Microsoft Small Basic

Introduzione alla Programmazione

Capitolo 1

Introduzione

Small Basic e la programmazione

La programmazione è definita come il processo di creazione di software per computer, utilizzando i linguaggi di programmazione. Proprio come parliamo e comprendiamo l'Inglese, il francese o lo spagnolo, i computer possono comprendere i programmi scritti in certi linguaggi. Questi sono chiamati linguaggi di programmazione. In principio c'erano solo un paio di linguaggi ed erano davvero semplici da imparare e comprendere. Ma, quanto più i computer ed i software diventavano sofisticati, tanto più velocemente si evolvevano i linguaggi di programmazione, arricchendosi lungo la strada di concetti più complessi. Di conseguenza la maggior parte dei linguaggi di programmazione moderni ed i loro concetti, sono piuttosto impegnativi da cogliere per i principianti.

Small Basic è un linguaggio progettato per rendere la programmazione estremamente semplice, accessibile e divertente per i principianti. L'intenzione di Small Basic è di abbattere le barriere e di fornire un trampolino di lancio verso le meraviglie della programmazione dei computer.

Ambiente di Small Basic

Iniziamo con una rapida introduzione all'ambiente di Small Basic. Quando si avvia Small Basic, si vedrà una finestra simile alla seguente figura.

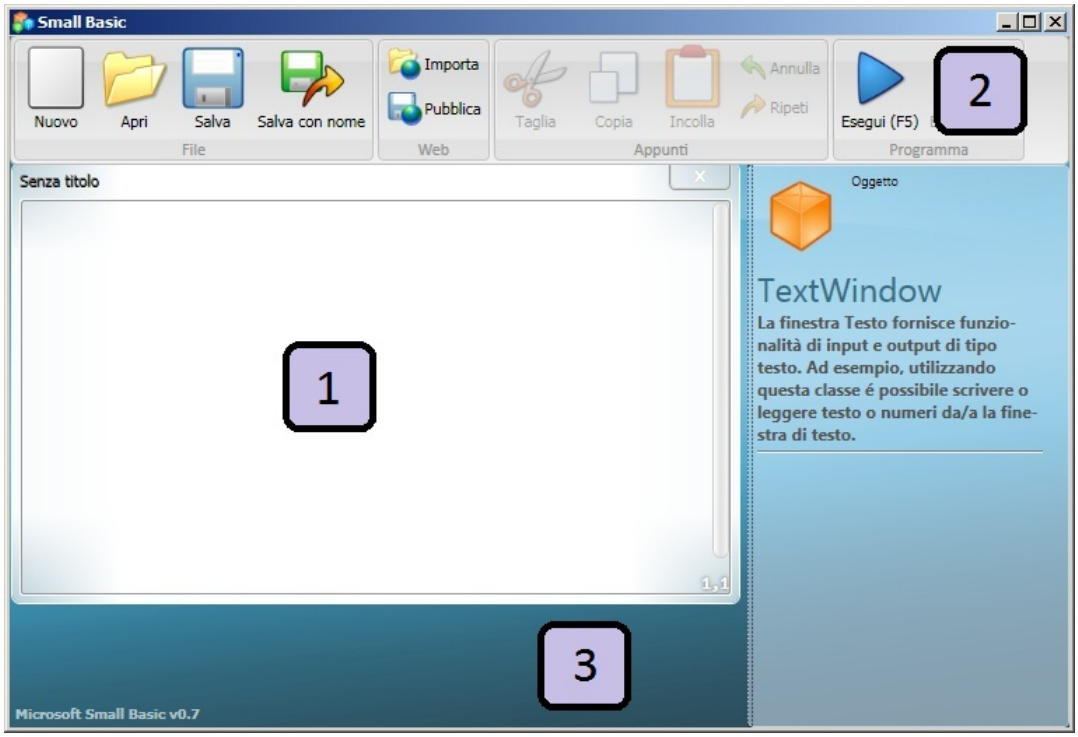

Figura 1 – Ambiente Small Basic

Questo è l'ambiente Small Basic, in cui scriveremo ed eseguiremo i nostri programmi. Questo ambiente ha parecchi elementi distinti che sono identificati dai numeri.

L'**Editor**, identificato da [1] è il posto in cui scriveremo i nostri programmi Small Basic. Quando si apre un programma di esempio o un programma precedentemente salvato, apparirà in questo editor. Si può poi modificare e salvare il programma per usi successivi.

Si può anche aprire e lavorare con più di un programma alla volta. Ogni programma con cui si lavora viene visualizzato in un editor separato. L'editor che contiene il programma su cui si sta lavorando è chiamato *editor attivo*.

La **Barra degli strumenti**, identificata da [2] è utilizzata per dare comandi all'*editor attivo* o all'ambiente. Impareremo i vari comandi nella barra degli strumenti strada facendo.

La **Superficie**, identificata da [3] è l'area in cui vengono aperte tutte le finestre di editor.

Il primo programma

Ora che abbiamo acquisito familiarità con l'ambiente di Small Basic, iniziamo a programmare. Come abbiamo appena notato, l'editor è l'area in cui scrivere i nostri programmi. Quindi procediamo scrivendo la seguente linea nell'editor.

```
TextWindow.WriteLine("Hello World")
```

Questo è il nostro primo programma in Small Basic e, se digitato correttamente, dovremmo avere qualcosa di simile alla seguente figura.

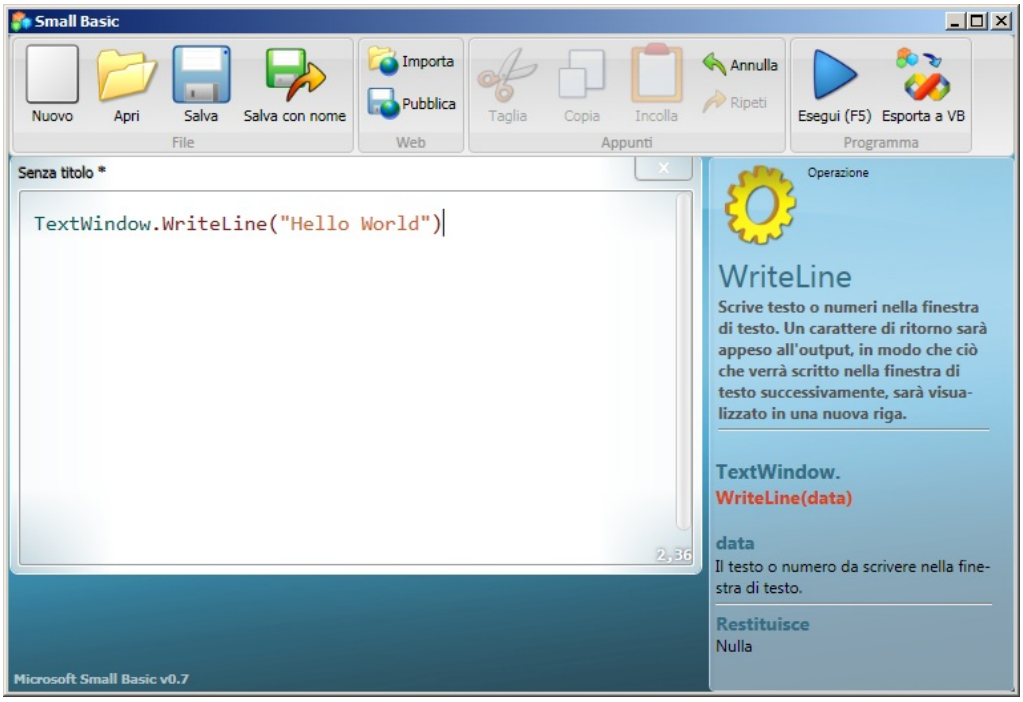

Figura 2 – Primo Programma

Ora che abbiamo digitato il nostro nuovo programma, eseguiamolo per vedere cosa succede. Possiamo eseguire il nostro programma sia cliccando sul pulsante *Esegui,* sulla Barra degli strumenti, che utilizzando la scorciatoia da tastiera F5. Se è andato tutto bene, il nostro programma dovrebbe essere eseguito con il risultato mostrato sotto.

