.

También puede abrir y trabajar con más de un programa a la vez. Cada programa aparecerá en un editor distinto. El editor que contiene el programa en el que está trabajando se denomina *editor activo*.

La **Barra de herramientas**, identificada con [2], se utiliza para emitir comandos bien al *editor activo* o al entorno. Veremos los distintos comandos de la barra de herramientas según vayamos avanzando.

La **Superficie**, identificada con [3], es el lugar donde se encuentran todas las ventanas del editor.

Nuestro primer programa

Ahora que ya se encuentra familiarizado con el entorno de Small Basic, empecemos a programar en él. Como se indicó anteriormente, el editor es el lugar donde escribiremos nuestros programas. Escriba la siguiente línea en el editor.

```
TextWindow.WriteLine("Hola mundo")
```

Este es nuestro primer programa de Small Basic. Si lo ha tecleado correctamente, verá algo parecido a la siguiente figura.

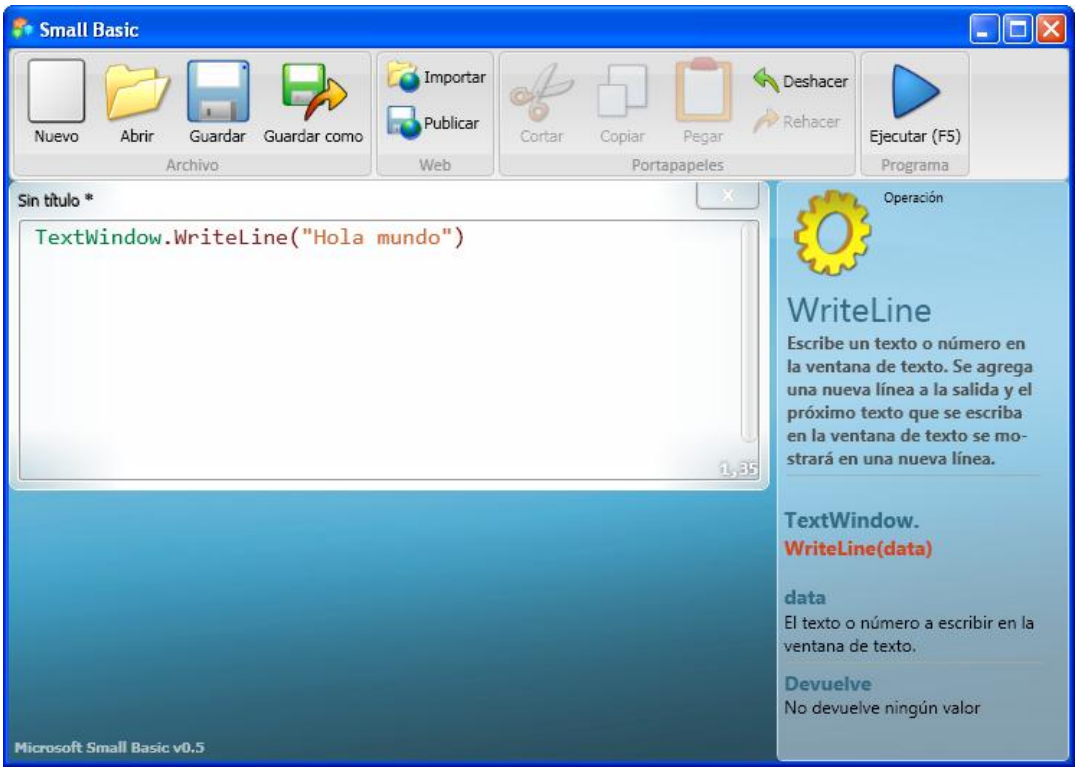

Figura 2 - Nuestro primer programa

Ahora que hemos escrito nuestro primer programa, vamos a ejecutarlo y ver qué pasa. Podemos ejecutar un programa bien haciendo clic en el botón **Ejecutar** de la barra de herramientas o utilizando la tecla de método abreviado F5 del teclado. Si todo va bien, nuestro programa debería ejecutarse con el resultado siguiente.

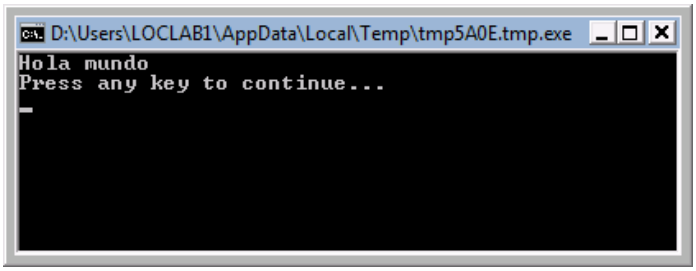

Figura 3 - Nuestro primer resultado

¡Enhorabuena! Acaba de escribir y ejecutar su primer programa de Small Basic. Es cierto que es un programa pequeño y sencillo, ¡pero es un gran paso para convertirse en un verdadero programador informático! Sólo un detalle más antes de crear programas más grandes. Necesita entender qué es lo que hemos hecho. ¿Qué le hemos dicho al ordenador que haga y cómo sabía

*Mientras tecleaba su primer programa, puede que se haya dado cuenta que le ha aparecido un menú contextual con una lista de elementos (*Figura 4*). Esto se llama "Intellisense" y le ayuda a programar más rápido. Puede ir a través de la lista presionando las teclas de las fechas arriba y abajo. Cuando encuentre lo que desee, presione la tecla Entrar para insertar el elemento seleccionado en su programa.*

el ordenador qué hacer? En el siguiente capítulo analizaremos el programa que acabamos de escribir para poder alcanzar dicho entendimiento.

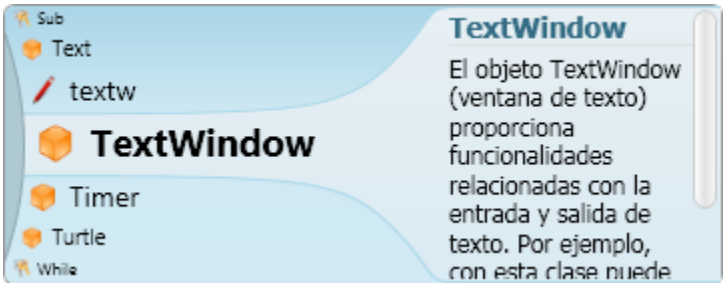

Figura 4 - Intellisense

Guardando nuestro programa

Si desea cerrar Small Basic y volver más tarde al programa que acaba de escribir, tiene que guardarlo. Es una buena práctica guardar los programas de vez en cuando, para no perder ninguna información en el caso de que el ordenador se apague accidentalmente o haya un corte de luz. Puede guardar el programa actual haciendo clic en el icono **Guardar** de la barra de herramientas o utilizando el método abreviado "CTRL+S" (pulse la tecla "S" mientras mantiene pulsada la tecla CTRL).

Capítulo 2

Entendiendo nuestro primer programa

¿Qué es realmente un programa de ordenador?

Un programa es un conjunto de instrucciones para el ordenador. Dichas instrucciones indican con precisión al ordenador lo que hacer y el ordenador siempre las sigue. Al igual que la gente, los ordenadores sólo siguen instrucciones si se especifican en un idioma o lenguaje que puedan entender. Estos se denominan lenguajes de programación. Hay muchos lenguajes que los ordenadores entienden y **Small Basic** es uno de ellos.

Imagine una conversación entre usted y un amigo. Usted y su amigo utilizan palabras, organizadas en frases para intercambiar información. De la misma manera, los lenguajes de programación contienen conjuntos de palabras que pueden organizarse en frases para transmitir información al ordenador. Los programas son básicamente grupos de frases (algunas veces unas pocas y otras miles) que, juntas, tienen sentido por igual tanto para el programador como para el ordenador.

Hay muchos lenguajes que los ordenadores entienden. Java, C++, Python, VB, etc. son todos potentes lenguajes de programación modernos que pueden utilizarse para crear tanto programas simples como complejos.

Los programas de Small Basic

Un programa típico de Small Basic consiste en un conjunto de *instrucciones*. Cada línea del programa representa una instrucción y cada instrucción es una orden para el ordenador. Cuando pedimos al ordenador que ejecute un programa de Small Basic, lee la primera instrucción del programa. Entiende lo que le pedimos y ejecuta la instrucción. Una vez que ha ejecutado la primera instrucción, vuelve al programa para leer y ejecutar la segunda instrucción. Y así continúa hasta alcanzar el final del programa. Es entonces cuando nuestro programa termina.

Un repaso a nuestro primer programa

He aquí el primer programa que escribió:

```
TextWindow.WriteLine("Hola mundo")
```

Es un programa muy simple que consiste en una única *instrucción*. Dicha instrucción pide al ordenador que escriba una línea de texto (*WriteLine*) que es **Hola mundo** en la ventana de texto (*TextWindow*).

Esta instrucción se traduce en el ordenador como:

```
Escribir (Write)  Hola mundo
```

Probablemente se haya dado cuenta que la instrucción se puede dividir en segmentos más pequeños de la misma manera que las frases se pueden dividir en palabras. La primera instrucción contiene 3 segmentos distintos:

a) *TextWindow*
b) *WriteLine*
c) "Hola mundo"

El punto, los paréntesis y las comillas son signos de puntuación que hay que poner en lugares concretos en la instrucción para que el ordenador pueda entender nuestras intenciones.

Probablemente usted recuerde que una ventana de color negro apareció cuando ejecutamos nuestro primer programa. La ventana de color negro se llama *TextWindow* aunque a veces también se denomina la *consola*. Dicha ventana es el lugar donde se muestra el resultado del programa. En nuestro programa, **TextWindow** se denomina *objeto*. Un buen número de objetos se encuentran disponibles para utilizar en sus programas. Podemos realizar varias *operaciones* con dichos objetos. Ya hemos utilizado la operación *WriteLine* en nuestro programa. Probablemente también se haya dado cuenta de que a la operación *WriteLine* le sigue **Hola mundo** entre comillas. Este texto se pasa como entrada a la operación *WriteLine*. Esto se denomina *entrada* de la operación. Algunas operaciones admiten una o más entradas mientras que otras no admiten ninguna.

> *Los signos de puntuación tales como comillas, espacios y paréntesis son muy importantes en un programa de ordenador. Pueden cambiar el significado basados en su posición y número.*

Nuestro segundo programa

Ahora que ya hemos examinado nuestro primer programa, vamos a darle una nota de color.

```
TextWindow.ForegroundColor = "Yellow"
TextWindow.WriteLine("Hola mundo")
```

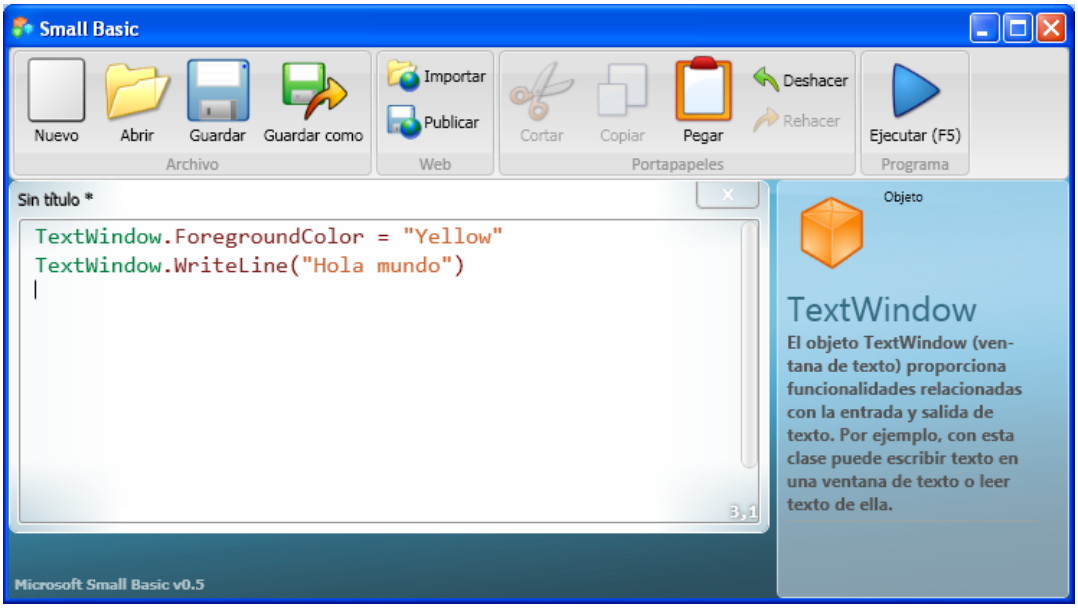

Figura 5 - Agregar colores

Cuando ejecute el programa anterior, notará que imprime el mismo mensaje que antes ("Hola mundo") dentro de la ventana de texto (*TextWindow*), pero esta vez la imprime en amarillo (*Yellow*) en lugar de en gris.

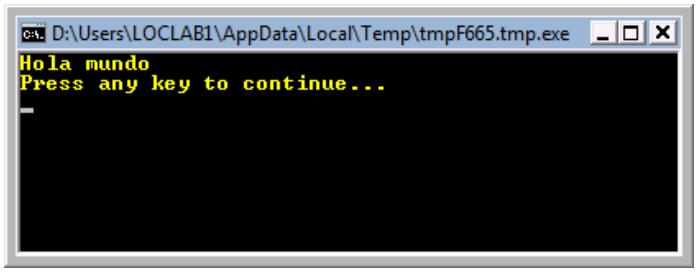

Figura 6 - Hola mundo en amarillo

Para ello, hemos agregado una nueva instrucción al programa original. Utiliza una nueva palabra *ForegroundColor* (color de fondo) al que se le asigna *"Yellow"* (amarillo). Esto significa que se asigna *"Yellow"* a *ForegroundColor*. La diferencia entre las operaciones *ForegroundColor* y *WriteLine* es que *ForegroundColor* no admite ningún parámetro y, por tanto, no necesita paréntesis. En su lugar, le sigue un símbolo *igual que* (=) y una palabra. *ForegroundColor* se define como una *propiedad* de *TextWindow*. He aquí una lista de valores que son válidos para la propiedad *ForegroundColor*. Reemplace "Yellow" con uno de los siguientes valores y observe los resultados. No se olvide de las comillas ya que son imprescindibles.

```
Black
Blue
Cyan
Gray
Green
Magenta
Red
White
Yellow
DarkBlue
DarkCyan
DarkGray
DarkGreen
DarkMagenta
DarkRed
DarkYellow
```

Capítulo 3

Introducción a las variables

Usar variables en nuestro programa

Sería bueno que nuestro programa pudiera decir "Hola" con el nombre real del usuario, en lugar de decir genéricamente "Hola mundo", ¿no? Para lograr esto, primero debemos preguntar al usuario su nombre, luego almacenarlo en algún lugar y, finalmente, imprimir "Hola" seguido del nombre del usuario. Vamos a ver cómo podemos hacerlo:

```
TextWindow.Write("Escriba su nombre: ")
nombre = TextWindow.Read() 'Leer
TextWindow.WriteLine("Hola " + nombre)
```

Cuando escriba y ejecute este programa, verá una salida como la siguiente:

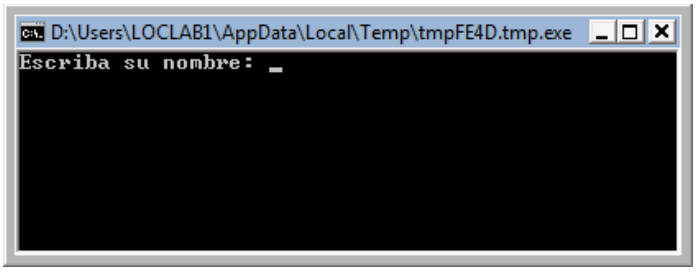

Figura 7 - Preguntar el nombre del usuario

Y cuando escriba su nombre y oprima la tecla Entrar, verá la siguiente salida:

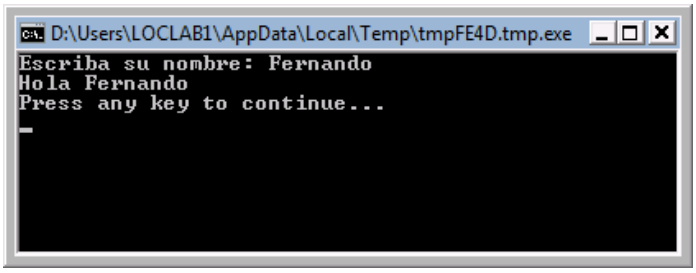

Figura 8 – Un cálido saludo

Ahora, si ejecuta el programa otra vez, se le hará la misma pregunta nuevamente. Puede escribir un nombre diferente y el equipo dirá *Hola* a ese nombre.

Análisis del programa

En el programa que acaba de ejecutar, la línea que puede haber captado su atención es esta:

```
nombre = TextWindow.Read()
```

Read es igual que *WriteLine*, pero sin entradas. Es una operación y básicamente dice al equipo que espere que el usuario escriba algo y presione la tecla Entrar.

> Write *(escribir)y WriteLine son operaciones de TextWindow. Write le permite escribir algo en la TextWindow y que el siguiente texto se escriba en la misma línea que el texto actual.*

Una vez que el usuario oprime Entrar, toma lo que el usuario haya escrito y lo devuelve al programa. El punto interesante es que lo que haya escrito el usuario está ahora almacenado en una *variable* llamada **nombre**. Una *variable* se define como un lugar donde puede guardar valores temporalmente y usarlos más tarde. En la línea de arriba, la variable **nombre** fue usada para guardar el nombre del usuario.