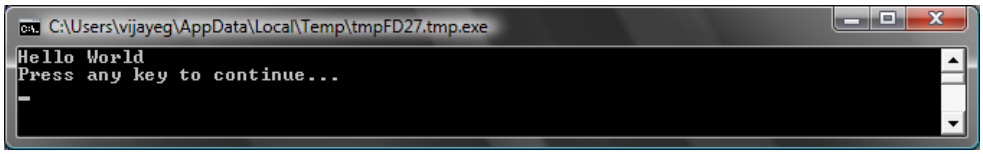

Figura 3 – Output del Primo Programma

Congratulazioni! Hai appena scritto ed eseguito il primo programma Small Basic. Un programma davvero piccolo e semplice, ma non di meno un grande passo per diventare un vero programmatore! C'è solo un altro dettaglio da considerare prima di creare programmi più grandi. Dobbiamo capire cosa è accaduto – cosa esattamente abbiamo comunicato al computer e come faceva il computer a sapere cosa fare? Nel prossimo capitolo, analizzeremo il programma

Mentre digitavi il tuo primo programma, avrai notato che è apparso un popup con una lista di voci (Figura 4). Questo è chiamato "intellisense" e aiuta a digitare i propri programmi più velocemente. È possibile scorrere la lista utilizzando i tasti freccia Su/Giù, e quando si trova quel che si vuole, si può premere il tasto Enter per inserire nel programma la voce selezionata.

appena scritto, in modo da poter giungere a una piena comprensione.

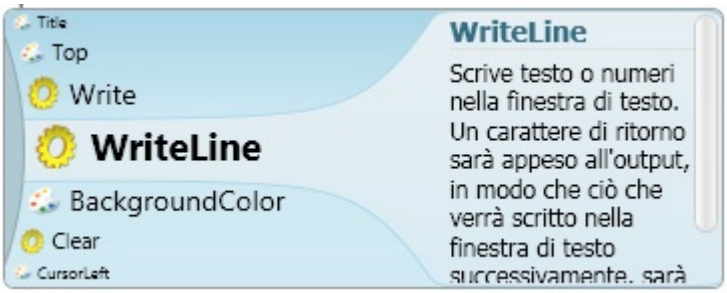

Figura 4 - Intellisense

Salvare il Programma

Se si desidera chiudere Small Basic e tornare in seguito a lavorare sul programma, é necessario salvarlo. È buona pratica salvare i programmi di tanto in tanto, in modo da non perdere le informazioni in seguito ad uno spegnimento accidentale o a una caduta di tensione. Puoi salvare il programma corrente sia cliccando sull'icona Salva sulla Barra degli strumenti che utilizzando il tasto di scelta rapida "Ctrl+S" (premendo il tasto S mentre si tiene premuto il tasto Ctrl).

Capitolo 2

Capire il Primo Programma

Cos'è davvero un Programma per Computer?

Un programma è un insieme di istruzioni per il computer. Queste istruzioni dicono al computer precisamente cosa fare, e il computer segue sempre queste istruzioni. Proprio come le persone, i computer possono solo seguire le istruzioni se specificate in un linguaggio che possono capire. Questi sono chiamati linguaggi di programmazione. Ci sono moltissimi linguaggi che i computer possono capire e **Small Basic** è uno di questi.

Immagina una conversazione tra te e un tuo amico. Tu e il tuo amico usereste le parole, organizzate in frasi per trasmettervi delle informazioni. Analogamente, i linguaggi di programmazione contengono insiemi di parole, che possono essere organizzate in frasi, che trasmettono informazioni al computer. I programmi sono fondamentalmente insiemi di frasi (qualche volta solo un paio e altre volte molte centinaia) che hanno senso sia per il programmatore che per il computer.

> *Ci sono molti linguaggi che il computer può capire. Java, C++, Pyton, VB, ecc. sono tutti potenti e moderni linguaggi per computer che sono utilizzati per lo sviluppo di software, da quello semplice a quello complesso.*

I Programmi Small Basic

Un tipico programma Small Basic é costituito da un insieme di *istruzioni*. Ogni riga del programma rappresenta un'istruzione e ogni istruzione é un comando per il computer. Quando diciamo al computer di eseguire un programma Small Basic, questo prende il programma e legge la prima istruzione. Capisce cosa stiamo cercando di fargli fare ed esegue la nostra istruzione. Una volta eseguita la prima istruzione, l'esecuzione ritorna al programma ove viene letta ed eseguita la seconda riga. Il computer continua così fintanto che non è raggiunta la fine del programma. Ovvero quando il nostro programma termina.

Torniamo al primo programma

Ecco il nostro primo programma:

```
TextWindow.WriteLine("Hello World")
```

È un programma molto semplice, costituito da una sola istruzione. Questa istruzione dice al computer di scrivere una riga di testo, che è **Hello World**, nella Finestra Testo.

Letteralmente lo traduce al computer come:

```
Scrivi Hello World
```

Potresti aver già notato che l'istruzione può essere divisa in segmenti più piccoli, proprio come le frasi possono essere divise in parole. Nella prima istruzione abbiamo 3 distinti segmenti:

a) TextWindow
b) WriteLine
c) "Hello World"

Il punto, le parentesi, e gli apici sono tutti segni di punteggiatura che devono essere posti nei punti giusti dell'istruzione perché il computer capisca il cosa vogliamo comunicargli.

Ricordi la finestra nera apparsa quando hai eseguito il primo programma? Quella finestra nera è chiamata **Finestra Testo (TextWindow)** o Console. È il posto in cui finiscono i risultati di questo programma. **TextWindow**, nel nostro programma, è un *oggetto*. Esistono diversi oggetti da utilizzare nei nostri programmi. Possiamo realizzare diverse *operazioni* su questi oggetti. Potresti aver notato che l'operazione WriteLine è seguita da **Hello World** all'interno dei doppi apici. Questo testo è passato all'operazione WriteLine, che successivamente lo mostra all'utente. Questo è un *input* per l'operazione. Alcune operazioni accettano uno o più input mentre altre non ne hanno nessuno.

Il secondo programma

Ora che è stato compreso il nostro primo programma, abbelliamolo aggiungendo un po' di colore.

```
TextWindow.ForegroundColor = "Yellow"
TextWindow.WriteLine("Hello World")
```

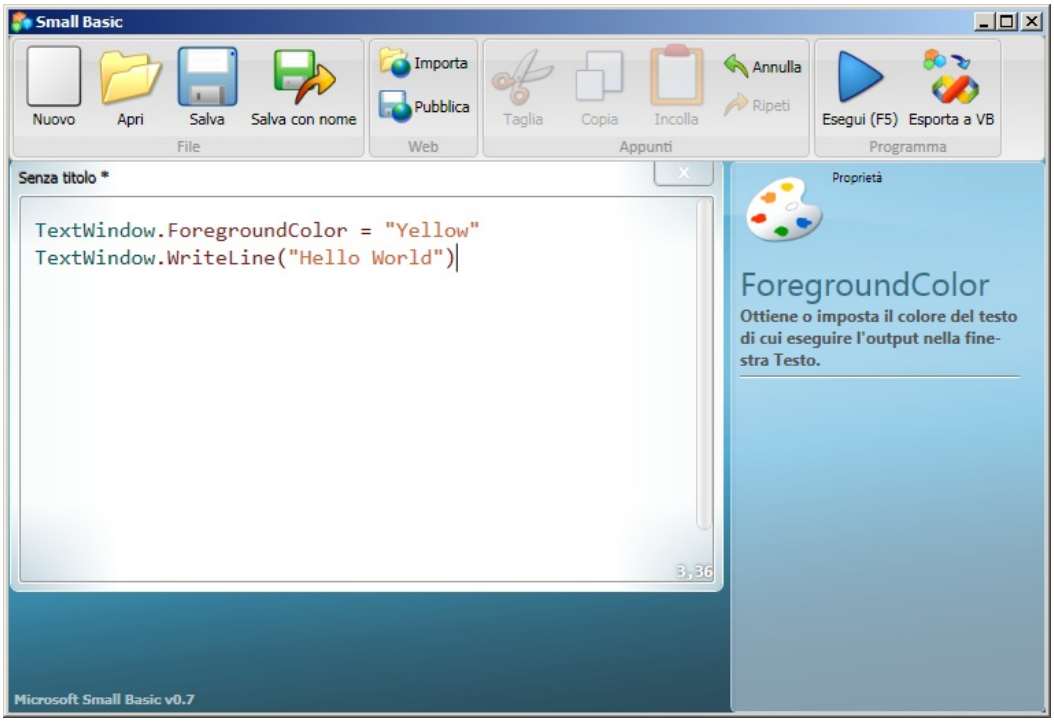

Figura 5 – Aggiungere i Colori

Quando esegui il programma qui sopra, noterai che è visualizzata la stessa frase all'interno della Finestra testo ma questa volta è visualizzata in giallo invece che in grigio come avveniva prima.

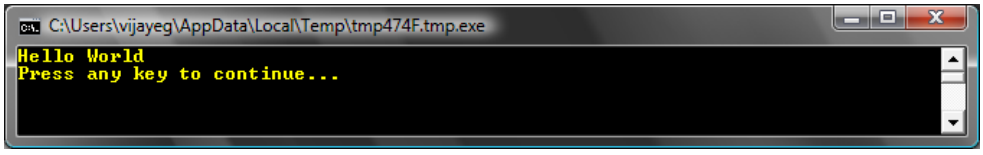

Figura 6 - Hello World in giallo

Nota la nuova istruzione aggiunta al programma originale. Utilizza una nuova parola, *ForegroundColor* cui viene assegnato il valore *"Yellow"* (giallo). La differenza tra ForegroundColor e l'operazione WriteLine è che ForegroundColor non accetta nessun input o non ha bisogno di alcun parametro. Invece è seguita da un simbolo di "uguale a" e una parola. Definiremo ForegroundColor come una *proprietà* di TextWindow. Segue un elenco di valori validi per la proprietà ForegroundColor. Prova a sostituire "Yellow" con uno di questi e osserva il risultato – non dimenticare i doppi apici, sono punteggiatura necessaria.

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

Capitolo 3

Introduzione alle Variabili

Utilizzare le Variabili nel programma

Non sarebbe bello se il nostro programma potesse dire “Hello” seguito dal nome dell’utente al posto di un generico “Hello World”? Per ottenere questo risultato dobbiamo prima chiedere all’utente il proprio nome, memorizzarlo da qualche parte e infine visualizzare “Hello” seguito dal nome dell’utente. Vediamo come farlo:

```
TextWindow.Write("Enter your Name: ")
name = TextWindow.Read()
TextWindow.WriteLine("Hello " + name)
```

Quando digiti ed esegui questo programma, vedrai un risultato simile al seguente:

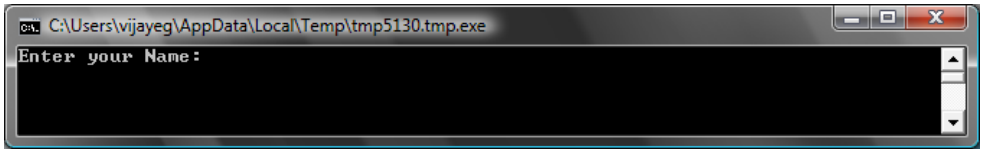

Figura 7 – Chiedere il Nome dell’Utente

E quando inserisci il tuo nome e premi il tasto Invio, vedrai un risultato simile al seguente:

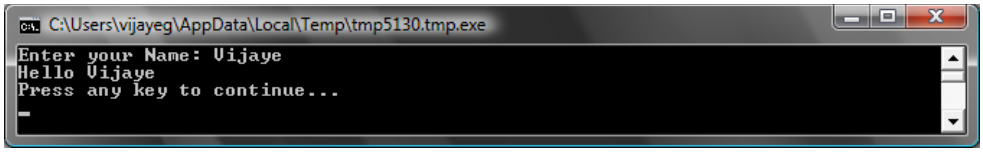

Figura 8 – Un Caloroso Benvenuto

Se esegui ancora il programma, ti sarà posta la stessa domanda. Puoi inserire un nome differente e vedrai Hello seguito da quel nome.

Analisi del programma

Nel programma appena eseguito, la linea che potrebbe aver catturato la tua attenzione è:

```
name = TextWindow.Read()
```

Read() appare proprio come *WriteLine()*, ma senza input. È un'operazione che dice semplicemente al computer di aspettare che l'utente inserisca qualcosa e prema il tasto Invio. Una volta che l'utente preme il tasto Invio, viene prelevato ciò che l'utente ha inserito e viene restituito al programma. La parte interessante è che, qualunque cosa abbia inserito l'utente è ora memorizzato nella *variabile* chiamata **name**. Una *variabile* è definita come una locazione in cui è possibile memorizzare temporaneamente dei valori, da utilizzare successivamente. Nella riga precedente, **name** è stata utilizzata per memorizzare il nome dell'utente.

> Write, *proprio come* WiteLine *è un'altra operazione sulla Console. Write permette di scrivere qualcosa sulla Console ma permette al testo successivo di rimanere sulla stessa linea del testo corrente.*

La riga successiva è anche interessante:

```
TextWindow.WriteLine("Hello " + name)
```

Questo è il posto in cui utilizziamo il valore memorizzato nella nostra variabile, **name**. Preleviamo il valore in **name,** lo appendiamo a "Hello" e lo scriviamo nella Finestra Testo.

Una volta che la variabile è impostata, puoi riutilizzarla tutte le volte che vuoi. Ad esempio, puoi fare quanto segue:

```
TextWindow.Write("Enter your Name: ")
name = TextWindow.Read()
TextWindow.Write("Hello " + name + ".  ")
TextWindow.WriteLine("How are you doing " + name + "?")
```

E vedrai il seguente risultato:

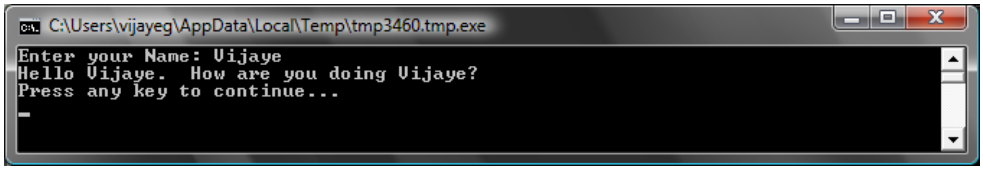

Figura 9 – Riutilizzare una Variabile

Regole per i nomi delle Variabili

Le Variabili hanno nomi associati a esse ed è il modo con cui vengono indentificate. Ci sono delle semplici regole e qualche buona linea guida per definire i nomi delle variabili:

1. Il nome deve iniziare con una lettera e non può coincidere con alcuna delle parole chiave come **if**, **for**, **then**, ecc.
2. Un nome può contenere qualunque combinazione di lettere, cifre e caratteri di sottolineatura.
3. È utile che il nome delle variabili sia significativo – dato che le variabili possono essere lunghe quanto vuoi, utilizza nomi di variabili che descrivono il loro scopo.

Giocare con i numeri

Abbiamo appena visto come utilizzare le variabili per memorizzare il nome dell'utente. Nei prossimi programmi, vedremo come sia possibile memorizzare e manipolare i numeri con le variabili. Iniziamo con un semplice programma:

```
number1 = 10
number2 = 20
number3 = number1 + number2
TextWindow.WriteLine(number3)
```

Quando esegui questo programma otterrai il seguente risultato:

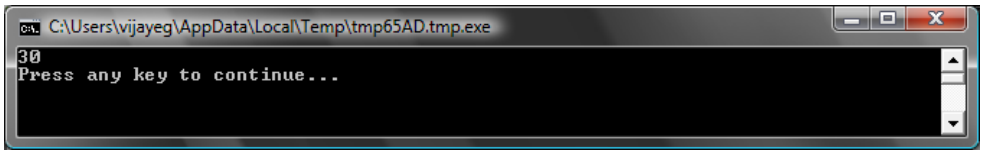

Figura 10 – Sommare due numeri

Nella prima linea del programma, stai assegnando un valore pari a 10 alla variabile **number1**. Nella seconda linea, stai assegnando un valore pari a 20 alla variabile **number2**. Nella terza linea, stai sommando **number1** e **number2** e poi assegni il

Si noti che i numeri non hanno apici attorno a se. Per i numeri, gli apici non sono necessari. Ne hai bisogno solo quando stai utilizzando del testo.

risultato a **number3**. Quindi, in questo caso, **number3** avrà un valore pari a 30. E questo è quello che visualizzeremo nella Finestra Testo.