La siguiente línea también es interesante:

```
TextWindow.WriteLine("Hola " + nombre)
```

Este es el lugar donde usamos el valor almacenado en nuestra variable, **nombre**. Tomamos el valor de **nombre**, lo agregamos a continuación de "Hola" y lo escribimos en la *TextWindow*.

Una vez que la variable está asignada, es decir, cuando se guarda un valor en una variable, puede volver a utilizarla cuantas veces quiera. Por ejemplo, puede hacer lo siguiente:

```
TextWindow.Write("Escriba su nombre: ")
nombre = TextWindow.Read()
TextWindow.Write("Hola " + nombre + ". ")
TextWindow.WriteLine("¿Cómo está " + nombre + "?")
```

Y verá la siguiente salida:

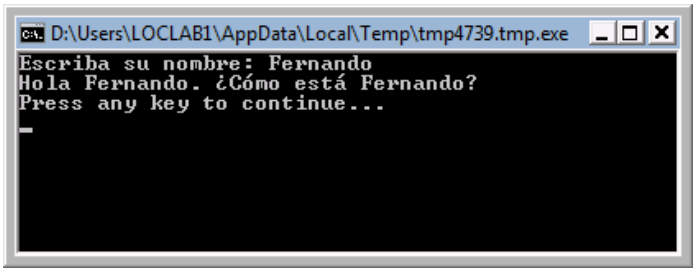

Figura 9 – Reutilizando una variable

Jugando con los números

Acabamos de ver cómo puede usar variables para almacenar el nombre del usuario. En los siguientes programas, verá cómo puede almacenar y manipular números en variables. Vamos a comenzar con un programa realmente simple:

```
número1 = 20
número2 = 10
número3 = número1 + número2
TextWindow.WriteLine(número3)
```

Cuando ejecute este programa obtendrá lo siguiente como salida:

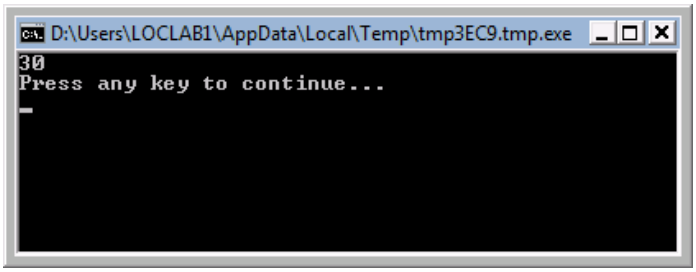

Figura 10 – Sumando dos números

En la primera línea del programa, se asigna a la variable **número1** un valor de 20. Y en la segunda línea, se asigna a la variable **número2** un valor de 10. En la tercera línea, se suma **número1** y **número2**, y se asigna el resultado de la suma a **número3**. Por tanto, la variable **número3** tendrá un valor de 30. Y eso es lo que imprimimos en la *TextWindow*.

> *Note que los números no están rodeados por comillas. Para los números, las comillas no son necesarias. Necesita comillas sólo cuando está usando texto.*

Ahora vamos a modificar ligeramente este programa y veamos los resultados:

```
número1 = 20
número2 = 10
número3 = número1 * número2
TextWindow.WriteLine(número3)
```

El programa de arriba multiplica **número1** por **número2** y almacena el resultado en **número3**. Puede ver el resultado de este programa aquí debajo:

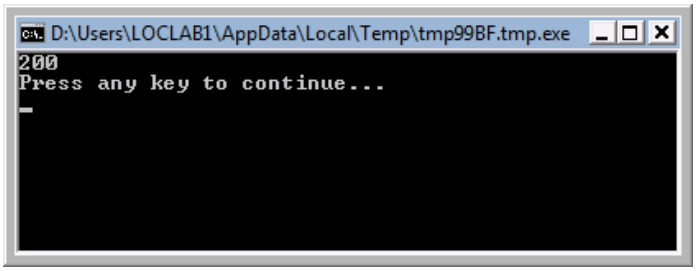

Figura 11 – Multiplicando dos números

De la misma forma, puede restar o dividir dos números. Aquí está la resta:

```
número3 = número1 - número2
```

El símbolo para la división es '/'. El programa resultará similar a este:

```
número3 = número1 / número2
```

Y el resultado de la división será:

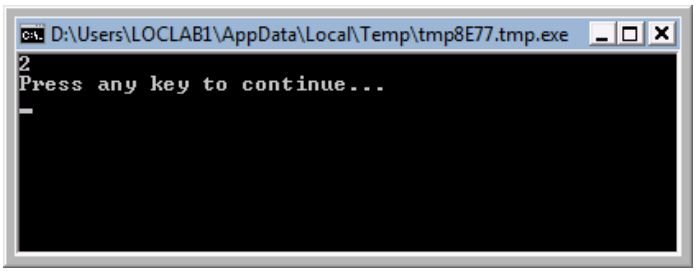

Figura 12 – Dividiendo dos números

Un convertidor de temperatura simple

Para el siguiente programa usaremos la fórmula $℃ = \frac{5(℉-32)}{9}$ para convertir temperaturas en Fahrenheit a temperaturas en Celsius.

Primero, obtendremos la temperatura en Fahrenheit del usuario y la almacenaremos en una variable. Hay una operación especial que nos permite leer los números que escribe el usuario y es *TextWindow.ReadNumber* (leer número).

```
TextWindow.Write("Introduczca la temperatura en Fahrenheit: ")
fahr = TextWindow.ReadNumber()
```

Una vez que tenemos la temperatura en Fahrenheit almacenada en una variable, podemos convertirla a Celsius así:

```
celsius = 5 * (fahr - 32) / 9
```

Los paréntesis dicen al equipo que calcule la parte **fahr – 32** primero y luego procese el resto. Ahora todo lo que tenemos que hacer es imprimir el resultado al usuario. Poniendo junto todo esto, obtenemos este programa:

```
TextWindow.Write("Introduczca la temperatura en Fahrenheit: ")
fahr = TextWindow.ReadNumber()
celsius = 5 * (fahr - 32) / 9
TextWindow.WriteLine("La temperatura en Celsius es " + celsius)
```

Y el resultado de este programa es:

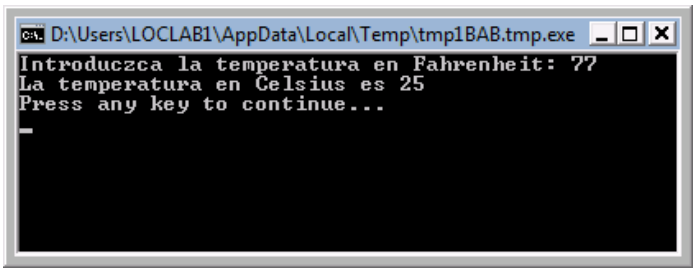

Figura 13 – Conversión de temperatura

Capítulo 4

Condiciones y bifurcaciones

Volviendo a nuestro primer programa, ¿no sería bueno que en lugar del genérico *Hola mundo*, pudiéramos decir "Buenos días mundo" o "Buenas tardes mundo", dependiendo de la hora del día? Para nuestro próximo programa, haremos que el equipo diga "Buenos días mundo" si la hora es anterior a las 12:00 y "Buenas tardes mundo" si la hora es posterior a las 12:00.

```
If (Clock.Hour < 12) Then
 TextWindow.WriteLine("Buenos días mundo")
EndIf
If (Clock.Hour >= 12) Then
 TextWindow.WriteLine("Buenas tardes mundo")
EndIf
```

Dependiendo de cuándo ejecute el programa verá una de las dos siguientes salidas:

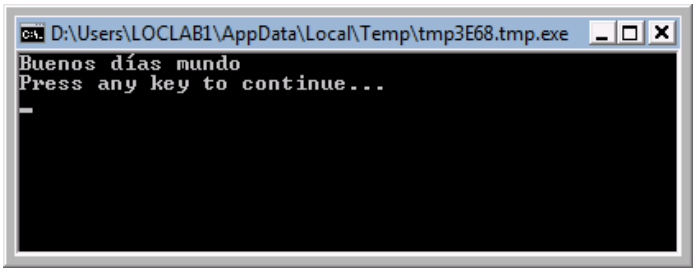

Figura 14 – Buenos días mundo

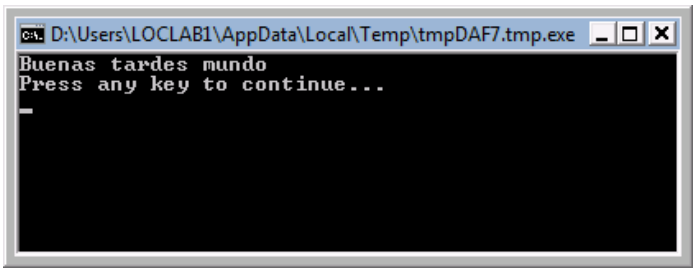

Figura 15 – Buenas tardes mundo

Vamos a analizar las primeras tres líneas del programa. Ya se habrá imaginado que estas líneas le dicen al equipo que si *Clock.Hour* es menor que 12, imprima "Buenos días mundo". Las palabas **If**, **Then** y **EndIf** son palabras especiales que comprende el equipo cuando ejecuta el programa. La palabra **If** (si…) siempre viene seguida de una condición, que en este caso es (**Clock.Hour < 12)**. Recuerde que los paréntesis son necesarios para que el equipo entienda sus intenciones. La condición es seguida por **Then** (entonces…) y la operación real a ejecutar. Y después de la operación viene **EndIf** (final de la condición If). Esto indica al equipo que la ejecución condicional ha terminado.

En Small Basic, puede usar el objeto Clock *(Reloj) para tener acceso a la fecha y hora actuales. También provee una cantidad de propiedades que permiten obtener el día (*Day*), mes (*Month*), año (*Year*), hora (*Hour*), minutos (*Minutes*) y segundos (*Seconds*) por separado.*

Entre **Then** y **EndIf**, puede haber más de una operación y el equipo las ejecutará todas si se cumple la condición. Por ejemplo, podría haber escrito algo como esto:

```
If (Clock.Hour < 12) Then
 TextWindow.Write("Buenos días. ")
 TextWindow.WriteLine("¿Cómo estuvo el desayuno?")
EndIf
```

Else

Puede que se haya dado cuenta que en el programa al comienzo de este capítulo, la segunda condición es redundante. El valor de *Clock.Hour* puede ser menor que 12 o no y sólo necesitamos hacer el segundo control. En casos como este, podemos abreviar las dos instrucciones **If**…**Then**…**EndIf** en una sola usando una nueva palabra, **Else** (en caso lo contrario).

Si fuéramos a escribir el mismo programa usando **Else**, así es como quedaría:

```
If (Clock.Hour < 12) Then
 TextWindow.WriteLine("Buenos días mundo")
Else
 TextWindow.WriteLine("Buenas tardes mundo")
EndIf
```

Y este programa hará exactamente lo mismo que el otro, lo que nos lleva a una lección muy importante en programación de ordenadores:

> *En programación, habitualmente hay muchas formas de hacer lo mismo. Algunas veces una forma tiene más sentido que la otra. La elección es del programador. A medida que escriba más programas y se vuelva más experimentado, comenzará a notar las diferentes técnicas y sus ventajas y desventajas.*

Sangría

En todos los ejemplos puede ver que las instrucciones entre **If**, **Else** y **EndIf** están sangradas. Este sangrado no es necesario. El equipo entenderá el programa perfectamente sin él. Sin embargo, las sangrías ayudan a ver y entender la estructura del programa más fácilmente. Por lo tanto, se considera una buena práctica sangrar las instrucciones entre esos bloques.

Par o impar

Ahora que tenemos las instrucciones **If…Then…Else…EndIf** en nuestra caja de herramientas, vamos a escribir un programa que, dado un número, nos diga si es par o impar.

```
TextWindow.Write("Introduzca un número: ")
número = TextWindow.ReadNumber()
resto = Math.Remainder(número, 2)
If (resto = 0) Then
 TextWindow.WriteLine("El número es par")
Else
 TextWindow.WriteLine("El número es impar")
EndIf
```

Y cuando ejecute este programa, verá una salida como:

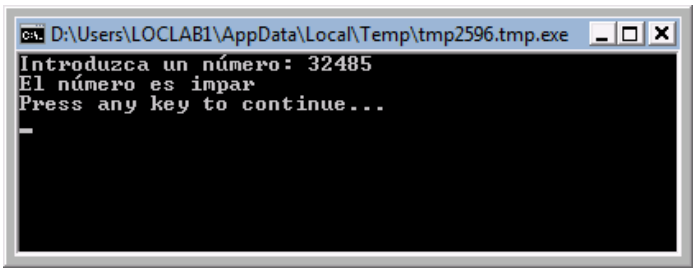

Figura 16 – Par o impar

En este programa hemos introducido una nueva operación muy útil, *Math.Remainder* (resultado de la división). Efectivamente, como ya se podrá haber imaginado, *Math.Remainder* dividirá el primer número por el segundo número y devolverá el resultado.

Bifurcaciones

Recuerde que en el segundo capítulo aprendimos que el equipo procesa un programa una instrucción cada vez de arriba a abajo. No obstante, hay una instrucción especial que puede hacer que el equipo salte a otra instrucción fuera de orden. Echemos un vistazo al siguiente programa.

```
i = 1
comienzo:
TextWindow.WriteLine(i)
i = i + 1
If (i < 25) Then
 Goto comienzo
EndIf
```

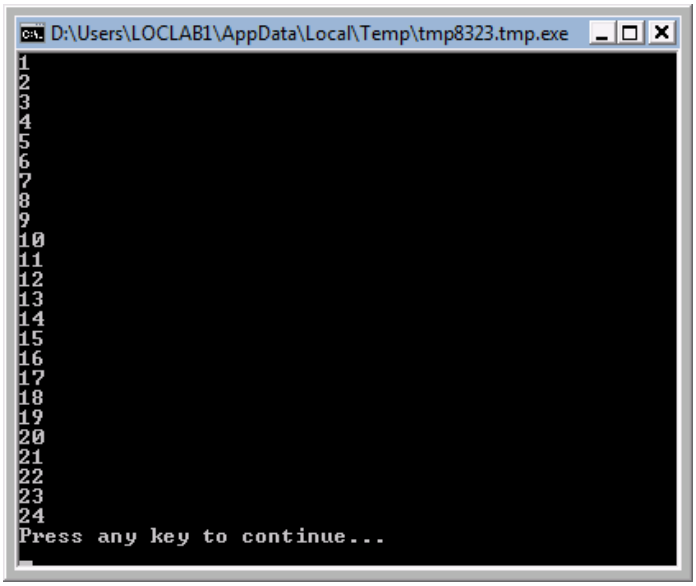

Figura 17 - Usando Goto

En el programa anterior, asignamos un valor 1 a la variable **i**. Y luego agregamos una nueva instrucción que termina en dos puntos (:)

```
comienzo:
```

Esto es una *etiqueta*. Las etiquetas son marcadores que el equipo entiende. Puede darle al marcador cualquier nombre y puede agregar tantas etiquetas como desee en su programa, siempre que los nombres sean únicos.

Aquí hay otra instrucción interesante:

```
i = i + 1
```

Esto dice al equipo que sume 1 a la variable **i** y asigne el resultado nuevamente a **i**. Por lo tanto, si el valor de **i** era 1 antes de esta instrucción, será 2 una vez que se ejecute esta instrucción.

Y finalmente:

```
If (i < 25) Then
 Goto comienzo
EndIf
```

Esta es la parte que le dice al equipo que si el valor de **i** es menor que 25, comience a ejecutar las instrucciones desde el marcador **comienzo**.