Proviamo ora a modificare leggermente il programma e vediamo i risultati:

```
number1 = 10
number2 = 20
number3 = number1 * number2
TextWindow.WriteLine(number3)
```

Il programma qui sopra moltiplicherà **number1** per **number2** e memorizzerà il risultato in **number3**. Sotto puoi vedere il risultato di questo programma:

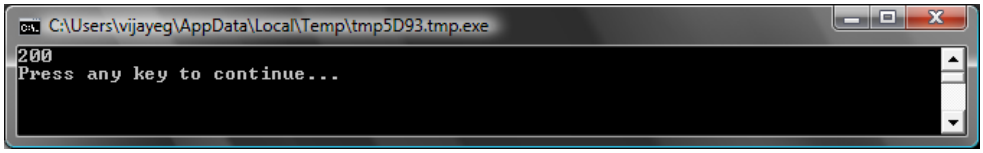

Figura 11 – Moltiplicare due numeri

Allo stesso modo puoi sottrarre o dividere numeri. Ecco la sottrazione:

```
number3 = number1 - number2
```

E il simbolo per la divisione è '/'. Il programma é:

```
number3 = number1 / number2
```

E il risultato di questa divisione sarà:

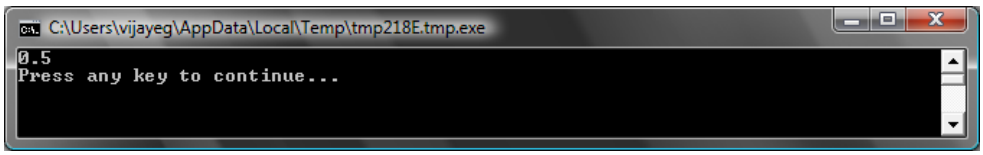

Figura 12 – Dividere due numeri

Un semplice Convertitore di temperatura

Per il prossimo programma utilizzeremo la formula $℃ = \frac{5(°F-32)}{9}$ per convertire le temperature Fahrenheit in temperature Celsius.

Prima otterremo la temperatura in Fahrenheit dall'utente e la memorizzeremo in una variabile. Esiste un'operazione speciale che permette di leggere i numeri inseriti dall'utente ed è **TextWindow.ReadNumber**.

```
TextWindow.Write("Enter temperature in Fahrenheit: ")
fahr = TextWindow.ReadNumber()
```

Una volta memorizzata in una variabile la temperatura Fahrenheit, possiamo convertirla in Celsius come segue:

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

Le parentesi indicano al computer di calcolare prima **fahr - 32** e poi elaborare il resto. Quello che dobbiamo fare adesso è visualizzare il risultato all'utente. Mettendo tutto insieme, otteniamo questo programma:

```
TextWindow.Write("Enter temperature in Fahrenheit: ")
fahr = TextWindow.ReadNumber()
celsius = 5 * (fahr - 32) / 9
TextWindow.WriteLine("Temperature in Celsius is " + celsius)
```

E il risultato di questo programma sarà:

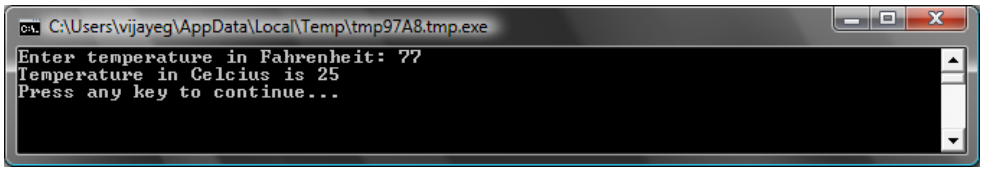

Figura 13 – Convertitore di temperatura

Capitolo 4

Condizioni e Ramificazioni

Tornando al nostro primo programma, non sarebbe interessante se invece di visualizzare il generico *Hello World*, potesse visualizzare *Good morning World* (buon giorno mondo), o *Good Evening World* (buona sera mondo) a seconda dell'ora del giorno? Per il nostro prossimo programma, faremo visualizzare al computer *Good Morinig World* se l'ora è precedente le 12 e *Good Evening World* se l'ora è successiva alle 12.

```
If (Clock.Hour < 12) Then
  TextWindow.WriteLine("Good Morning World")
EndIf
If (Clock.Hour >= 12) Then
  TextWindow.WriteLine("Good Evening World")
EndIf
```

A seconda di quando esegui il programma vedrai I seguenti risultati:

Figura 14 - Good Morning World

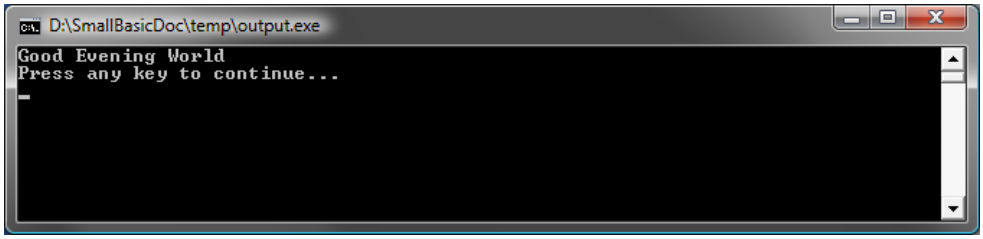

Figura 15 - Good Evening World

Analizziamo le prime tre righe del programma. Avrai già immaginato quello che comunicano al computer ovvero che se Clock.Hour è minore di 12, visualizza "Good Morning World". Le parole I**f**, **Then** e **EndIf** sono parole speciali, comprese dal computer quando il programma è in esecuzione. La parola **If** è sempre seguita da una condizione, che in questo caso è (**Clock.Hour < 12**). Ricorda che le parentesi sono necessarie al computer per capire le tue intenzioni. La condizione è seguita da **Then** e l'operazione da eseguire. Dopo l'operazione viene **EndIf**. Questo dice al computer che l'esecuzione condizionale è terminata.

Tra **Then** ed **EndIf**, ci potrebbe essere più di un'operazione e il computer le eseguirebbe tutte nel caso in cui la condizione fosse valida. Per esempio, potresti scrivere qualcosa del genere:

```
If (Clock.Hour < 12) Then
  TextWindow.Write("Good Morning. ")
  TextWindow.WriteLine("How was breakfast?")
EndIf
```

Else

Nel programma all'inizio di questo capitolo, potresti aver notato che la seconda condizione è ridondante. Il valore di **Clock.Hour** può essere minore di 12 oppure no. Non abbiamo davvero bisogno di fare il secondo controllo. In questi occasioni, possiamo abbreviare le due istruzioni **If…Then…EndIf** in una sola, utilizzando la parola chiave parola **Else**.

Se riscrivessimo quel programma utilizzando **Else**, ecco come apparirebbe:

```
If (Clock.Hour < 12) Then
  TextWindow.WriteLine("Good Morning World")
Else
  TextWindow.WriteLine("Good Evening World")
EndIf
```

Questo programma farà esattamente la stessa cosa dell'altro, il che ci da una lezione molto importante nella programmazione dei computer:

> *Nella programmazione, ci sono molti modi di fare la stessa cosa. Qualche volta un modo ha più senso di un altro. La scelta è lasciata al programmatore. Quando avrai scritto più programmi e acquisito maggiore esperienza, inizierai a notare queste differenti tecniche e i loro vantaggi e svantaggi.*

Formattazione rientrata

In tutti gli esempi hai potuto vedere come le istruzioni tra *If, Else* e *EndIf* sono rientrate (allineate più a destra). Questa formattazione rientrata non è necessaria. Il computer capirebbe il programma anche senza. D'altra parte, ci aiuta a vedere e capire più facilmente la struttura del programma. Quindi, è ritenuta buona norma formattare con un rientro le istruzioni tra tali blocchi.

Pari o dispari

Ora che abbiamo acquisito nel nostro bagaglio l'istruzione **If…Then…Else…EndIf**, scriviamo un programma che dato un numero, ci dica se é pari (even) o dispari (odd).

```
TextWindow.Write("Enter a number: ")
num = TextWindow.ReadNumber()
remainder = Math.Remainder(num, 2)
If (remainder = 0) Then
  TextWindow.WriteLine("The number is Even")
Else
  TextWindow.WriteLine("The number is Odd")
EndIf
```

Quando esegui questo programma, avrai un risultato simile questo:

Figura 16 – Pari o dispari

In questo programma, abbiamo introdotto un'altra nuova e utile operazione, **Math.Remainder**. Come puoi già aver immaginato, **Math.Reminder** divide il primo numero per il secondo e restituisce il resto.

Ramificazioni

Se ricordi, nel secondo capitolo hai appreso che il computer elabora un programma una istruzione alla volta, in ordine dall'alto verso il basso. D'altra parte, c'è una speciale istruzione che può far saltare l'esecuzione a una istruzione al di fuori dell'ordine prestabilito. Dai un'occhiata al prossimo programma.

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

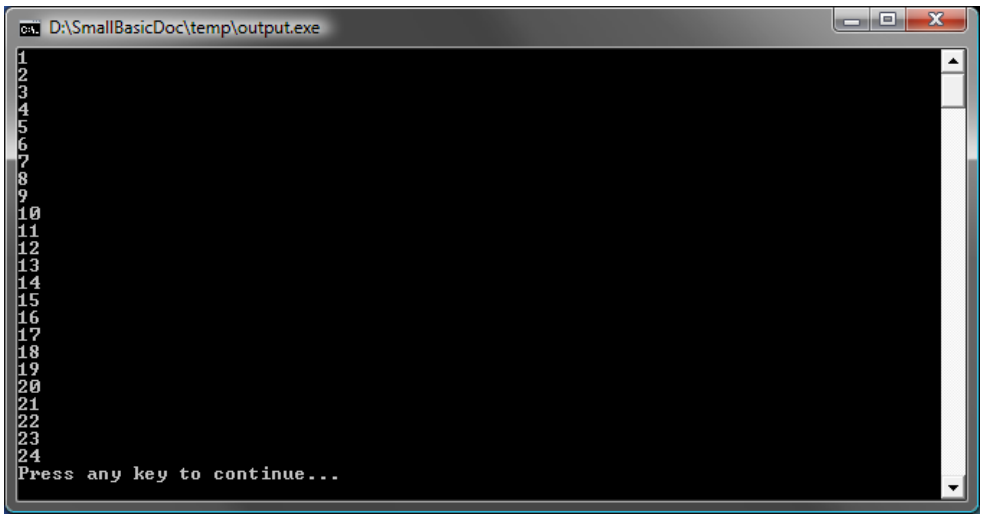

Figura 17 – Utilizzo di Goto

Nel programma, abbiamo assegnato il valore 1 alla variabile **i**. Abbiamo quindi aggiunto una nuova istruzione che termina con due punti (:)

```
start:
```

Questa è chiamata e*tichetta* (in inglese, *label*). Le etichette sono come segnalibri che il computer comprende. Puoi utilizzare il segnalibro quando vuoi e puoi aggiungere tutte le etichette che desideri nel tuo programma, fin tanto che i loro nomi sono univoci.

Un'altra istruzione interessante è:

```
i = i + 1
```

Questa dice al computer di aggiungere 1 alla variabile **i** e assegnare il valore risultante a **i**. Quindi, se prima di questa istruzione, il valore di **i** era 1, dopo l'esecuzione sarà 2.

E per finire,

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

Questa è la parte che dice al computer che se il valore di **i** è minore di 25, l'esecuzione delle istruzioni deve andare all'etichetta **start**.

Esecuzione senza fine

Utilizzando l'istruzione **Goto** puoi fare in modo che il computer ripeta un'operazione per un numero di volte qualsiasi. Per esempio, puoi prendere il programma Pari o Dispari e modificarlo come segue, di modo che il programma venga eseguito per sempre. Puoi arrestare un programma cliccando sul pulsante di chiusura (X) nell'angolo superiore destro della finestra.

```
begin:
TextWindow.Write("Enter a number: ")
num = TextWindow.ReadNumber()
remainder = Math.Remainder(num, 2)
If (remainder = 0) Then
  TextWindow.WriteLine("The number is Even")
Else
  TextWindow.WriteLine("The number is Odd")
EndIf
Goto begin
```

Figura 18 – Pari o Dispari eseguito senza fine

Capitolo 5

Cicli

Ciclo For

Prendiamo un programma scritto nel precedente capitolo.

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

Questo programma visualizza i numeri da 1 a 24 in ordine. Questo processo di incremento di una variabile è talmente comune che i linguaggi di programmazione di norma forniscono un metodo semplice per le sua implementazione. Il programma precedente è equivalente al seguente:

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

E il risultato è:

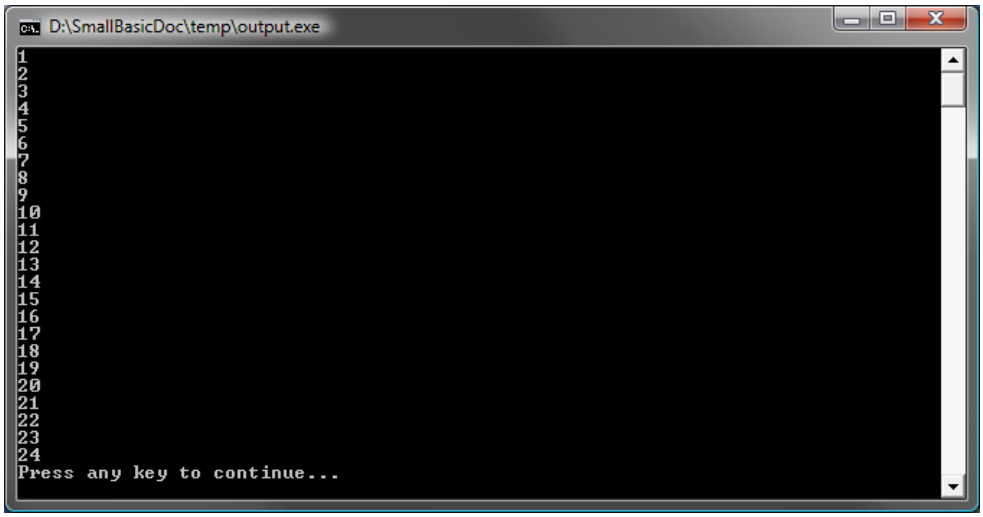

Figura 19 – Utilizzo del Ciclo For

Puoi notare che abbiamo ridotto il programma di 8 righe ad uno di 4 righe, e che questi programmi fanno esattamente la stessa cosa! Ricordi quando in precedenza abbiamo detto che di norma ci sono diversi modi di fare la stessa cosa? Questo è un esempio significativo.

For..EndFor è chiamato, in termini di programmazione, *ciclo*. Permette di prendere una variabile, darle un valore iniziale e finale, e lasciare che il computer incrementi la variabile per te. Ogni volta che il computer incrementa la variabile, esegue le istruzioni tra **For** ed **EndFor**.

Ma se volessi che la variabile fosse incrementata di 2 invece che di 1 – per visualizzare tutti i numeri dispari tra 1 e 24 - sarebbe ancora possibile utilizzare un ciclo For.

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

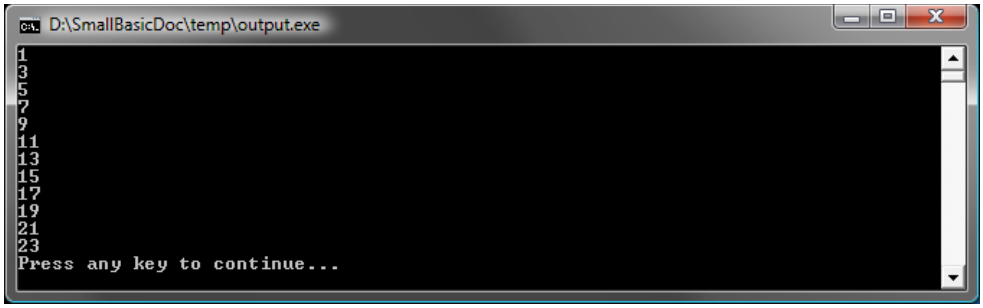

Figura 20 – Solo numeri dispari

La parte **Step 2** dell'istruzione **For** dice al computer di incrementare il valore di **i** di 2 invece che di 1. Utilizzando **Step** (letteralmente "passo") puoi specificare l'incremento desiderato. Puoi anche specificare un valore negativo per il passo e far contare il computer alla rovescia, come nel seguente esempio:

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

Figura 21 – Conto alla rovescia

Ciclo While

Il ciclo While è ancora un altro tipo di ciclo, che è utile specialmente quando il conteggio del ciclo non è noto. Considerato che il ciclo For è ripetuto un predeterminato numero di volte, il ciclo While è eseguito fin tanto che una data condizione è vera. Nell'esempio seguente, dimezziamo un numero fintanto che il risultato è maggiore di 1.

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

Figura 22 – Ciclo di Divisioni Successive

Nel programma, assegniamo il valore 100 a *number* ed eseguiamo il ciclo While fintantoché il numero è maggiore di 1. All'interno del ciclo, visualizziamo il numero e poi lo dividiamo per due. E come ci si aspetta, il risultato del programma è un elenco di numeri ottenuti dimezzandoli uno dopo l'altro.

È davvero difficile scrivere questo programma utilizzando un ciclo For, poiché non sappiamo a priori quante volte il ciclo andrà eseguito. Con un ciclo While è semplice controllare una condizione e istruire il computer a continuare o a terminare il ciclo.