Ejecución eterna

Usando la instrucción **Goto** puede hacer que el equipo repita algo cualquier número de veces. Por ejemplo, puede modificar el programa de pares e impares como aparece más abajo y el programa se ejecutará para siempre. Puede detener el programa haciendo clic en el botón *Cerrar* (X) en la esquina superior derecha de la ventana.

```
comienzo:
TextWindow.Write("Introduzca un número: ")
número = TextWindow.ReadNumber()
resto = Math.Remainder(número, 2)
If (resto = 0) Then
 TextWindow.WriteLine("El número es par")
Else
 TextWindow.WriteLine("El número es impar")
EndIf
Goto comienzo
```

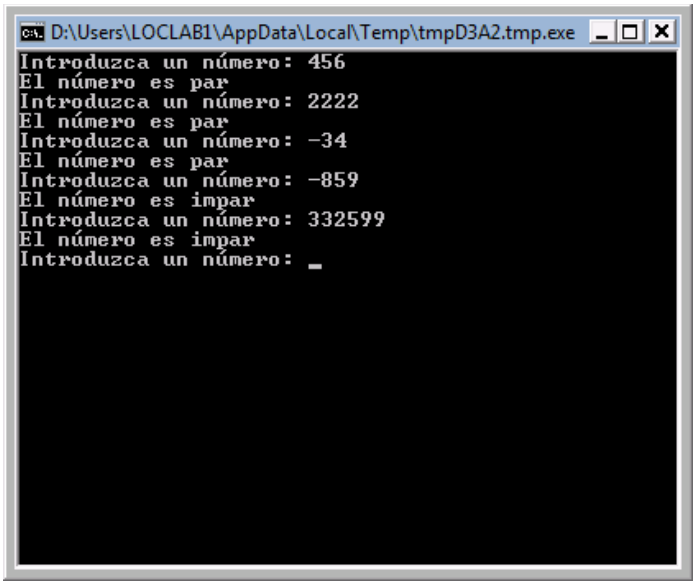

Figura 18 – Par o impar ejecutado eternamente

Capítulo 5

Bucles

El bucle For

Tomemos el programa que escribimos en el capítulo anterior.

```
i = 1
comienzo:
TextWindow.WriteLine(i)
i = i + 1
If (i < 25) Then
 Goto comienzo
EndIf
```

Este programa imprime en orden los números del 1 al 24. Este proceso de incrementar una variable es tan común en programación que algunos lenguajes proveen un método más fácil de hacerlo. El programa anterior es equivalente al siguiente programa:

```
For i = 1 To 24
 TextWindow.WriteLine(i)
EndFor
```

Y la salida es:

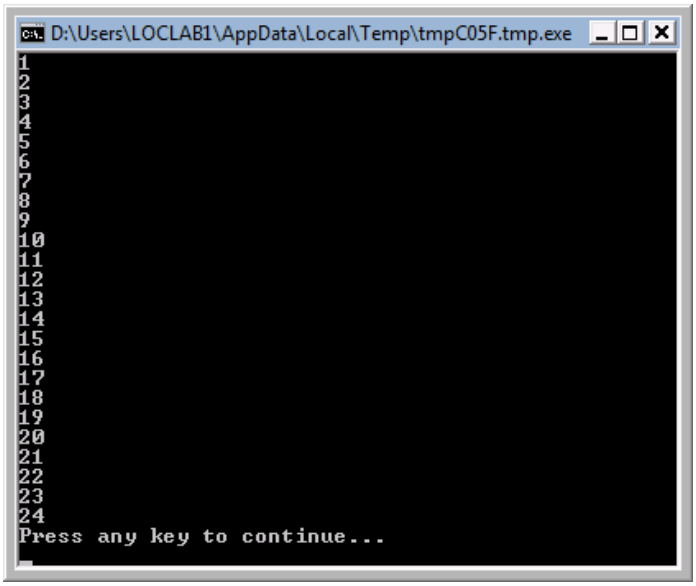

Figura 19 - Usando el bucle For

Note que hemos reducido un programa de 8 líneas a un programa de 4, ¡y hace exactamente lo mismo que el programa de 8 líneas! ¿Recuerda que antes dijimos que habitualmente hay varias formas de hacer la misma cosa? Este es un gran ejemplo.

For..EndFor se llama, en términos de programación, un bucle. Le permite tomar una variable, darle un valor inicial y un valor final, y dejar que el equipo incremente el valor de la variable por usted. Cada vez que el equipo incrementa la variable, ejecuta las instrucciones entre **For** y **EndFor**.

Pero si desea que la variable se incremente de 2 en 2, en lugar de 1 en 1 (supongamos que desea imprimir todos los números impares entre 1 y 24), también puede usar el bucle para hacerlo.

```
For i = 1 To 24 Step 2
 TextWindow.WriteLine(i)
EndFor
```

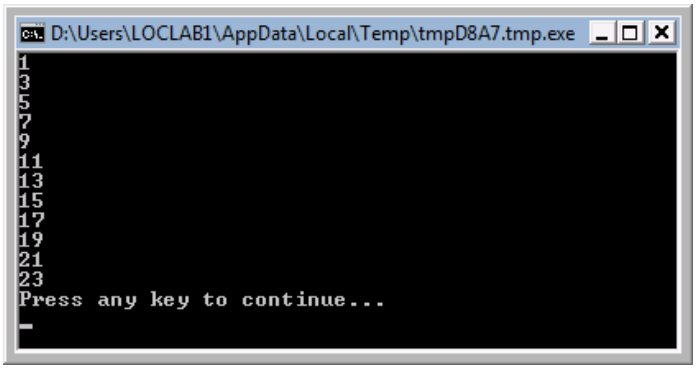

Figura 20 - Sólo los números impares

El **Step 2** que forma la instrucción **For** dice al equipo que incremente el valor de **i** en 2 en lugar de 1 como siempre. Usando **Step** puede especificar cualquier incremento que desee. Hasta puede especificar un valor negativo y hacer que el equipo cuente hacia atrás, como en el siguiente ejemplo:

```
For i = 10 To 1 Step -1
 TextWindow.WriteLine(i)
EndFor
```

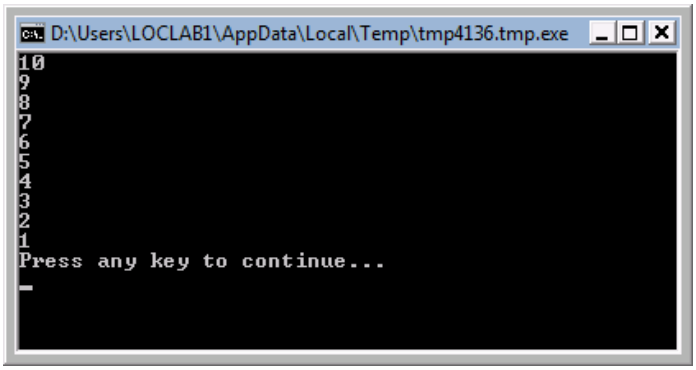

Figura 21 - Contando hacia atrás

El bucle While

El bucle **While** (mientras) es otro método de repetición, que es especialmente útil cuando la cantidad de repeticiones no se conoce por adelantado. Mientras el bucle **For** se ejecuta una cantidad predefinida de veces, el bucle **While** se ejecuta mientras una condición dada es cierta. En el siguiente ejemplo, estamos dividiendo a la mitad un número mientras el resultado sea mayor que 1.

```
número = 100
While (número > 1)
 TextWindow.WriteLine(número)
 número = número / 2
EndWhile
```

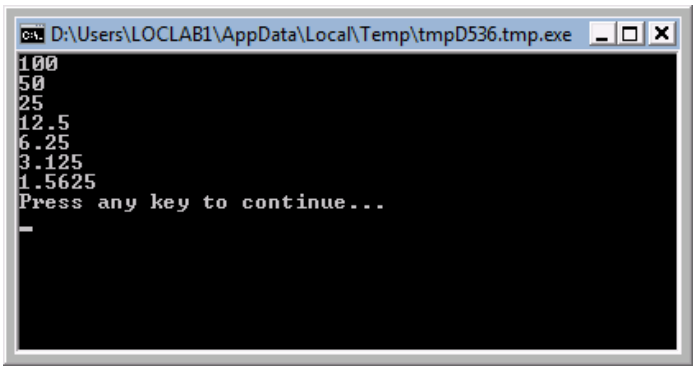

Figura 22 – El bucle divisor

En el programa anterior, asignamos el valor 100 a **número** y ejecutamos el bucle mientras **número** sea mayor que 1. Dentro del bucle, imprimimos el número y luego lo dividimos entre 2. Como cabía esperar, la salida del programa son números que progresivamente son la mitad del anterior.

*De hecho, el equipo internamente escribe nuevamente cada bucle **While** usando instrucciones que usan **If..Then** junto con una o más instrucciones **Goto***

Sería realmente difícil escribir este programa usando un bucle **For**, porque no sabemos cuántas veces el bucle se ejecutará. Con un bucle **While** es fácil evaluar una condición y preguntar al equipo si hay que continuar con el bucle o terminarlo.

```
número = 100
etiquetaInicio:
TextWindow.WriteLine(número)
número = número / 2

If (número > 1) Then
 Goto etiquetaInicio
EndIf
```

Capítulo 6

Comenzando con los gráficos

Hasta ahora, en nuestros ejemplos hemos usado *TextWindow* para explicar los fundamentos del lenguaje Small Basic. No obstante, Small Basic viene con un poderoso conjunto de capacidades gráficas que comenzaremos a explorar en este capítulo.

Presentando GraphicsWindow

Así como tenemos *TextWindow*, que nos permite trabajar con texto y números, Small Basic también tiene *GraphicsWindow* (ventana de gráficos), que podemos usar para dibujar cosas. Vamos a comenzar mostrando la *GraphicsWindow*.

```
GraphicsWindow.Show()
```

Cuando ejecute este programa, notará que en lugar de la habitual ventana negra de texto, obtiene una ventana blanca como la que se muestra a continuación. No hay mucho que hacer con esta ventana por ahora, pero será la ventana base sobre la cual trabajaremos en este capítulo. Puede cerrar esta ventana haciendo clic en el botón 'X' en la esquina superior derecha.

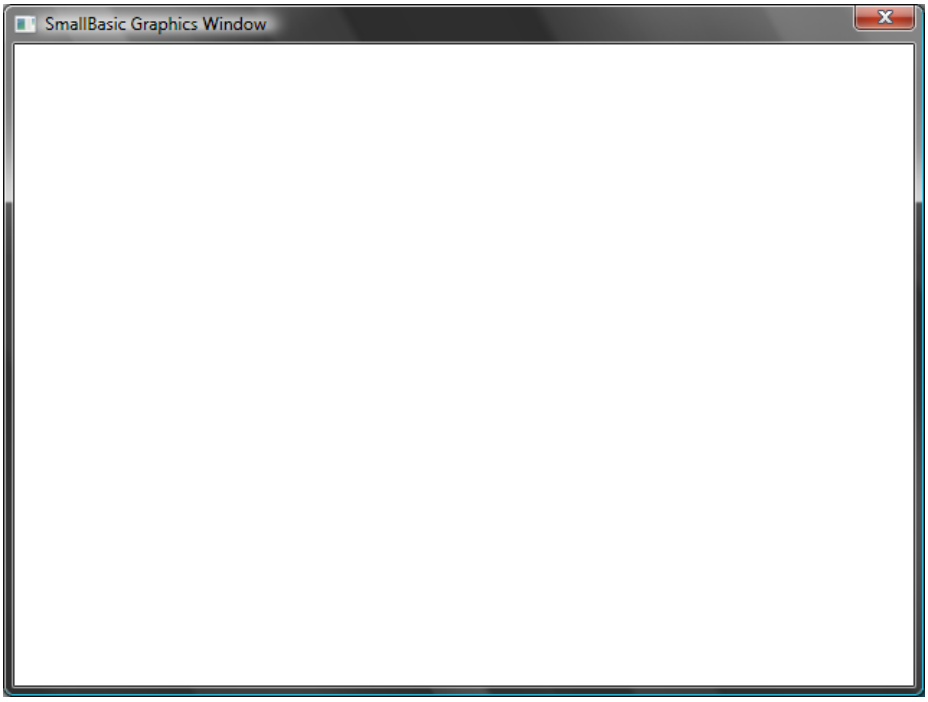

Figura 23 - Una ventana de gráficos vacía

Configurando la ventana de gráficos

La ventana de gráficos le permite cambiar su apariencia como desee. Puede cambiar el título, el fondo y el tamaño. Vamos a seguir adelante y modificarla un poco, sólo para familiarizarnos con la ventana.

```
GraphicsWindow.BackgroundColor = "SteelBlue"
GraphicsWindow.Title = "Mi ventana de gráficos"
GraphicsWindow.Width = 320
GraphicsWindow.Height = 200
GraphicsWindow.Show()
```

Así es como se ve la ventana configurada. Puede cambiar el color de fondo por uno de los diferentes valores listados en el Apéndice B. Juegue con estas propiedades para ver cómo puede modificar la apariencia de la ventana.

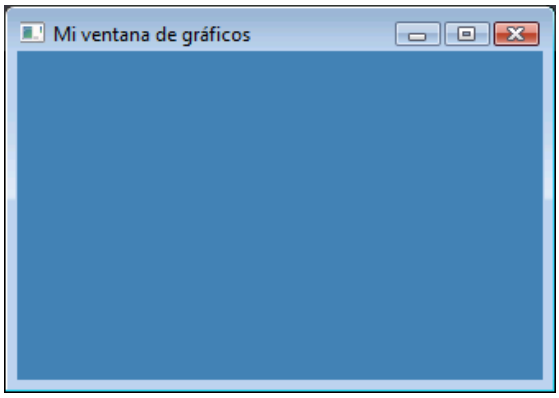

Figura 24 - Una ventana de gráficos a medida

Dibujando líneas

Una vez que tenemos abierta la *GraphicsWindow*, podemos dibujar figuras, texto y hasta imágenes en ella. Vamos a comenzar dibujando algunas figuras simples. He aquí un programa que dibuja un par de líneas en la ventana de gráficos.

```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

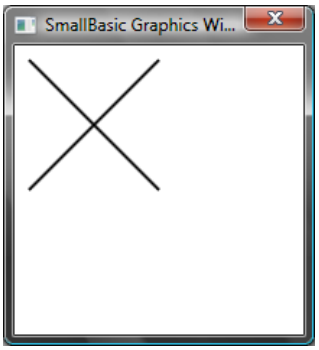

Figura 25 – Líneas cruzadas

Las primeras dos líneas del programa configuran la ventana y las siguientes dos líneas dibujan las líneas cruzadas. Los primeros dos números que siguen a *DrawLine* (dibujar una línea) especifican las coordenadas X e Y iniciales y los otros especifican las coordenadas X e Y finales. Lo interesante de los gráficos de ordenador es que las coordenadas (0, 0) comienzan en la esquina superior izquierda de la ventana. En efecto, el espacio de coordenadas de la ventana se considera que está en el segundo cuadrante.

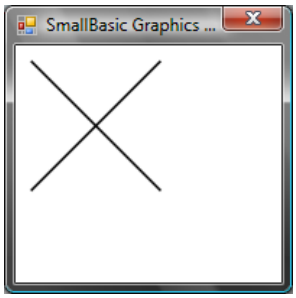

Figura 26 - El mapa de coordenadas

Volviendo al programa de las líneas, es interesante mencionar que Small Basic le permite modificar las propiedades de la línea, tales como su color y su grosor. Primero vamos a modificar el color de las líneas como se muestra en el siguiente programa:

En vez de usar nombres para los colores, puede usar la notación de colores de la web (#RRGGBB). Por ejemplo, #FF0000 es el rojo, #FFFF00 el amarillo, y así sucesivamente. Aprenderemos más acerca de los colores en el capítulo Colores.

```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.PenColor = "Green"
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.PenColor = "Gold"
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

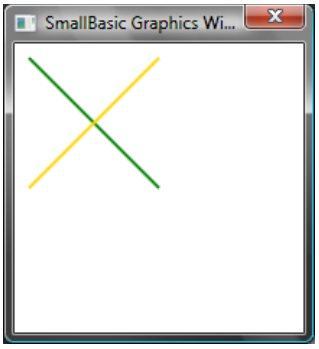

Figura 27 - Cambiando el color de la línea

Ahora, vamos a modificar también el tamaño. En el siguiente programa, cambiamos el ancho de la línea a 10, en lugar del valor predeterminado, que es 1.

```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.PenWidth = 10
GraphicsWindow.PenColor = "Green"
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.PenColor = "Gold"
GraphicsWindow.DrawLine(10, 100, 100, 10)
```

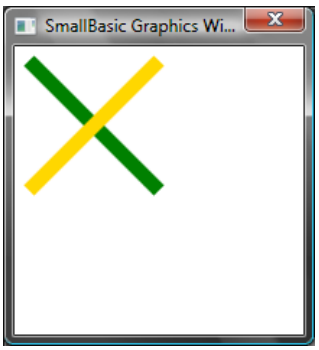

Figura 28 - Líneas gruesas y coloridas

PenWidth (ancho del lápiz) y *PenColor* (color del lápiz) modifican el lápiz que dibuja esas líneas. No afectan solamente a las líneas, sino a cualquier figura que se dibuje una vez que se modifican las propiedades.

Usando las instrucciones de bucle que aprendimos en los capítulos anteriores, podemos escribir fácilmente un programa que dibuje múltiples líneas aumentando el grosor del lápiz.

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.Width = 200
GraphicsWindow.Height = 160
GraphicsWindow.PenColor = "Blue"

For i = 1 To 10
 GraphicsWindow.PenWidth = i
 GraphicsWindow.DrawLine(20, i * 15, 180, i * 15)
EndFor
```

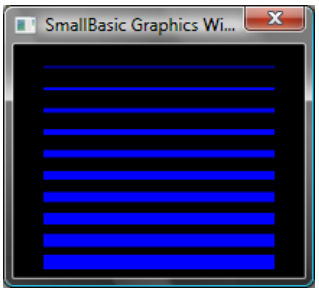

Figura 29 - Lápices de múltiples grosores

La parte interesante de este programa es el bucle, donde podemos aumentar *PenWidth* cada vez que se ejecuta el bucle y luego dibujar una nueva línea debajo de la anterior.

Dibujando y rellenando formas

Cuando tenemos que dibujar formas, normalmente hay dos tipos de operaciones para cada una. Son las operaciones *Draw* de dibujo y las operaciones *Fill* de relleno. Las operaciones *Draw* dibujan el contorno de la forma usando un lápiz, y las operaciones *Fill* rellenan la forma usando un lápiz. Por ejemplo, en el siguiente programa hay dos rectángulos, uno que se dibuja usando un lápiz rojo y otro que se rellena usando un lápiz verde.

```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawRectangle(20, 20, 300, 60)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillRectangle(60, 100, 300, 60)
```

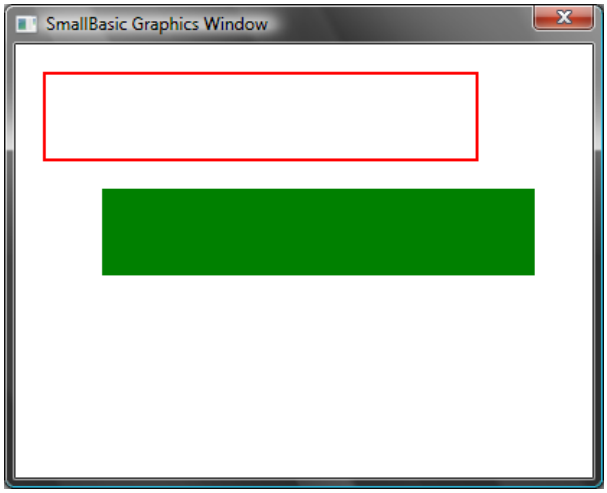

Figura 30 - Dibujando y rellenando

Para dibujar o rellenar un rectángulo necesita cuatro números. Los primeros dos números representan las coordenadas X e Y del vértice superior izquierdo del rectángulo. El tercer número especifica el ancho del rectángulo, mientras que el cuarto especifica su altura. De hecho, lo mismo se aplica para dibujar y rellenar elipses, como se muestra en el siguiente programa:

```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawEllipse(20, 20, 300, 60)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillEllipse(60, 100, 300, 60)
```

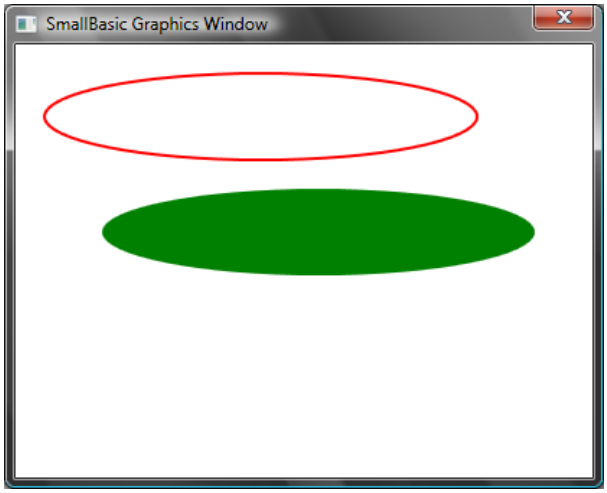

Figura 31 - Dibujando y rellenando elipses

Los círculos son un tipo particular de elipse. Si desea dibujar círculos, tendrá que especificar el mismo ancho y alto.

```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

GraphicsWindow.PenColor = "Red"
GraphicsWindow.DrawEllipse(20, 20, 100, 100)

GraphicsWindow.BrushColor = "Green"
GraphicsWindow.FillEllipse(100, 100, 100, 100)
```

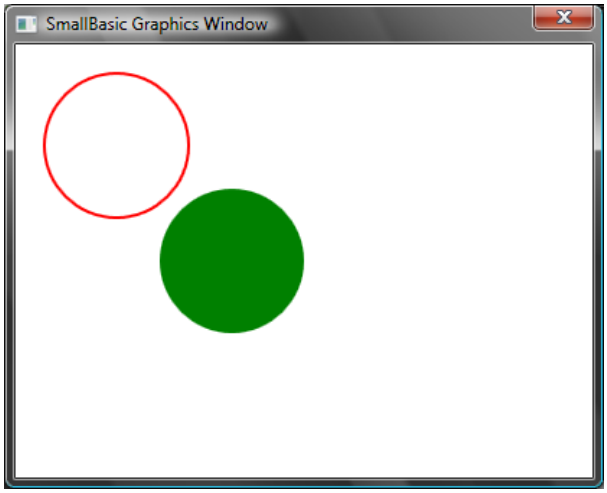

Figura 32 – Círculos

Capítulo 7

Diversión con las formas

Vamos a divertirnos un poco en este capítulo con lo que hemos aprendido hasta ahora. Este capítulo contiene ejemplos que muestran maneras interesantes de combinar todo lo que ha aprendido hasta ahora para crear algunos programas atractivos.

Rectangular

Aquí dibujamos múltiples rectángulos en un bucle a la vez que aumentamos su tamaño.

```
GraphicsWindow.BackgroundColor = "Black" 'Negro
GraphicsWindow.PenColor = "LightBlue" 'Celeste
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200

For i = 1 To 100 Step 5
 GraphicsWindow.DrawRectangle(100 - i, 100 - i, i * 2, i * 2)
EndFor
```

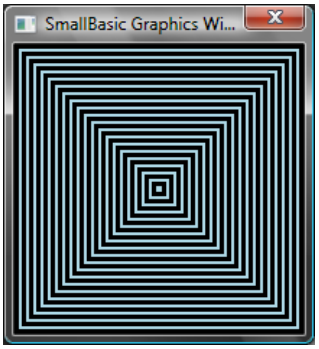

Figura 33 - Rectangular

Circular

Una variante del programa anterior, dibuja círculos en lugar de cuadrados.

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightGreen" 'Verde claro
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200

For i = 1 To 100 Step 5
 GraphicsWindow.DrawEllipse(100 - i, 100 - i, i * 2, i * 2)
EndFor
```

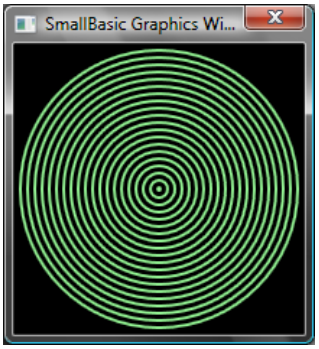

Figura 34 – Circular

Aleatorio

Este programa usa la operación *GraphicsWindow.GetRandomColor* (obtener color al azar) para asignar colores aleatorios al lápiz y luego usa *Math.GetRandomNumber* (obtener número al azar) para asignar las coordenadas X e Y de los círculos. Estas dos operaciones se pueden combinar de maneras interesantes para crear programas atractivos que dan diferentes resultados cada vez que se ejecutan.

```
GraphicsWindow.BackgroundColor = "Black"
For i = 1 To 1000
 GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
 x = Math.GetRandomNumber(640)
 y = Math.GetRandomNumber(480)
 GraphicsWindow.FillEllipse(x, y, 10, 10)
EndFor
```

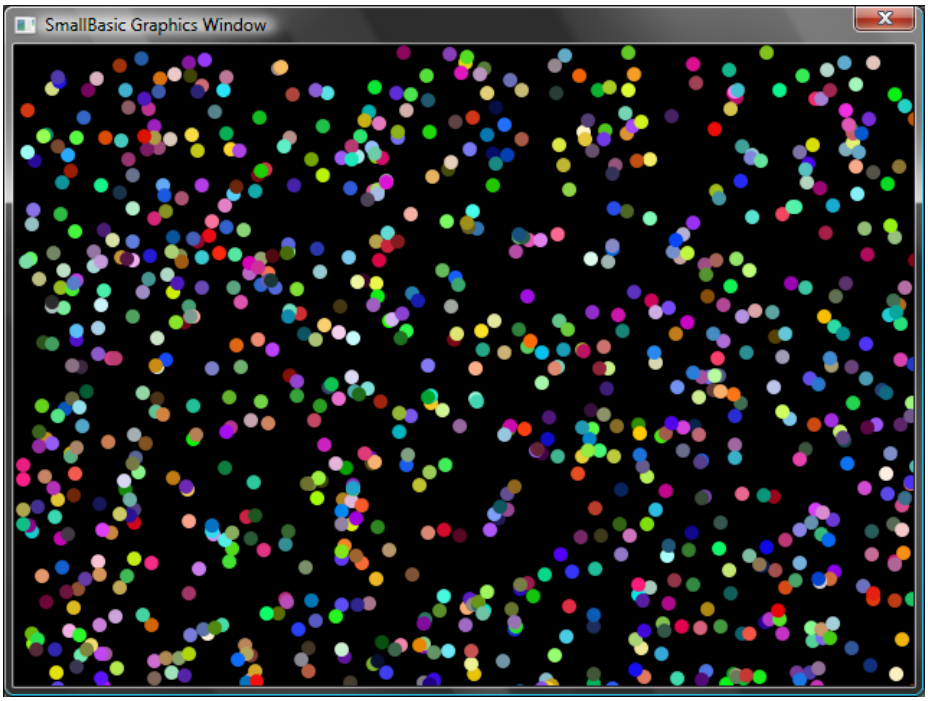

Figura 35 – Aleatorio

Fractales

El siguiente programa dibuja un simple triángulo fractal usando números aleatorios. Un fractal es una forma geométrica que se puede subdividir en partes, cada una de las cuales se asemeja a la forma original. En este caso, el programa dibuja cientos de triángulos cada uno de los cuales se asemeja a su triángulo original. Y como el programa se ejecuta durante unos pocos segundos, puede ver en realidad cómo se forman los triángulos a partir de simples puntitos. La lógica en sí misma es en cierta forma difícil de describir y dejaremos como ejercicio que pueda explorarla.

```
GraphicsWindow.BackgroundColor = "Black"
x = 100
y = 100

For i = 1 To 100000
 r = Math.GetRandomNumber(3)
 ux = 150
 uy = 30
 If (r = 1) then
 ux = 30
 uy = 1000
 EndIf

 If (r = 2) Then
 ux = 1000
 uy = 1000
 EndIf

 x = (x + ux) / 2
 y = (y + uy) / 2

 GraphicsWindow.SetPixel(x, y, "LightGreen")
EndFor
```

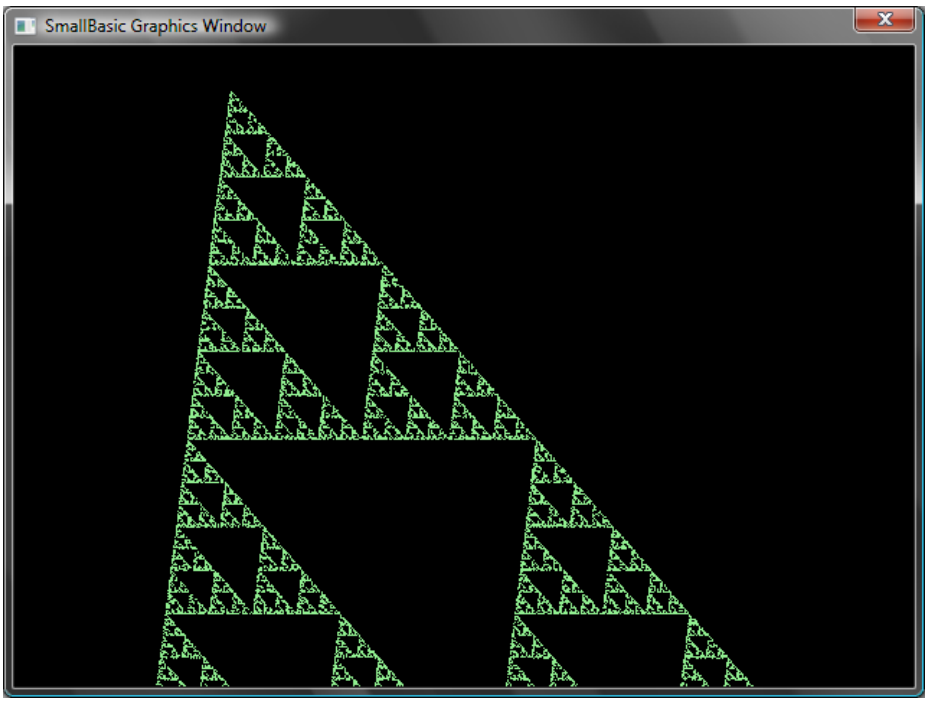

Figura 36 - Triángulos fractales

Si realmente desea ver los puntos formando lentamente el fractal, puede introducir un retraso en el bucle mediante el uso de la operación *Program.Delay*. Esta operación recibe un número que especifica cuánto tiempo esperar, en milisegundos. Aquí está el programa modificado, con la línea cambiada en negrita.

```
GraphicsWindow.BackgroundColor = "Black"
x = 100
y = 100

For i = 1 To 100000
 r = Math.GetRandomNumber(3)
 ux = 150
 uy = 30
 If (r = 1) Then
 ux = 30
 uy = 1000
 EndIf

 If (r = 2) Then
 ux = 1000
 uy = 1000
 EndIf

 x = (x + ux) / 2
 y = (y + uy) / 2

 GraphicsWindow.SetPixel(x, y, "LightGreen") 'Poner píxel
 Program.Delay(2) 'Demora
EndFor
```

Aumentar el retraso hace que el programa sea más lento. Experimente con los números para ver cuál le gusta más.

Otra modificación que puede hacer a este programa es reemplazar la siguiente línea:

```
GraphicsWindow.SetPixel(x, y, "LightGreen")
```

con:

```
color = GraphicsWindow.GetRandomColor()
GraphicsWindow.SetPixel(x, y, color)
```

Este cambio hará que el programa dibuje los píxeles del triángulo usando colores aleatorios.

Capítulo 8

Tortuga gráfica

Logo

En los años 70, había un lenguaje de programación muy simple pero potente, llamado Logo, que era usado por algunos investigadores. Esto fue hasta que alguien agregó al lenguaje lo que se llamó la "tortuga gráfica" e hizo disponible una "tortuga" que era visible en la pantalla y respondía a comandos como "muévete", "ve hacia adelante" o "gira a la izquierda". Usando la tortuga, las personas eran capaces de dibujar formas interesantes en la pantalla. Esto hizo que el lenguaje fuera inmediatamente accesible y atractivo a personas de todas las edades y fue el principal responsable de su gran popularidad en los años 80.

Small Basic viene con un objeto *Turtle* (tortuga) con varios comandos que pueden ser llamados desde programas de Small Basic. En este capítulo, usaremos *Turtle* para dibujar gráficos en la pantalla.

La tortuga

Para comenzar, necesitamos mostrar la tortuga en la pantalla. Esto se puede lograr con un programa de una sola línea.

```
Turtle.Show() 'Mostrar
```

Cuando ejecute este programa verá una ventana blanca, tal como la que vimos en el capítulo anterior, excepto que esta tiene una tortuga en su centro. Esta es la tortuga que seguirá nuestras instrucciones y dibujará lo que le pidamos.

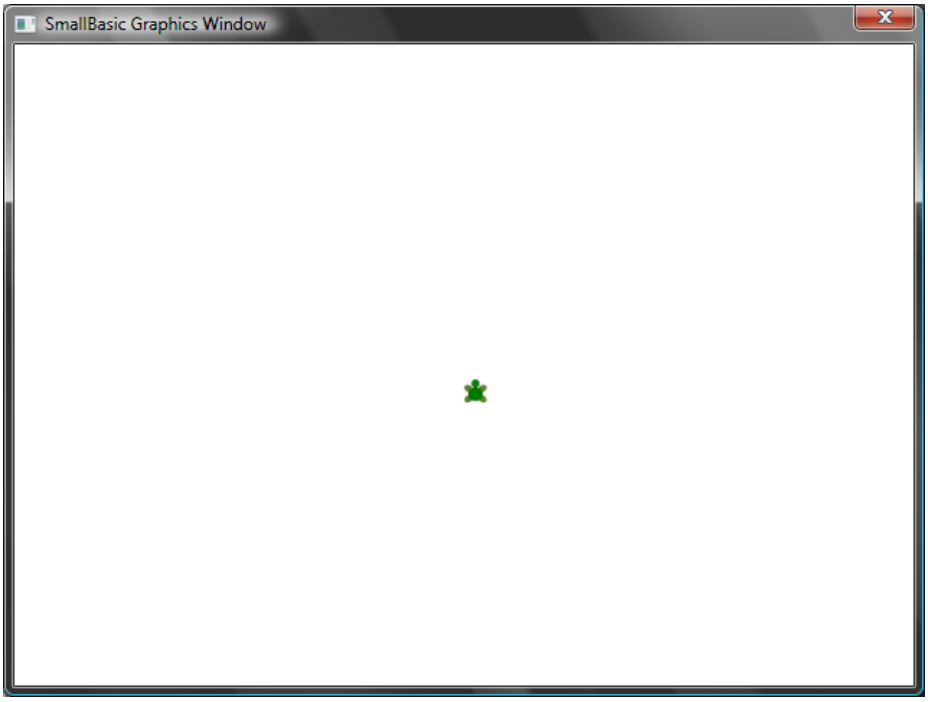

Figura 37 - La tortuga es visible

Moviendo y dibujando

Una de las instrucciones que *Turtle* entiende es *Move* (mover). Esta operación toma un número como entrada. Ese número le dice a *Turtle* cuánto tiene que moverse. En el siguiente ejemplo, le pedimos a *Turtle* que se mueva 100 píxeles.

```
Turtle.Move(100)
```

Cuando ejecute este programa, puede ver la tortuga realmente moverse lentamente 100 píxeles hacia arriba. Mientras se mueve, también verá que dibuja una línea detrás de ella. Cuando la tortuga ha terminado de moverse, el resultado tendrá el mismo aspecto que la siguiente figura.