È interessante notare che ogni ciclo While può essere espanso in un istruzione If…Then e una istruzione GoTo. Per esempio, il programma precedente può essere riscritto come segue, senza influenzare il risultato.

```
number = 100
startLabel:
TextWindow.WriteLine(number)
number = number / 2

If (number > 1) Then
  Goto startLabel
EndIf
```

In effetti, il computer al suo interno riscrive ogni ciclo While in istruzioni che utilizzano If…Then con una o più istruzioni Goto.

Capitolo 6

Principi di Grafica

Finora in tutti i nostri esempi, abbiamo utilizzato la Finestra Testo per spiegare i fondamenti del linguaggio Small Basic. D'altra parte, Small Basic è dotato di un potente insieme di funzionalità grafiche che inizieremo a esplorare in questo capitolo.

Introduzione alla Finestra Grafica

Proprio come abbiamo la Finestra Testo, che ci permette di lavorare con testo e numeri, Small Basic fornisce anche una Finestra Grafica (**GraphicsWindow**) che possiamo utilizzare per disegnare le cose. Iniziamo visualizzando la Finestra Grafica.

```
GraphicsWindow.Show()
```

Quando esegui il programma, noterai che invece dell'usuale Finestra Testo nera, otterrai una finestra bianca come quella mostrata di seguito. Non c'è altro da fare in questa finestra per ora. Ma questa sarà la finestra base su cui lavoreremo in questo capitolo. Puoi chiudere questa finestra cliccando sul pulsante 'X' nell'angolo in alto a destra.

Figura 23 – la Finestra Grafica vuota

Impostazioni per la Finestra Grafica

La Finestra Grafica permette la personalizzazione del proprio aspetto secondo le proprie necessità. Puoi cambiare il titolo, lo sfondo e la dimensione. Procediamo e modifichiamola un po', quel tanto che basta per acquisire familiarità con la finestra.

```
GraphicsWindow.BackgroundColor = "SteelBlue"
GraphicsWindow.Title = "My Graphics Window"
GraphicsWindow.Width = 320
GraphicsWindow.Height = 200
GraphicsWindow.Show()
```

Ecco come appare la Finestra Grafica personalizzata. Puoi cambiare il colore di sfondo con uno dei valori elencati nell'Appendice B. Gioca con queste proprietà per vedere come puoi modificare l'aspetto della finestra.

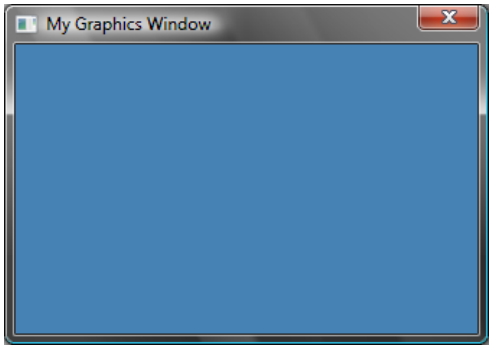

Figure 24 – Finestra Grafica personalizzata

Disegnare Linee

Una volta che abbiamo la Finestra Grafica, vi possiamo disegnare forme, testo e anche immagini. Iniziamo disegnando qualche semplice forma. Questo è un programma che disegna una coppia di linee sulla GraphicsWindow.

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

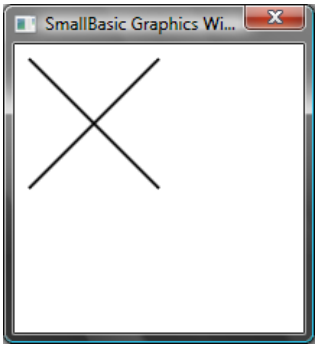

Figura 25 – Intersezione

Le prime due righe del programma impostano la dimensione della finestra e le successive due righe disegnano le linee intersecate. I primi due numeri che seguono *DrawLine* specificano le coordinate x e y iniziali e gli altri due le coordinate x e y finali. La

Invece di utilizzare i nomi per i colori, è possibile utilizzare la notazione dei colori del web (#RRGGBB). Per esempio, #FF0000 denota il rosso, #FFFF00 per il giallo, e così via.

cosa interessante con la grafica del computer è che le coordinate (0,0) corrispondono all'angolo superiore sinistro della finestra. In effetti, nello spazio delle coordinate, la finestra è considerata essere il secondo quadrante.

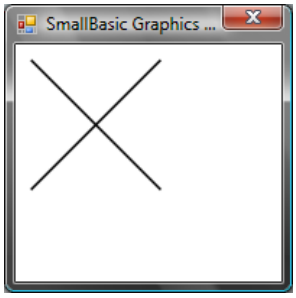

Figura 26 – La mappa delle coordinate

Se torniamo al programma delle linee, è interessante notare che Small Basic permette di modificare le proprietà delle linee, per esempio colore e spessore. Prima, modifichiamo il colore delle linee come mostrato nel programma sotto.

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

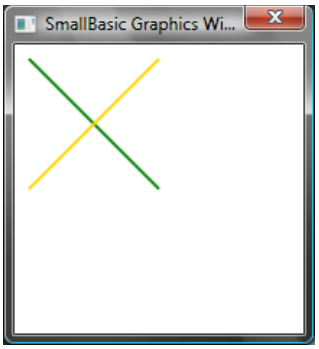

Figura 27 – Cambiare colore alla linea

Ora, modifichiamo anche la dimensione. Nel programma sotto, cambiamo lo spessore della linea a 10, invece del default che è 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)
```

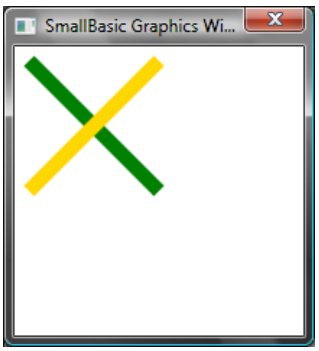

Figure 28 – Spesse linee colorate

PenWidth e *PenColor* modificano la penna con cui queste linee sono disegnate. Queste proprietà non solo influenzano le linee ma anche qualunque forma disegnata dopo la modifica. Utilizzando le istruzioni di ciclo, apprese nei precedenti capitoli, possiamo facilmente scrivere un programma che disegna linee multiple con una penna di spessore crescente.

```
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
```

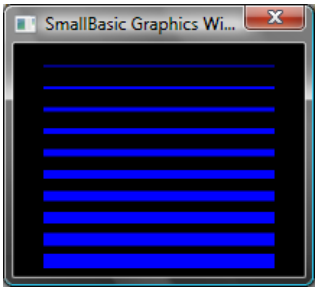

Figura 29 – Ampiezza di penna variabile

La cosa interessante di questo programma è il ciclo, in cui incrementiamo PenWidth ogni volta che il ciclo è eseguito e poi disegniamo una nuova linea sotto le precedenti.

Disegnare e riempire le forme

Quando si disegnano le forme, ci sono di norma due tipi di operazioni per ogni forma. Queste sono operazioni *Draw* e *Fill*. Le operazioni Draw disegnano il contorno della forma utilizzando una penna, e le operazioni Fill colorano l'interno della forma utilizzando un pennello. Ad esempio, nel programma seguente ci sono due rettangoli, uno che è disegnato utilizzando la penna rossa e l'altro che è riempito utilizzando il pennello 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)
```

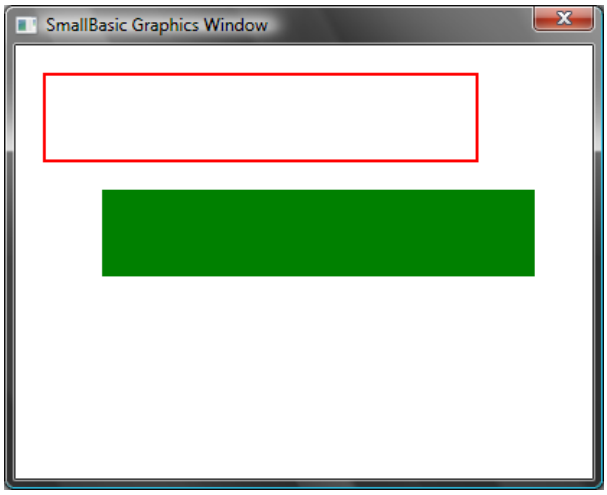

Figura 30 Disegnare e riempire

Per disegnare o riempire un rettangolo, hai bisogno di quattro numeri. I primi due numeri rappresentano le coordinate X e Y per l'angolo superiore sinistro del rettangolo. Il terzo numero specifica l'ampiezza del rettangolo, mentre il quarto specifica l'altezza. In effetti, la stessa cosa si fa per disegnare e riempire le ellissi, come mostrato nel programma seguente.