Cuando use operaciones con Turtle*, no es necesario llamar a* Show*. La tortuga quedará visible automáticamente cuando se ejecuta cualquier operación con* Turtle*.*

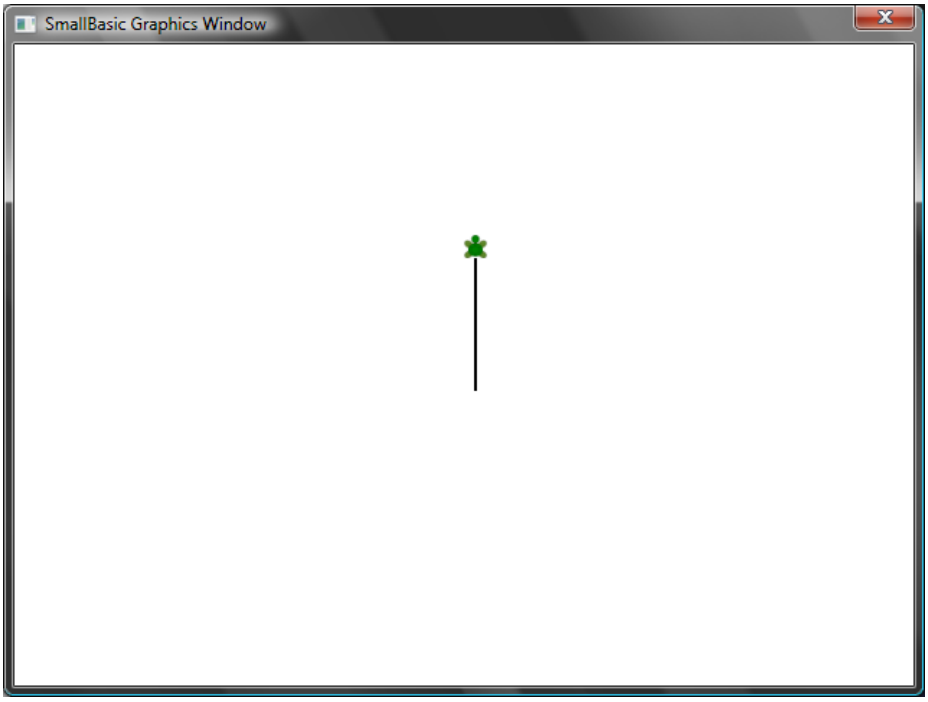

Figura 38 - Moviéndose un centenar de píxeles

Dibujando un cuadrado

Un cuadrado tiene cuatro lados, dos verticales y dos horizontales. Para dibujar un cuadrado necesitamos conseguir que la tortuga dibuje una línea, gire a la derecha y dibuje otra línea, y continúe así hasta que los cuatro lados del cuadrado estén terminados. Si traducimos esto en un programa, así es como queda:

```
Turtle.Move(100)
Turtle.TurnRight() 'Mover a la derecha
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
Turtle.Move(100)
Turtle.TurnRight()
```

Cuando ejecute este programa, puede ver la tortuga dibujando un cuadrado, una línea cada vez, y el resultado es igual al de la siguiente figura.

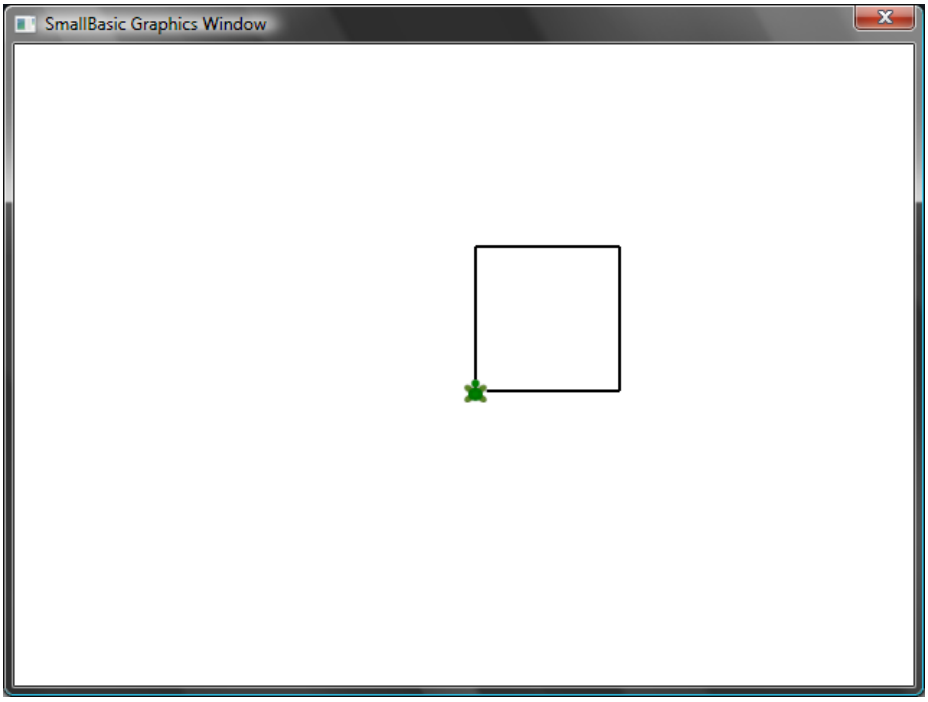

Figura 39 - La tortuga dibujando un cuadrado

Es interesante mencionar que estamos realizando las mismas dos operaciones una y otra vez, cuatro veces para ser más precisos. Y ya hemos aprendido que tales comandos repetitivos pueden ser ejecutados usando bucles. Por lo tanto, si tomamos el programa anterior y lo modificamos para usar el bucle **For..EndFor**, lograremos un programa mucho más simple.

```
For i = 1 To 4
 Turtle.Move(100)
 Turtle.TurnRight()
EndFor
```

Cambiando los colores

La tortuga dibuja en la misma *GraphicsWindow* que vimos en el capítulo anterior. Esto significa que todas las operaciones que aprendimos en el capítulo anterior las podemos utilizar con la tortuga. Por ejemplo, el siguiente programa dibujará un cuadrado con cada lado de un color diferente.

```
For i = 1 To 4
 GraphicsWindow.PenColor = GraphicsWindow.GetRandomColor()
 'PenColor: color del lápiz; GetRandomColor: obtener color al azar
 Turtle.Move(100)
 Turtle.TurnRight()
EndFor
```

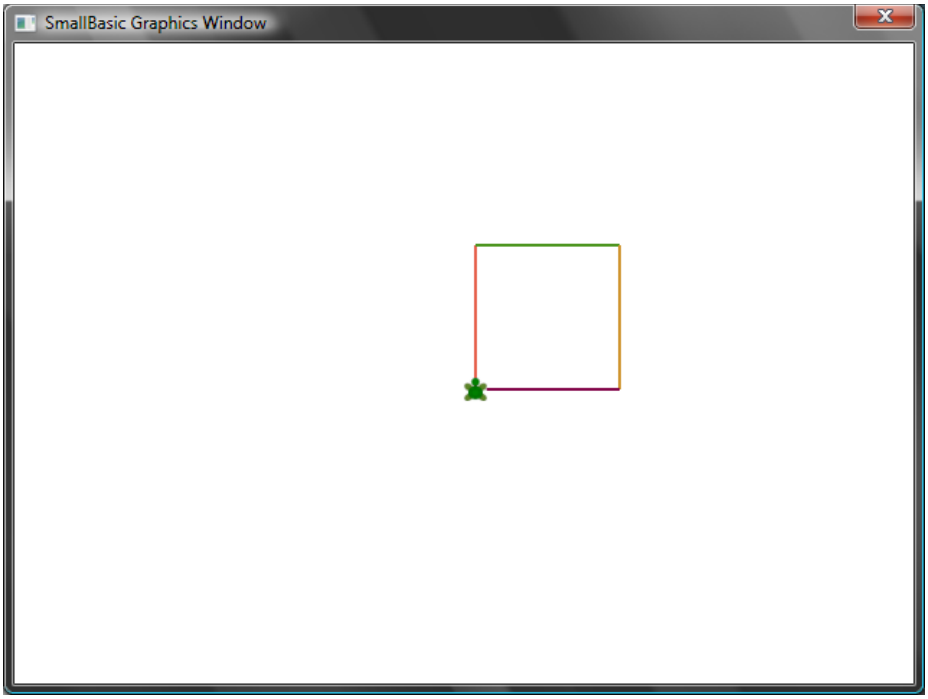

Figura 40 - Cambiando los colores

Dibujando formas más complejas

Turtle, además de las operaciones *TurnRight* (girar a la derecha) y *TurnLeft* (girar a la izquierda), tiene una operación *Turn* (girar). Esta operación toma una entrada que especifica el ángulo de giro. Usando esta operación, es posible dibujar un polígono de cualquier cantidad de lados. El siguiente programa dibuja un hexágono (un polígono de seis lados).

```
For i = 1 To 6
 Turtle.Move(100)
 Turtle.Turn(60)
EndFor
```

Pruebe este programa para ver si realmente dibuja un hexágono. Observe que como el ángulo entre los lados es de 60 grados, usamos **Turn(60)**. Para ese polígono, cuyos lados son todos iguales, el ángulo entre los lados puede ser obtenido fácilmente dividiendo 360 entre el número de lados. Sabiendo de

esta información y usando variables, podemos escribir un programa bastante genérico que dibuje un polígono de cualquier cantidad de lados.

```
lados = 12

largo = 400 / lados
ángulo = 360 / lados

For i = 1 To lados
 Turtle.Move(largo)
 Turtle.Turn(ángulo)
EndFor
```

Usando este programa, puede dibujar cualquier polígono con sólo modificar la variable *lados*. Poniendo 4 dibujará un cuadrado. Poniendo un valor suficientemente grande, digamos 50, el resultado será similar a un círculo.

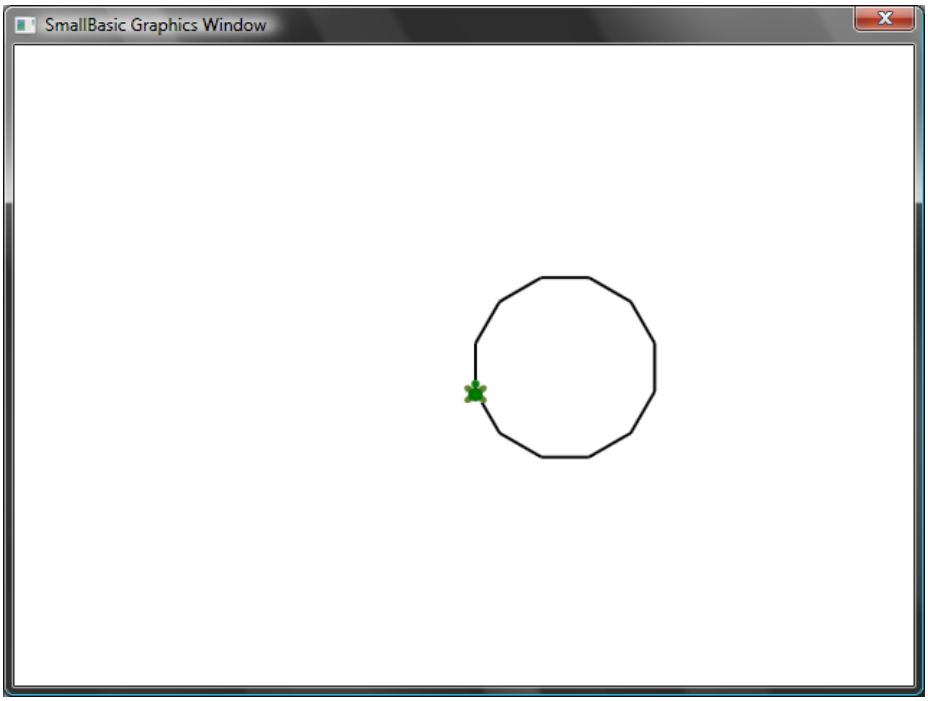

Figura 41 - Dibujando un polígono de 12 lados

Usando la técnica que acaba de aprender, puede hacer que la tortuga dibuje múltiples círculos, cada vez con un pequeño giro, resultando en una salida interesante.

```
lados = 50
largo = 400 / lados
ángulo = 360 / lados

Turtle.Speed = 9

For j = 1 To 20
 For i = 1 To lados
 Turtle.Move(largo)
 Turtle.Turn(ángulo)
 EndFor
 Turtle.Turn(18)
EndFor
```

El programa anterior tiene dos bucles **For..EndFor**, uno dentro del otro. El bucle interior (*i = 1 to lados*) es similar al del programa del polígono y es el responsable de dibujar el círculo. El bucle exterior (*j = 1 to 20*) es el responsable de girar la tortuga un poco cada vez que se dibuja el círculo. Esto dice a la tortuga que dibuje 20 círculos. Cuando se ponen juntos, este programa resulta en un patrón muy interesante, como el que se muestra a continuación.

En el programa anterior, hicimos que la tortuga se moviera más rápido, cambiado Speed *(velocidad) a 9. Puede asignar esta propiedad con cualquier valor entre 1 y 10 para hacer que la tortuga se mueva tan rápido como desee.*

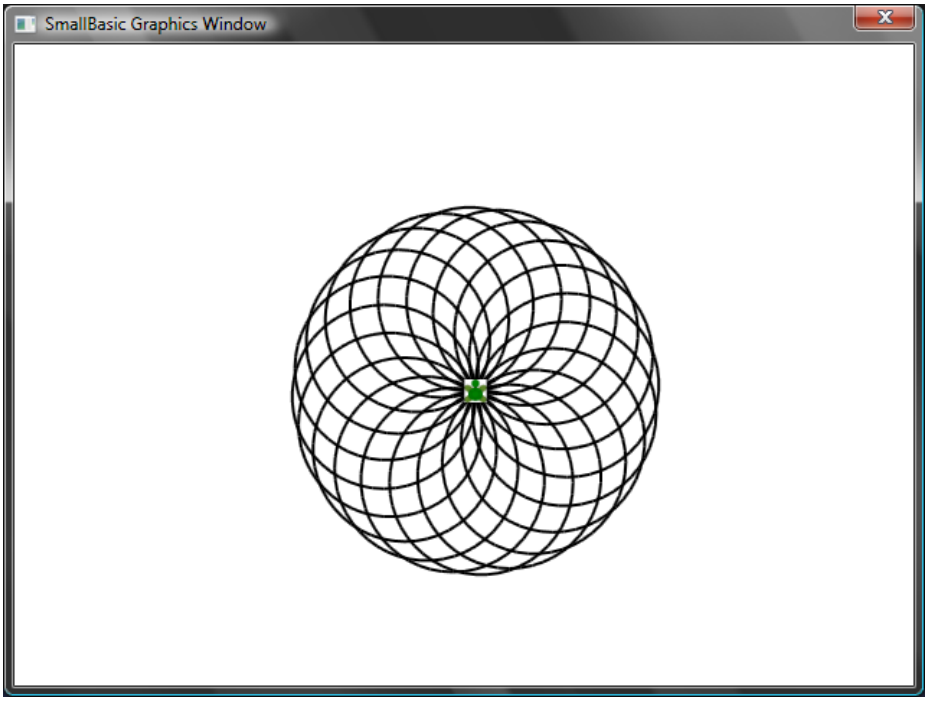

Figura 42 - Yendo en círculos

Moviéndose por ahí

Puede hacer que la tortuga no dibuje usado la operación *PenUp* (subir el lápiz). Esto permite mover la tortuga a cualquier lado de la pantalla sin que dibuje una línea. Usando *PenDown* (bajar el lápiz) hará que la tortuga dibuje nuevamente. Esto se puede utilizar para obtener algunos efectos interesantes, como por ejemplo, líneas punteadas. He aquí un programa que usa este concepto para dibujar un polígono de líneas punteadas.

```
lados = 6

largo = 400 / lados
ángulo = 360 / lados

For i = 1 To lados
 For j = 1 To 6
 Turtle.Move(largo / 12)
 Turtle.PenUp()
 Turtle.Move(largo / 12)
 Turtle.PenDown()
 EndFor
 Turtle.Turn(ángulo)
EndFor
```

Nuevamente, este programa tiene dos bucles. El bucle interior dibuja una sola línea punteada, mientras que el bucle exterior indica cuántas líneas a dibujar. En nuestro ejemplo, hemos usado 6 para la variable *lados* y por lo tanto tenemos un hexágono de líneas punteadas, como el que aparece a continuación.

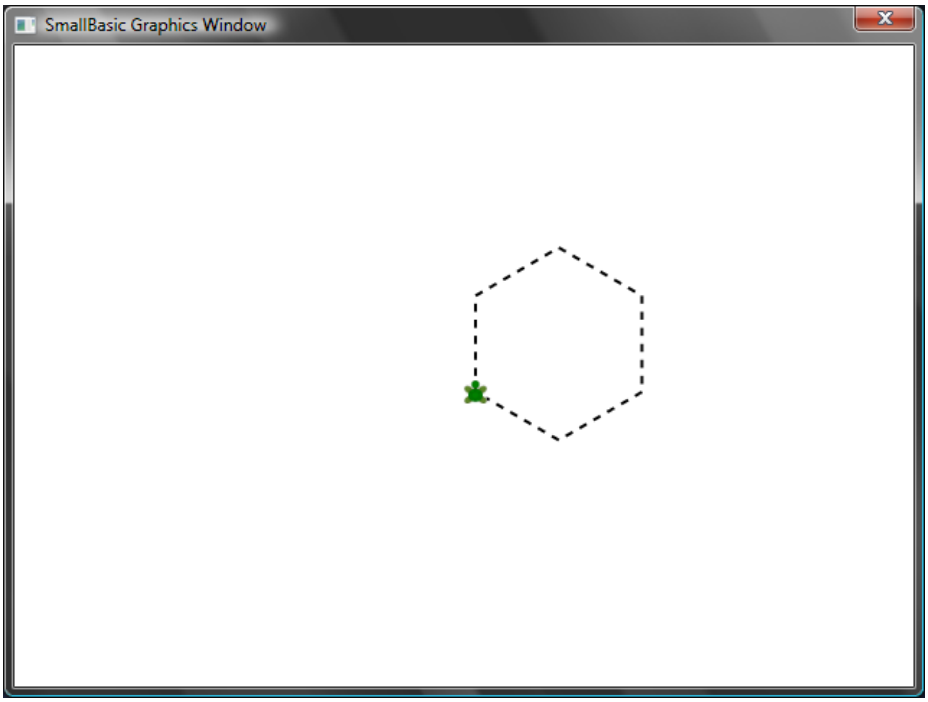

Figura 43 – Usando PenUp y PenDown

Capítulo 9

Subrutinas

Muy frecuentemente, mientras escribimos programas encontramos casos donde tenemos que ejecutar el mismo conjunto de pasos, una y otra vez. En estos casos, probablemente no tenga sentido volver a escribir las mismas instrucciones varias veces. Es entonces cuando resultan útiles las subrutinas.

Una subrutina es una porción de código dentro de un programa más grande que habitualmente hace algo específico, y que puede ser invocada desde cualquier parte del programa. Las subrutinas están identificadas por un nombre que sigue a la palabra clave **Sub** y terminan con la palabra clave **EndSub**. Por ejemplo, el siguiente fragmento representa una subrutina cuyo nombre es *ImprimirHora*, y hace el trabajo de imprimir la hora actual en la *TextWindow*.

```
Sub ImprimirHora
 TextWindow.WriteLine(Clock.Time)
EndSub
```

A continuación hay un programa que incluye la subrutina y la invoca desde varios lugares.

```
ImprimirHora()
TextWindow.Write("Introduzca su nombre: ")
nombre = TextWindow.Read()
TextWindow.Write(nombre + ", la hora es: ")
ImprimirHora()

Sub ImprimirHora
 TextWindow.WriteLine(Clock.Time)
EndSub
```

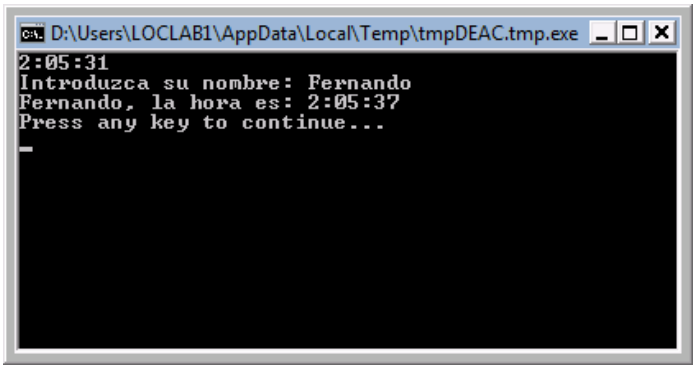

Figura 44 – Invocando una subrutina simple

Usted ejecuta una subrutina invocando *NombreSubrutina()*. Como siempre, los paréntesis "()"son necesarios para indicar al equipo que quiere ejecutar una subrutina.

Recuerde, puede invocar una subrutina de Small Basic solo desde el mismo programa. No puede invocar una subrutina desde otro programa.

Ventajas de usar subrutinas

Como acabamos de ver, las subrutinas ayudan a reducir la cantidad de código que hay que escribir. Una vez que tiene la subrutina *ImprimirHora* escrita, puede llamarla desde cualquier lado del programa, y ella imprimirá la hora actual.

Además, las subrutinas pueden ayudar a descomponer problemas complejos en tareas más simples. Digamos que tiene una ecuación compleja para resolver, puede escribir varias subrutinas que resuelvan partes más pequeñas de la ecuación compleja. Luego puede unir los resultados para obtener la solución a la ecuación original.

Las subrutinas también pueden ayudar a mejorar la legibilidad de un programa. En otras palabras, si utiliza buenos nombres para las partes del programa que se ejecutan frecuentemente, su programa resulta fácil de leer y comprender. Esto es muy importante si pretende entender el programa de alguien más, o si quiere que su programa sea comprendido por otros. A veces, es útil también cuando quiere leer su propio programa, digamos una semana después de escribirlo.

Usando variables

Puede acceder y usar cualquier variable que tenga en un programa desde una subrutina. Por ejemplo, el siguiente programa acepta dos números e imprime el mayor de ellos. Note que la variable *máx* es usada tanto dentro de la subrutina como fuera de ella.

```
TextWindow.Write("Introduzca el primer número: ")
núm1 = TextWindow.ReadNumber()
TextWindow.Write("Introduzca el segundo número: ")
núm2 = TextWindow.ReadNumber()

FindMax()
TextWindow.WriteLine("El mayor de los dos número es: " + máx)

Sub FindMax
 If (núm1 > núm2) Then
 máx = núm1
 Else
 máx = núm2
 EndIf
EndSub
```

Y la salida de este programa es como sigue:

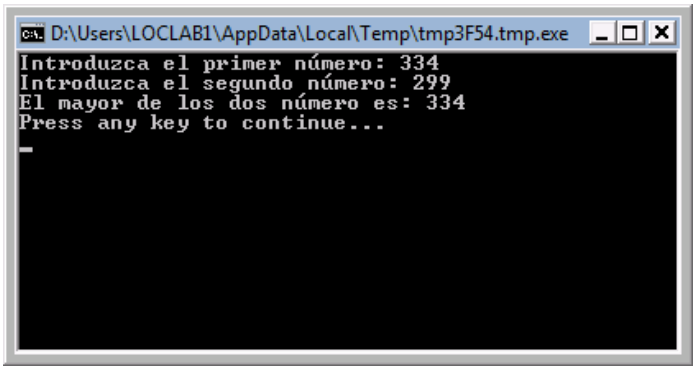

Figura 45 - El mayor de dos números usando subrutinas

Tomemos otro ejemplo que ilustrará el uso de subrutinas. Este vez usaremos un programa gráfico que calcula varios puntos, que almacenará en variables *x* e *y*. Luego llama a la subrutina *DibujarCirculoUsandoCentro*, que es responsable de dibujar un círculo usando *x* e *y* como centro.

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightBlue"
GraphicsWindow.Width = 480
For i = 0 To 6.4 Step 0.17
 x = Math.Sin(i) * 100 + 200 'Seno'
 y = Math.Cos(i) * 100 + 200 'Coseno'

 DibujarCírculoUsandoCentro()
EndFor

Sub DibujarCírculoUsandoCentro
 comienzoX = x - 40
 comienzoY = y - 40

 GraphicsWindow.DrawEllipse(comienzoX, comienzoY, 120, 120)
EndSub
```

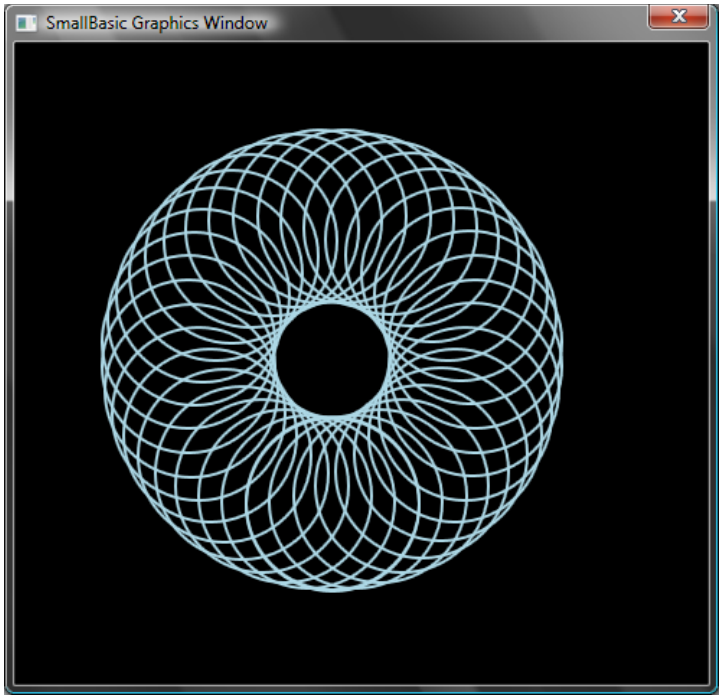

Figura 46 – Ejemplo gráfico con subrutinas

Invocando subrutinas dentro de bucles

A veces las subrutinas son invocadas desde dentro de un bucle, durante el cual ejecutan el mismo conjunto de instrucciones, pero con diferentes valores en una o más de las variables. Por ejemplo, digamos que tiene una subrutina llamada *ComprobarPrimo* y esta subrutina determina si un número es

primo o no. Puede escribir un programa que permita al usuario introducir un valor y le indique si es primo o no, usando esta subrutina. El siguiente programa muestra cómo hacerlo.

```
TextWindow.Write("Introduzca un número: ")
i = TextWindow.ReadNumber()
esPrimo = "Cierto"
ComprobarPrimo()
If (esPrimo = "Cierto") Then
 TextWindow.WriteLine(i + " es un número primo")
Else
 TextWindow.WriteLine(i + " no es un número primo")
EndIf

Sub ComprobarPrimo
 For j = 2 To Math.SquareRoot(i)
 If (Math.Remainder(i, j) = 0) Then
  esPrimo = "Falso"
  Goto FinBucle
 EndIf
 EndFor
FinBucle:
EndSub
```

La subrutina *ComprobarPrimo* toma el valor de *i* y trata de dividirlo por números menores. Si un número divide a *i* con resto cero, entonces i no es un número primo. En ese momento la subrutina asigna a *esPrimo* el valor *"Falso"* y termina. Si el número no es divisible por números más pequeños, entonces el valor de *esPrimo* permanece como *"Cierto"*.

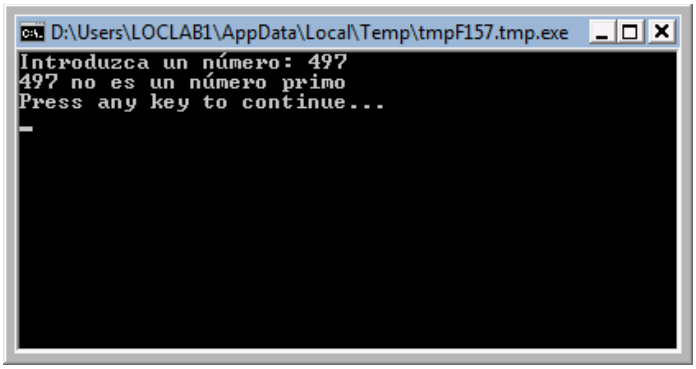

Figura 47 – Comprobando números primos

Ahora que tiene una subrutina que puede hacer la prueba de números primos por nosotros, puede querer usarla para listar todos los números primos menores que, digamos, 100. Es realmente fácil modificar el programa anterior y hacer la invocación de *ComprobarPrimo* dentro del bucle. Esto da a la

subrutina diferentes valores para calcular, cada vez que el bucle se ejecuta. Veamos cómo hacer esto en el siguiente ejemplo.

```
For i = 3 To 100
 esPrimo = "Cierto"
 ComprobarPrimo()
 If (esPrimo = "Cierto") Then
 TextWindow.WriteLine(i)
 EndIf
EndFor

Sub ComprobarPrimo
 For j = 2 To Math.SquareRoot(i)
 If (Math.Remainder(i, j) = 0) Then
  esPrimo = "Falso"
  Goto FinLoop
 EndIf
 Endfor
FinLoop:
EndSub
```

En el programa anterior, se actualiza el valor de *i* cada vez que el bucle se ejecuta. Dentro del bucle, se hace una invocación a la subrutina *ComprobarPrimo*. La subrutina *ComprobarPrimo* entonces toma el valor de *i* y calcula si *i* es o no un número primo. El resultado es almacenado en la variable *esPrimo*, a la que se puede acceder en el bucle fuera de la subrutina. El valor de *i* se imprime si resulta un número primo. Y como el bucle comienza en 3 y se incrementa hasta 100, tenemos una lista de los primeros números primos entre 3 y 100. Debajo aparece el resultado del programa.

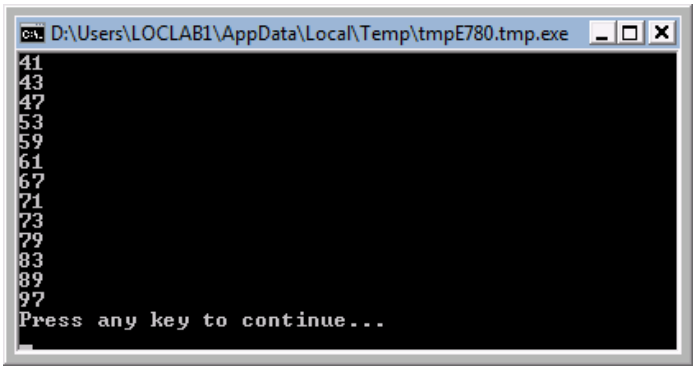

Figura 48 – Números primos

Capítulo 10

Matrices

Ahora ya conoce a fondo cómo utilizar las variables, al fin y al cabo ha llegado hasta aquí y continúa divirtiéndose, ¿verdad?

Vamos, por un momento, a volver a visitar el primer programa que escribimos con variables:

```
TextWindow.Write("Escriba su nombre: ")
nombre = TextWindow.Read()
TextWindow.WriteLine("Hola " + nombre)
```

En este programa, hemos recibido y almacenado el nombre del usuario en una variable llamada **nombre**. A continuación dijimos "Hola" al usuario. Ahora, imaginemos que hay más de un usuario, por ejemplo 5. ¿Cómo almacenaríamos todos sus nombres? Una forma de hacerlo sería:

```
TextWindow.Write("Introduczca su nombre, usuario número 1: ")
nombre1 = TextWindow.Read()
TextWindow.Write("Introduczca su nombre, usuario número 2: ")
nombre2 = TextWindow.Read()
TextWindow.Write("Introduczca su nombre, usuario número 3: ")
nombre3 = TextWindow.Read()
TextWindow.Write("Introduczca su nombre, usuario número 4: ")
nombre4 = TextWindow.Read()
TextWindow.Write("Introduczca su nombre, usuario número 5: ")
nombre5 = TextWindow.Read()

TextWindow.Write("Hola ")
TextWindow.Write(nombre1 + ", ")
TextWindow.Write(nombre2 + ", ")
TextWindow.Write(nombre3 + ", ")
TextWindow.Write(nombre4 + ", ")
TextWindow.WriteLine(nombre5)
```

Cuando ejecute el programa, obtendrá el siguiente resultado:

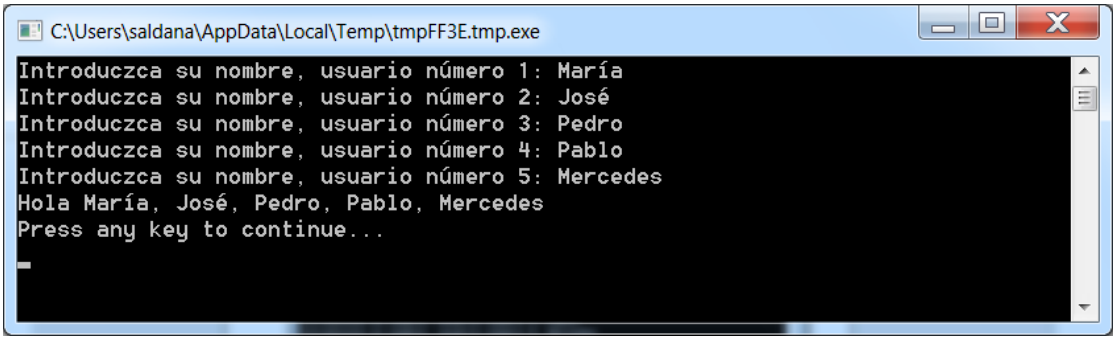

Figura 49 – Sin utilizar matrices

Tiene que haber una manera mejor de escribir un programa de este tipo tan sencillo, ¿verdad? Sobre todo porque el ordenador es realmente bueno ejecutando tareas repetitivas, ¿por qué tenemos que molestarnos con escribir el mismo código una y otra vez para cada nuevo usuario? El truco está en almacenar y recuperar el nombre de más de un usuario utilizando la misma variable. Si podemos hacer eso, entonces podemos utilizar un bucle **For** que ya aprendimos en capítulos anteriores. Aquí es donde las matrices vienen al rescate.

¿Qué es una matriz?

Una matriz es un tipo de variable que puede contener más de un valor en cada momento. En otras palabras, lo que esto significa es que en lugar de tener que crear las variables **nombre1**, **nombre2**, **name3**, **name4** y **name5** para poder almacenar cinco nombres de usuario, podemos utilizar sólo **nombre**

para almacenar el nombre de los cinco usuarios. La forma de almacenar varios valores es utilizando un "índice". Por ejemplo, **nombre[1]**, **nombre[2]**, **nombre[3]**, **nombre[4]** y **nombre[5]** pueden almacenar un valor de cada una. Los números 1, 2, 3, 4 y 5 se denominan *índices* de la matriz.

A pesar de que **nombre[1]**, **nombre[2]**, **nombre[3]**, **nombre[4]** y **nombre[5]** parecen diferentes variables, en realidad son todas la misma variable. Se puede estar preguntando ¿y cuál es la ventaja de todo esto? Lo mejor de almacenar valores en una matriz es que puede especificar el índice, utilizando una variable, lo que permite acceder fácilmente a matrices dentro de bucles.

Ahora, veamos cómo podemos poner nuestros nuevos conocimientos en práctica para reescribir nuestro programa anterior con matrices.