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

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

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

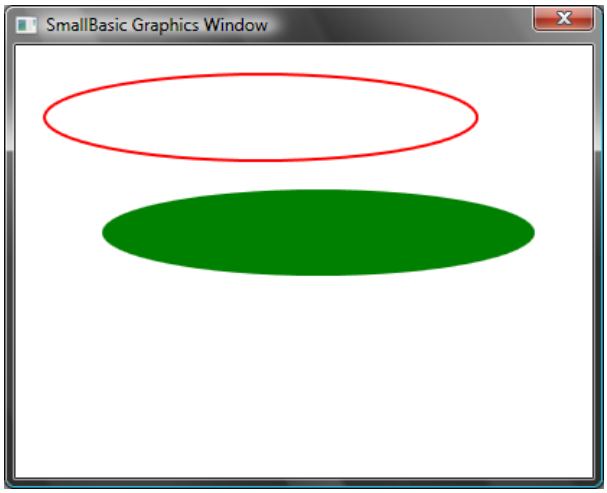

Figura 31 – Disegnare e riempire le ellissi

Le ellissi sono solo un caso più generale dei cerchi. Se vuoi disegnare un cerchio, devi specificare un'ampiezza e un'altezza uguali.

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

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

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

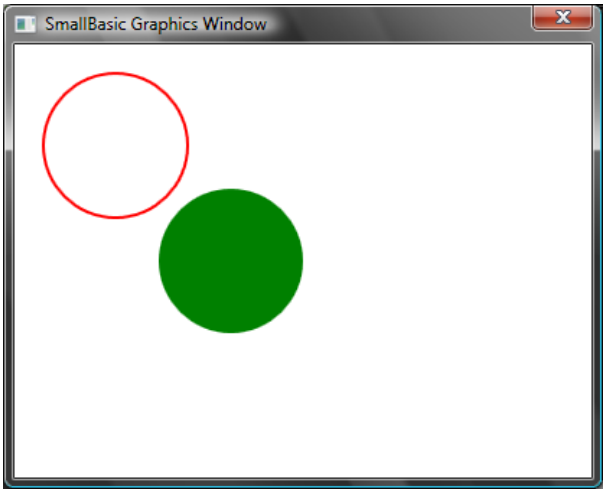

Figura 32 – Cerchi

Capitolo 7

Divertirsi con le forme

Avremo modo di divertirci in questo capitolo con quello che abbiamo appreso fin qui. Questo capitolo illustra degli esempi che mostrano come l'opportuna combinazione di tutto quello che hai imparato fin qui permette di creare dei programmi divertenti.

"Rettangolando"

Utilizzando un ciclo, disegneremo rettangoli multipli con dimensione crescente.

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

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

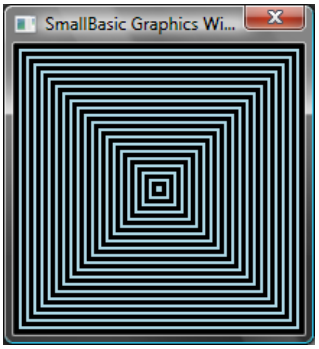

Figura 33 - Rettangolando

"Circolarizzando"

Una variante del precedente programma, disegna cerchi al posto dei quadrati.

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

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

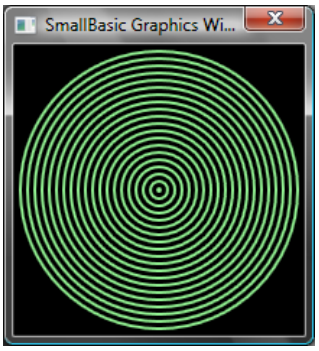

Figura 34 – Circolarizzando

Randomize

Questo programma utilizza l'operazione *GraphicsWindow.GetRandomColor* per impostare colori casuali del pennello e poi utilizza *Math.GetRandomNumber* per impostare le coordinate X e Y dei cerchi. Queste due operazioni possono essere combinate in modi interessanti per creare programmi divertenti che danno risultati differenti ogni volta che vengono eseguiti.

```
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
```

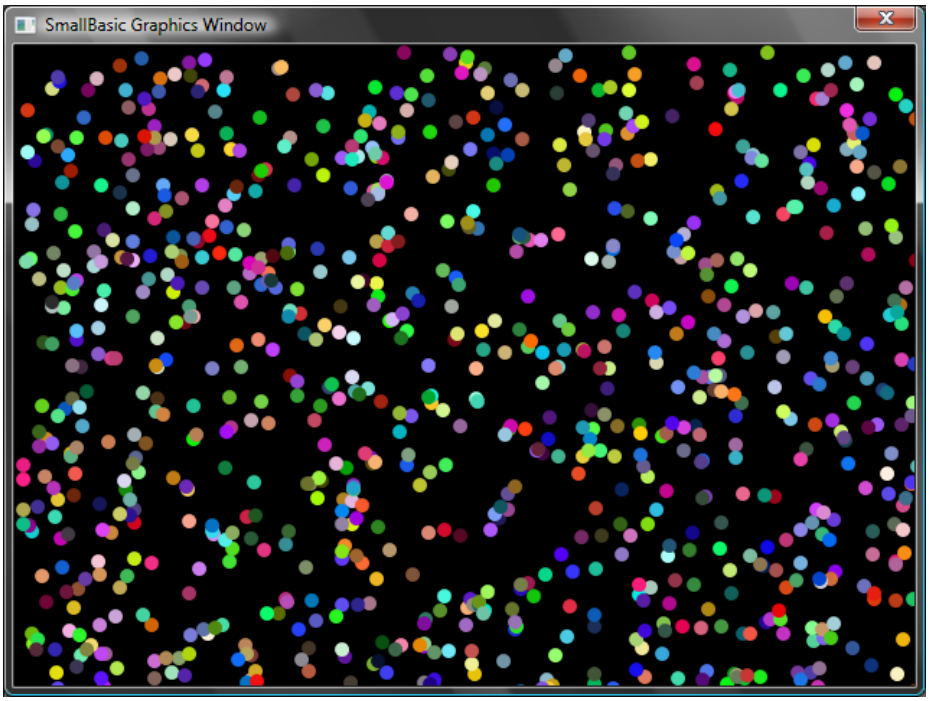

Figura 35 – Randomize

Frattali

Il seguente programma disegna un semplice frattale del triangolo utilizzando numeri casuali. Un frattale è una forma geometrica che può essere suddivisa in parti, ognuna delle quali assomiglia perfettamente alla forma da cui deriva. In questo caso, il programma disegna centinaia di triangoli ognuno dei quali assomiglia al triangolo principale. E mentre il programma è in esecuzione per un paio di secondi, puoi vedere i triangoli formarsi lentamente da semplici punti. La sua logica è qualcosa di difficile da descrivere e la lasceremo come esercizio per la tua esplorazione.

```
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
```

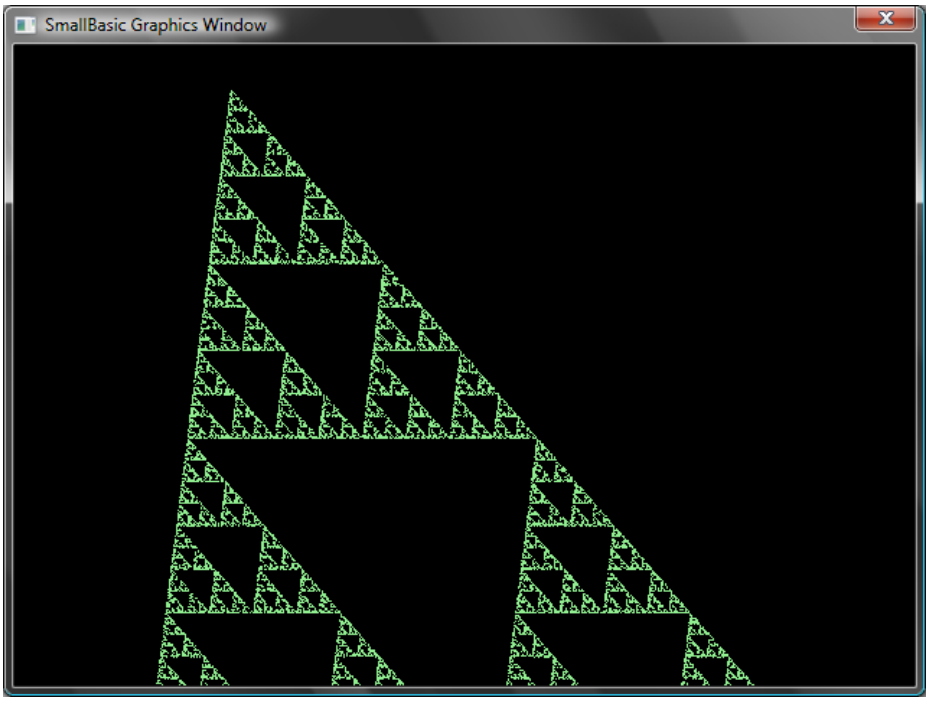

Figura 36 – Frattale del Triangolo

Se vuoi davvero vedere i punti formare lentamente il frattale, puoi introdurre una istruzione che generau un'attesa nel ciclo, utilizzando l'operazione **Program.Delay**. Questa operazione accetta un numero che specifica il tempo da attendere in millisecondi. Ecco il programma modificato, con la linea modificata in grassetto.

```
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")
  Program.Delay(2)
EndFor
```

Aumentando il tempo di attesa si rallenterà il programma. Prova diversi numeri per vedere qual è quello che ti soddisfa maggiormente.