```
For i = 1 To 5
  TextWindow.Write("Introduczca su nombre, usuario número " + i + ": ")
  nombre[i] = TextWindow.Read()
EndFor

TextWindow.Write("Hola ")
For i = 1 To 5
  TextWindow.Write(nombre[i] + ", ")
EndFor
TextWindow.WriteLine("")
```

Much easier to read, isn't it? Notice the two bolded lines. The first one stores a value in the array and the second one reads it from the array. The value you store in **name[1]** will not be affected by what you store in **name[2]**. Hence for most purposes you can treat **name[1]** and **name[2]** as two different variables with the same identity.

Mucho más fácil de leer, ¿verdad? Tenga en cuenta las dos líneas en negrita. La primera almacena un valor de la matriz y la segunda lee de la matriz. El valor que se almacena en **nombre[1]** no se verá afectado por el que se almacena en **nombre[2]**. Por lo tanto para la mayoría de los propósitos se pueden tratar **nombre[1]** y **nombre[2]** como dos variables distintas con la misma identidad.

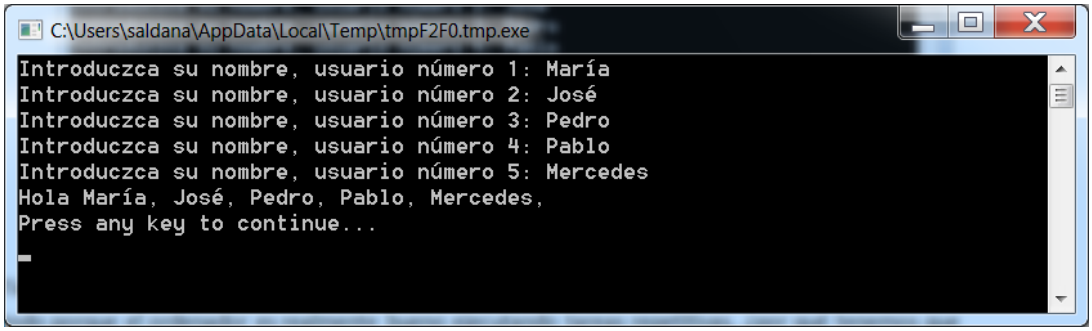

Figura 50 – Utilizando matrices

The above program gives almost the exact same result as the one without arrays, except for the comma at the end of *Mantis*. We can fix that by rewriting the printing loop as:

El programa anterior produce casi el mismo resultado que el que escribimos sin matrices, excepto por la coma al final de *Mercedes*. Podemos arreglando simplemente reescribiendo el bucle de impresión de esta forma:

```
TextWindow.Write("Hola ")
For i = 1 To 5
  TextWindow.Write(nombre[i])
  If i < 5 Then
    TextWindow.Write(", ")
  EndIf
EndFor
TextWindow.WriteLine("")
```

Indexando una matriz

In our previous program you saw that we used numbers as indices to store and retrieve values from the array. It turns out that the indices are not restricted to just numbers and in practice it's very useful to use textual indices too. For example, in the following program, we ask and store various pieces of information about a user and then print out the info that the user asks for.

Como vimos en nuestro programa anterior utilizamos números como índices para almacenar y recuperar valores de la matriz. Los índices no se limitan a sólo números y en la práctica es también muy útil utilizar índices textuales. Por ejemplo, en el siguiente programa, preguntamos y almacenamos distinta información acerca de un usuario para imprimir después la información que el usuario pida.

```
TextWindow.Write("Escriba su nombre: ")
user["nombre"] = TextWindow.Read()
TextWindow.Write("Escriba su edad: ")
user["edad"] = TextWindow.Read()
TextWindow.Write("Escriba la ciudad donde vive: ")
user["ciudad"] = TextWindow.Read()
TextWindow.Write("Escriba su código postal: ")
user["códigopostal"] = TextWindow.Read()

TextWindow.Write("¿Qué información desea? ")
índice = TextWindow.Read()
TextWindow.WriteLine(índice + " = " + user[índice])
```

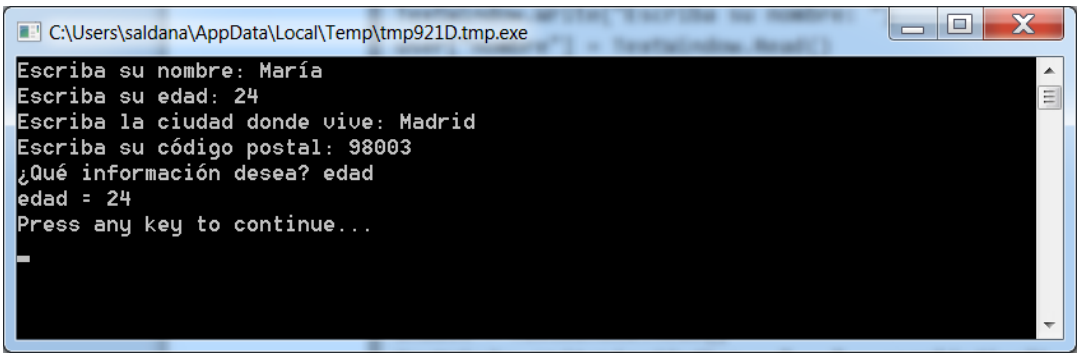

Figura 51 – Utlizando indices no numéricos

Más de una dimensión

Supongamos que desea almacenar el nombre y el número de teléfono de todos tus amigos para poder buscar sus números de teléfono cuando lo necesite, como si fuera una libreta de teléfonos. ¿Cómo escribiríamos un programa de este tipo?

En este caso, tenemos dos conjuntos de índices (también conocidos como dimensión de la matriz). Supongamos que identificamos a cada amigo por su sobrenombre. Este será el primer índice en la matriz. Una vez que usamos el primer índice para obtener la variable de nuestro amigo, el segundo de los índices, **nombre y número de teléfono** nos ayuda a obtener el número de teléfono y el nombre real de ese amigo.

Los índices de las matrices no distinguen entre mayúsculas y minúsculas, al igual que las variables regulares.

The way we store this data would be like this:

Almacenamos los datos de esta manera:

```
amigos["Nuri"]["Nombre"] = "Nuria"
amigos["Nuri"]["Teléfono"] = "123-4567"

amigos["Arthur"]["Nombre"] = "Arturo"
amigos["Arthur"]["Teléfono"] = "890-1234"

amigos["Fran"]["Nombre"] = "Francisco"
amigos["Fran"]["Teléfono"] = "56-7890"
```

Puesto que tenemos dos índices en la misma matriz, **amigos**, esta matriz se conoce como matriz bidimensional.

Una vez que hemos creado este programa, puede escribir el apodo de un amigo y se le mostrará la información almacenada de cada uno de ellos. Aquí tiene el programa completo que lo hace:

```
amigos["Nuri"]["Nombre"] = "Nuria"
amigos["Nuri"]["Teléfono"] = "123-4567"

amigos["Arthur"]["Nombre"] = "Arturo"
amigos["Arthur"]["Teléfono"] = "890-1234"

amigos["Fran"]["Nombre"] = "Francisco"
amigos["Fran"]["Teléfono"] = "56-7890"

TextWindow.Write("Escriba el sobrenombre: ")
sobrenombre = TextWindow.Read()

TextWindow.WriteLine("Nombre: " + amigos[sobrenombre]["Nombre"])
TextWindow.WriteLine("Teléfono: " + amigos[sobrenombre]["Teléfono"])
```

Figura 52 – Una guía de teléfono sencilla

Utilizar matrices para representar cuadrículas

Un uso muy común de las matrices multidimensionales es el de representar cuadrículas y tablas. Las cuadrículas tienen filas y columnas, que pueden encajar perfectamente en un matriz bidimensional. A continuación se muestra un programa sencillo que se presenta cuadrados en una cuadrícula:

```
filas = 8
columnas = 8
tamaño = 40

For f = 1 To filas
  For c = 1 To columnas
    GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
    cuadrados[f][c] = Shapes.AddRectangle(tamaño, tamaño)
    Shapes.Move(cuadrados[f][c], c * tamaño, f * tamaño)
  EndFor
EndFor
```

Este programa agrega rectángulos y los coloca para formar una cuadrícula de 8x8. Además de diseñar estos cuadrados, también los almacena en una matriz. Esto nos permite poder hacer un seguimiento de los cuadrados y usarlos de nuevo cuando lo necesitamos.

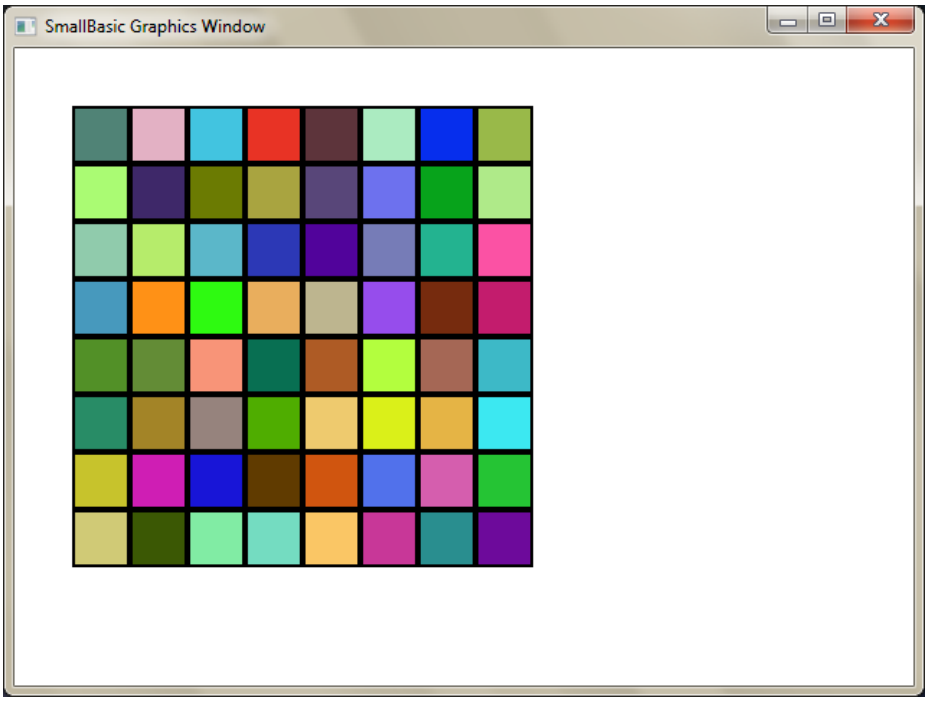

Figura 53 - Diseño de cuadrados en una cuadrícula

Por ejemplo, si agregamos el siguiente código al final del programa anterior hará que los cuadrados se muevan a la esquina superior izquierda.

```
For f = 1 To filas
  For c = 1 To columnas
    Shapes.Animate(cuadrados[f][c], 0, 0, 1000)
    Program.Delay(300)
  EndFor
EndFor
```

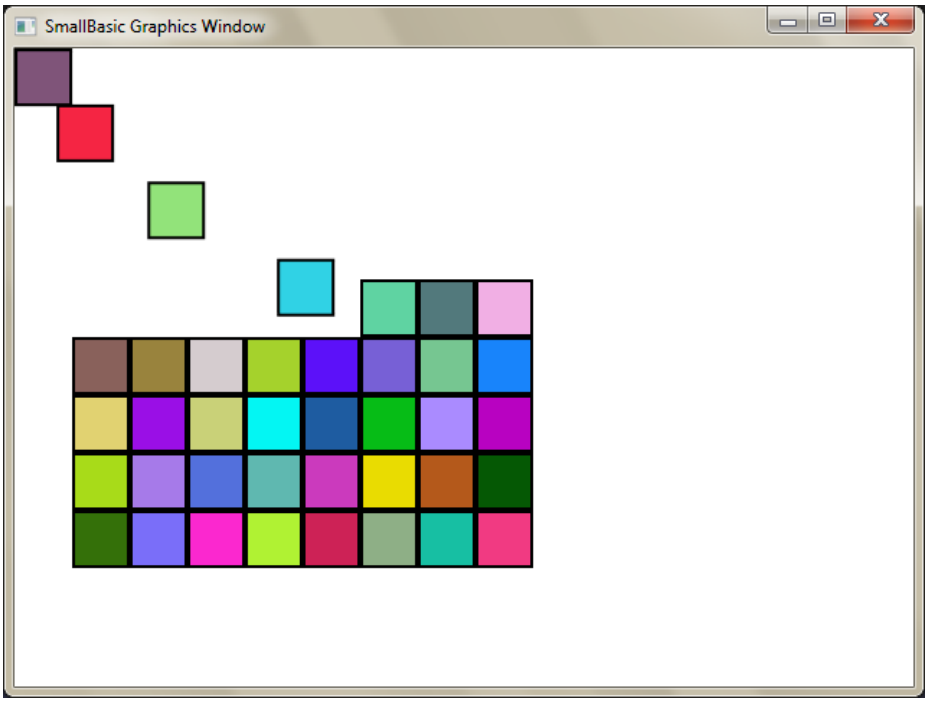

Figura 54 - Seguimiento de los cuadrados en la cuadrícula

Capítulo 11

Eventos e interactividad

En los primeros dos capítulos, introdujimos objetos que tienen *propiedades* y *operaciones*. Además de las propiedades y operaciones, algunos objetos tienen lo que llamamos ***eventos***. Los eventos son como señales que se levantan, por ejemplo, en respuesta a acciones del usuario, como el movimiento del mouse, o hacer clic con él. En cierto sentido, los eventos son opuestos a las operaciones. En el caso de las operaciones, usted como programador las invoca para hacer que el equipo haga algo, mientras que en el caso de los eventos, el ordenador le indica cuándo ha ocurrido algo interesante.

¿Cómo son de útiles los eventos?

Los eventos son fundamentales para introducir interactividad a un programa. Si usted desea permitir a un usuario interactuar con su programa, los eventos son lo que usted usará. Digamos que está escribiendo un programa de las tres en raya. Deseará permitir al usuario elegir su jugada, ¿cierto? Ahí es donde intervienen los eventos: usted recibirá la entrada del usuario en su programa usando eventos. Si esto parece difícil de entender, no se preocupe, vamos a dar una mirada a un ejemplo muy simple que le ayudará a entender qué son los eventos cómo pueden ser usados.

A continuación se muestra un programa muy simple que tiene sólo una sentencia y una subrutina. La subrutina usa la operación *ShowMessage* (mostrar mensaje) en el objeto *GraphicsWindow* para mostrar un cuadro de mensaje al usuario.

```
GraphicsWindow.MouseDown = OnMouseDown 'Al presionar el mouse

Sub OnMouseDown
 GraphicsWindow.ShowMessage("Ha hecho clic.", "Hola")
EndSub
```

La parte interesante a destacar del programa anterior es la línea donde asignamos el nombre de la subrutina al evento **MouseDown** (hace clic con el mouse) del objeto *GraphicsWindow*. Habrá notado que *MouseDown* se parece a una propiedad, excepto en que en lugar de asignarle un valor, le asignamos la subrutina *OnMouseDown*. Ahí está lo especial de los eventos, cuando el evento pasa, se llama a la subrutina automáticamente. En este caso, se llama a la subrutina *OnMouseDown* cada vez que el usuario hace clic con el mouse en *GraphicsWindow*. Adelante, ejecute el programa y pruébelo. Siempre que haga clic en GraphicsWindow con el mouse, aparecerá un cuadro de mensaje como el que se muestra en la siguiente figura.

Figura 55 - Respuesta a un evento

Este tipo de manejo de eventos es muy potente y posibilita programas muy creativos e interesantes. Los programas escritos de esta forma suelen llamarse programas dirigidos por eventos.

Puede modificar la subrutina *OnMouseDown* para hacer otras cosas que mostrar un cuadro de mensaje. Por ejemplo, como en el siguiente programa, puede dibujar grandes círculos azules donde el usuario hace clic con el mouse.

```
GraphicsWindow.BrushColor = "Blue" 'Azul
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
 x = GraphicsWindow.MouseX - 10
 y = GraphicsWindow.MouseY - 10
 GraphicsWindow.FillEllipse(x, y, 20, 20)
EndSub
```

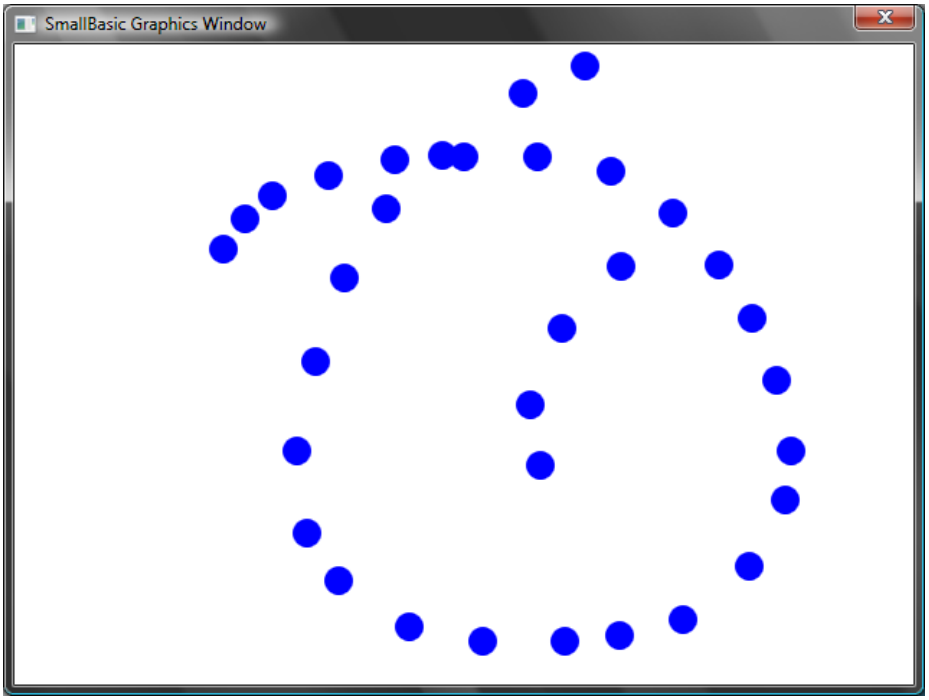

Figura 56 - Usando el evento *OnMouseDown*

Note que en el programa anterior, usamos *MouseX* y *MouseY* para obtener las coordenadas del mouse. Luego las usamos para dibujar un círculo con las coordenadas como centro del círculo.

Usando múltiples eventos

En realidad no hay límite a la cantidad de eventos que puede usar. Hasta puede tener una subrutina que use múltiples eventos. Sin embargo, puede usar un evento sólo una vez. Si trata de asignar dos subrutinas al mismo evento, la segunda gana.

Para ilustrar esto, tomemos el ejemplo anterior y agreguemos una subrutina que controle si se pulsa una tecla o no. Además, hagamos que esta nueva subrutina cambia el color del lápiz, de tal forma que cuando haga clic con el mouse, obtendrá puntos de colores diferentes.

```
GraphicsWindow.BrushColor = "Blue"
GraphicsWindow.MouseDown = OnMouseDown
GraphicsWindow.KeyDown = OnKeyDown 'Al presionar una tecla

Sub OnKeyDown
 GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
EndSub

Sub OnMouseDown
 x = GraphicsWindow.MouseX - 10
 y = GraphicsWindow.MouseY - 10
 GraphicsWindow.FillEllipse(x, y, 20, 20)
EndSub
```

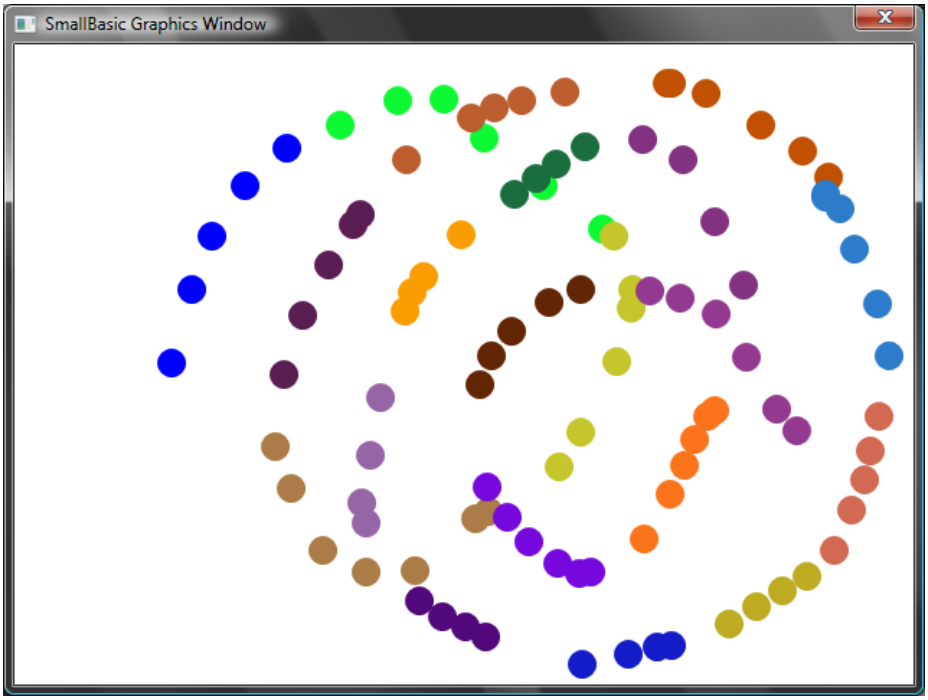

Figura 57 - Usando múltiples eventos

Si ejecuta este programa y hace clic en la ventana, obtendrá un punto azul. Ahora, si presiona cualquier tecla una vez y hace clic de nuevo, obtendrá un punto de un color diferente. Lo que sucede cuando presiona una tecla es que la subrutina *OnKeyDown* (al presionar una tecla) se ejecuta, lo que cambia el color del lápiz a un color aleatorio. Después de eso, cuando hace clic con el mouse, un círculo se dibuja usando el color recién cambiado, resultando en puntos de colores aleatorios.

Un programa de dibujo

Ahora que ya conocemos los eventos y subrutinas, podemos escribir un programa que permita al usuario dibujar en la ventana. Es muy simple escribir dicho programa, dado que podemos separar el problema en pequeñas partes. Como primer paso, vamos a escribir un programa que permita a los usuarios mover el mouse por cualquier parte de la ventana de gráficos, dejando un rastro mientras mueve el mouse.

```
GraphicsWindow.MouseMove = OnMouseMove 'Al mover el mouse

Sub OnMouseMove
 x = GraphicsWindow.MouseX
 y = GraphicsWindow.MouseY
 GraphicsWindow.DrawLine(prevX, prevY, x, y)
 prevX = x
 prevY = y
EndSub
```

Sin embargo, cuando ejecute este programa, la primera línea siempre comienza en la esquina superior izquierda de la ventana (0, 0). Podemos solucionar este problema utilizando el evento *MouseDown* y capturando los valores de *prevX* y *prevY* cuando ocurra ese evento.

Además, realmente sólo necesitamos dejar el rastro cuando el usuario mantiene el botón presionado. Las demás veces, no deberíamos dibujar una línea. Para obtener este comportamiento, usaremos la propiedad *IsLeftButtonDown* (el botón izquierdo del mouse está presionado) en el objeto *Mouse*. Esta propiedad dice si el botón izquierdo del mouse está siendo presionado o no. Si el valor es cierto, dibujaremos la línea, de lo contrario, no.

```
GraphicsWindow.MouseMove = OnMouseMove
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
 prevX = GraphicsWindow.MouseX
 prevY = GraphicsWindow.MouseY
EndSub

Sub OnMouseMove
 x = GraphicsWindow.MouseX
 y = GraphicsWindow.MouseY
 If (Mouse.IsLeftButtonDown) Then
  GraphicsWindow.DrawLine(prevX, prevY, x, y)
 EndIf
 prevX = x
 prevY = y
EndSub
```

Apéndice A

Ejemplos divertidos

Fractal con la tortuga

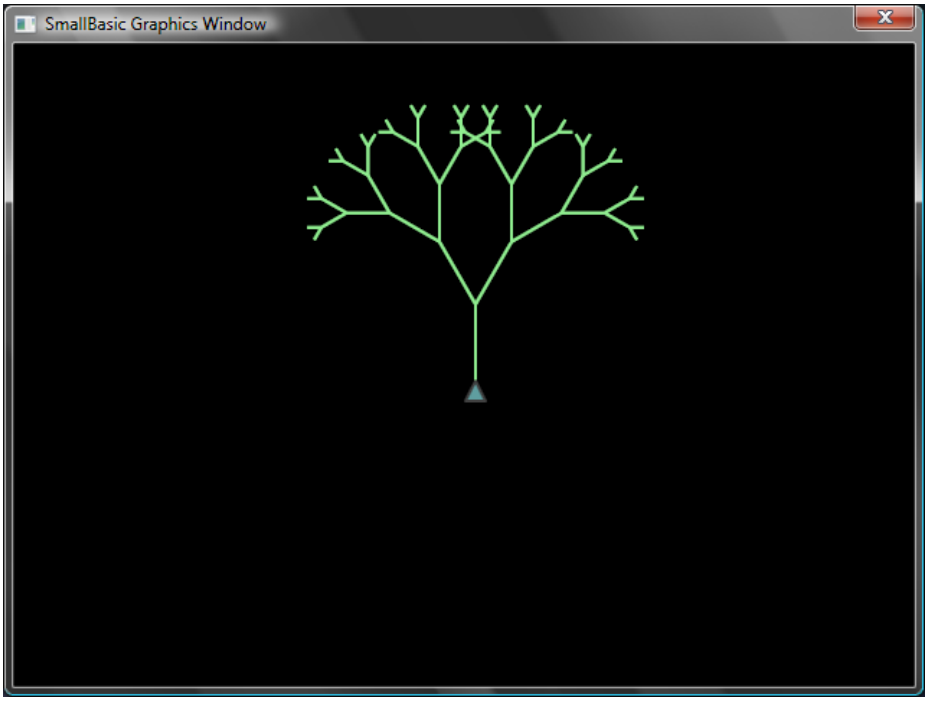

Figura 58 – La tortuga dibuja un fractal con forma de árbol

```
ángulo = 30
delta = 10
distancia = 60
Turtle.Speed = 9
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightGreen"
DrawTree()

Sub DrawTree
 If (distancia > 0) Then
 Turtle.Move(distancia)
 Turtle.Turn(ángulo)

 Stack.PushValue("distancia", distancia)
 distance = distancia - delta
 DrawTree()
 Turtle.Turn(-ángulo * 2)
 DrawTree()
 Turtle.Turn(ángulo)
 distancia = Stack.PopValue("distancia")

 Turtle.Move(-distancia)
 EndIf
EndSub
```

Fotografías de Flickr

Figura 59 – Mostrar fotografías de Flickr

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
 foto = Flickr.GetRandomPicture("montaña, río")
 GraphicsWindow.DrawResizedImage(foto, 0, 0, 640, 480)
EndSub
```

Fondo de escritorio dinámico

```
For i = 1 To 10
 foto = Flickr.GetRandomPicture("montaña")
 Desktop.SetWallPaper(foto)
 Program.Delay(10000)
EndFor
```

El juego del Paddle

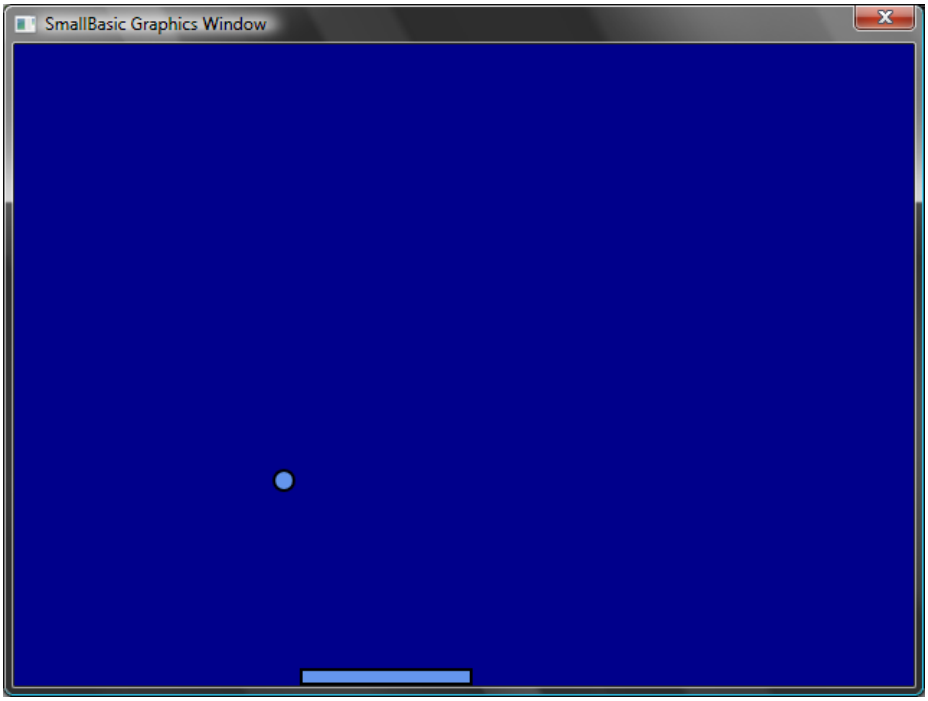

Figura 60 – El juego del Paddle

```
GraphicsWindow.BackgroundColor = "DarkBlue"
paddle = Shapes.AddRectangle(120, 12)
bola = Shapes.AddEllipse(16, 16)
GraphicsWindow.MouseMove = OnMouseMove

x = 0
y = 0
deltaX = 1
deltaY = 1

EjecutarBucle:
 x = x + deltaX
 y = y + deltaY

 gw = GraphicsWindow.Width
 gh = GraphicsWindow.Height
 If (x >= gw - 16 or x <= 0) Then
  deltaX = -deltaX
 EndIf
 If (y <= 0) Then
  deltaY = -deltaY
 EndIf

 padX = Shapes.GetLeft(paddle)
 If (y = gh - 28 and x >= padX and x <= padX + 120) Then
  deltaY = -deltaY
 EndIf

 Shapes.Move(bola, x, y)
 Program.Delay(5)

 If (y < gh) Then
  Goto EjecutarBucle
 EndIf

GraphicsWindow.ShowMessage("Ha perdido", "Paddle")

Sub OnMouseMove
 paddleX = GraphicsWindow.MouseX
 Shapes.Move(paddle, paddleX - 60, GraphicsWindow.Height - 12)
EndSub
```

Apéndice B

Colores

A continuación se muestra una lista de nombres de colores compatibles con Small Basic. Están clasificados en función de su matiz.

Rojos

IndianRed	#CD5C5C
LightCoral	#F08080
Salmon	#FA8072
DarkSalmon	#E9967A
LightSalmon	#FFA07A
Crimson	#DC143C
Red	#FF0000
FireBrick	#B22222
DarkRed	#8B0000

Rosas

Pink	#FFC0CB
LightPink	#FFB6C1
HotPink	#FF69B4
DeepPink	#FF1493
MediumVioletRed	#C71585

PaleVioletRed	#DB7093

Naranjas

LightSalmon	#FFA07A
Coral	#FF7F50
Tomato	#FF6347
OrangeRed	#FF4500
DarkOrange	#FF8C00
Orange	#FFA500

Amarillos

Gold	#FFD700
Yellow	#FFFF00
LightYellow	#FFFFE0
LemonChiffon	#FFFACD
LightGoldenrodYellow	#FAFAD2
PapayaWhip	#FFEFD5
Moccasin	#FFE4B5
PeachPuff	#FFDAB9
PaleGoldenrod	#EEE8AA
Khaki	#F0E68C
DarkKhaki	#BDB76B

Púrpuras

Lavender	#E6E6FA
Thistle	#D8BFD8
Plum	#DDA0DD
Violet	#EE82EE
Orchid	#DA70D6
Fuchsia	#FF00FF
Magenta	#FF00FF
MediumOrchid	#BA55D3
MediumPurple	#9370DB

BlueViolet	#8A2BE2
DarkViolet	#9400D3
DarkOrchid	#9932CC
DarkMagenta	#8B008B
Purple	#800080
Indigo	#4B0082
SlateBlue	#6A5ACD
DarkSlateBlue	#483D8B
MediumSlateBlue	#7B68EE

Verdes

GreenYellow	#ADFF2F
Chartreuse	#7FFF00
LawnGreen	#7CFC00
Lime	#00FF00
LimeGreen	#32CD32
PaleGreen	#98FB98
LightGreen	#90EE90
MediumSpringGreen	#00FA9A
SpringGreen	#00FF7F
MediumSeaGreen	#3CB371
SeaGreen	#2E8B57
ForestGreen	#228B22
Green	#008000
DarkGreen	#006400
YellowGreen	#9ACD32
OliveDrab	#6B8E23
Olive	#808000
DarkOliveGreen	#556B2F
MediumAquamarine	#66CDAA
DarkSeaGreen	#8FBC8F
LightSeaGreen	#20B2AA

DarkCyan	#008B8B
Teal	#008080

Azules

Aqua	#00FFFF
Cyan	#00FFFF
LightCyan	#E0FFFF
PaleTurquoise	#AFEEEE
Aquamarine	#7FFFD4
Turquoise	#40E0D0
MediumTurquoise	#48D1CC
DarkTurquoise	#00CED1
CadetBlue	#5F9EA0
SteelBlue	#4682B4
LightSteelBlue	#B0C4DE
PowderBlue	#B0E0E6
LightBlue	#ADD8E6
SkyBlue	#87CEEB
LightSkyBlue	#87CEFA
DeepSkyBlue	#00BFFF
DodgerBlue	#1E90FF
CornflowerBlue	#6495ED
MediumSlateBlue	#7B68EE
RoyalBlue	#4169E1
Blue	#0000FF
MediumBlue	#0000CD
DarkBlue	#00008B
Navy	#000080
MidnightBlue	#191970

Marrones

Cornsilk	#FFF8DC

BlanchedAlmond	#FFEBCD
Bisque	#FFE4C4
NavajoWhite	#FFDEAD
Wheat	#F5DEB3
BurlyWood	#DEB887
Tan	#D2B48C
RosyBrown	#BC8F8F
SandyBrown	#F4A460
Goldenrod	#DAA520
DarkGoldenrod	#B8860B
Peru	#CD853F
Chocolate	#D2691E
SaddleBrown	#8B4513
Sienna	#A0522D
Brown	#A52A2A
Maroon	#800000

Blancos

White	#FFFFFF
Snow	#FFFAFA
Honeydew	#F0FFF0
MintCream	#F5FFFA
Azure	#F0FFFF
AliceBlue	#F0F8FF
GhostWhite	#F8F8FF
WhiteSmoke	#F5F5F5
Seashell	#FFF5EE
Beige	#F5F5DC
OldLace	#FDF5E6
FloralWhite	#FFFAF0
Ivory	#FFFFF0
AntiqueWhite	#FAEBD7

Linen	#FAF0E6
LavenderBlush	#FFF0F5
MistyRose	#FFE4E1

Grises

Gainsboro	#DCDCDC
LightGray	#D3D3D3
Silver	#C0C0C0
DarkGray	#A9A9A9
Gray	#808080
DimGray	#696969
LightSlateGray	#778899
SlateGray	#708090
DarkSlateGray	#2F4F4F
Black	#000000