Un’altra modifica che puoi fare a questo programma è sostituire la seguente linea di codice:

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

con

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

Questo cambiamento farà disegnare al programma il triangolo utilizzando pixel di colori casuali.

Capitolo 8

Grafica della Tartaruga

Logo

Nel 1970, esisteva un linguaggio di programmazione semplice ma potente, chiamato Logo che era utilizzato da un paio di ricercatori. Così fu fino a quando qualcuno aggiunse ciò che viene chiamato "Turtle Graphics" (Grafica della Tartaruga) al linguaggio e resa disponibile una "Tartaruga" che era invisibile sullo schermo e che rispondeva a comandi come *Move (spòstati), Forward (avanza), Turn Right (gira a destra), Turn Left (gira a sinistra),* ecc. Utilizzando la Tartaruga, le persone ebbero la possibilità di disegnare delle forme sullo schermo. Questo rese il linguaggio immediatamente accessibile e appetibile a persone di tutte le età, e fu responsabile della sua grande popolarità negli anni '80.

Small Basic è dotato di un oggetto Tartaruga con molti comandi che possono essere chiamati all'interno dei programmi Small Basic. In questo capitolo utilizzeremo la Tartaruga per disegnare sullo schermo.

La Tartaruga

Per cominciare, rendiamo visibile la Tartaruga sullo schermo. Questo si ottiene con un semplice programma di una riga.

```
Turtle.Show()
```

Eseguendo questo programma, vedrai una finestra bianca, proprio come quella vista nel precedente capitolo, con la differenza che questa ha nel centro una tartaruga. È questa la Tartaruga che eseguirà le nostre istruzioni disegnando quello che desideriamo.

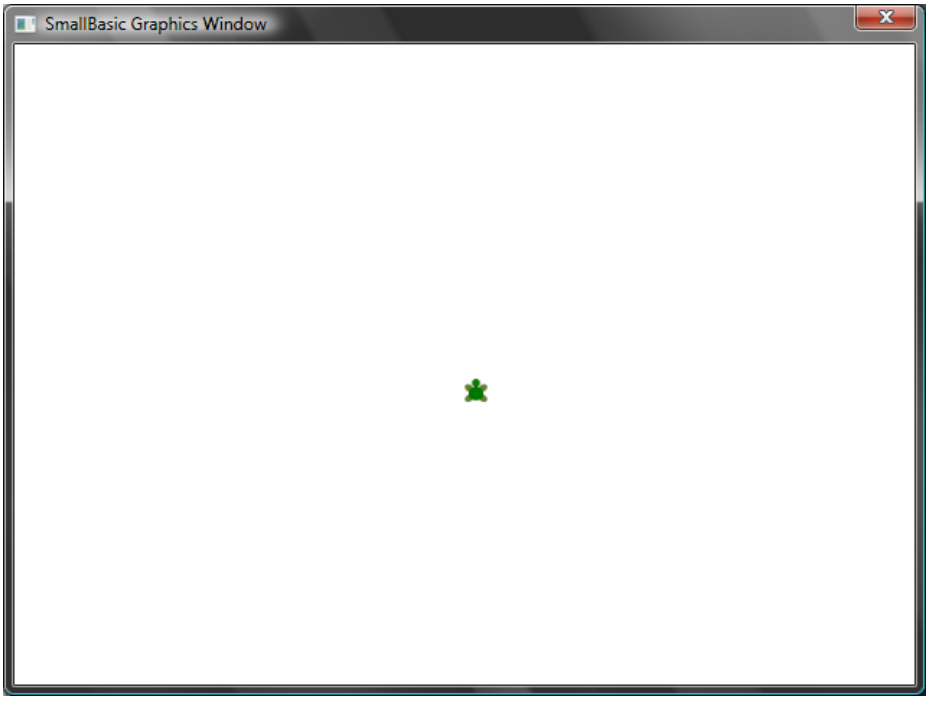

Figura 37 – La Tartaruga è visibile

Spostare e disegnare

Una delle istruzioni che la Tartaruga capisce è **Move** (spòstati). Quest'operazione accetta un numero come input. Questo numero dice alla Tartaruga di quanto spostarsi. Nell'esempio seguente, diciamo alla Tartaruga di spostarsi di 100 pixel.

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

Eseguendo questo programma, puoi vedere la Tartaruga muoversi lentamente di 100 pixel in avanti. Noterai anche che mentre si muove disegna una linea alle sue spalle. Quando la Tartaruga ha finito di muoversi, il risultato sarà qualcosa di simile alla figura seguente.

Quando utilizzi le operazioni sulla Tartaruga, non è necessario chiamare Show(). La Tartaruga sarà automaticamente visibile ogni volta che esegue una operazione.

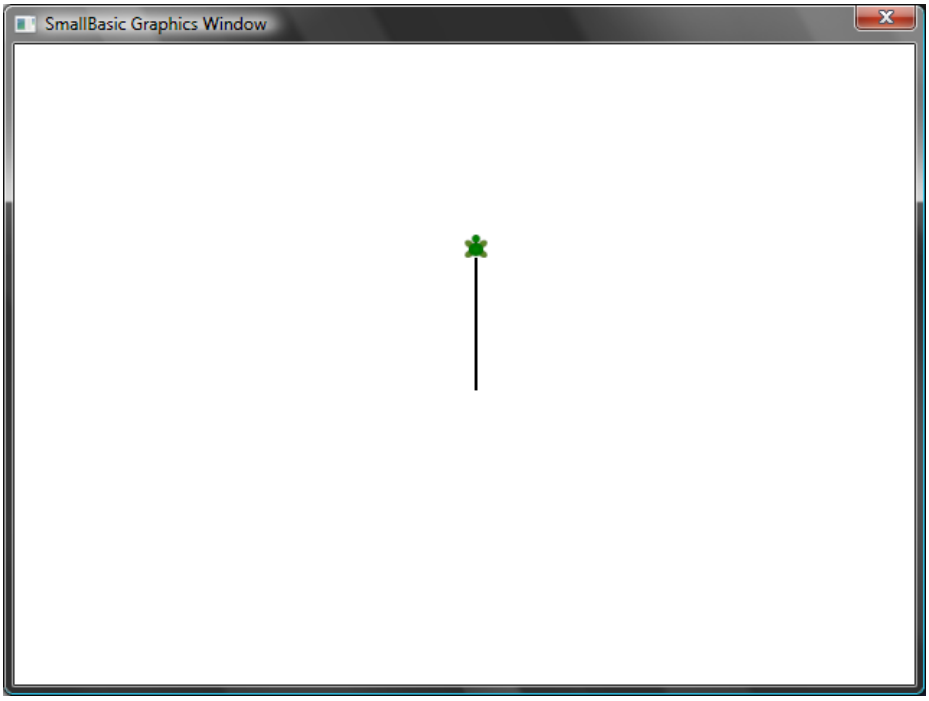

Figura 38 – Spostarsi di un centinaio di pixel

Disegnare un quadrato

Un quadrato ha quatto lati: due verticali e due orizzontali. Al fine di disegnare un quadrato abbiamo bisogno di far disegnare alla Tartaruga una linea, ruotarla a destra e disegnare un'altra linea e continuare così fino a quando non avremo realizzato tutti e quattro i lati. Tradotto in un programma, ecco il risultato.

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

Eseguendo questo programma, puoi osservare la Tartaruga disegnare un quadrato, una linea alla volta, e il risultato è quello mostrato nella figura seguente.

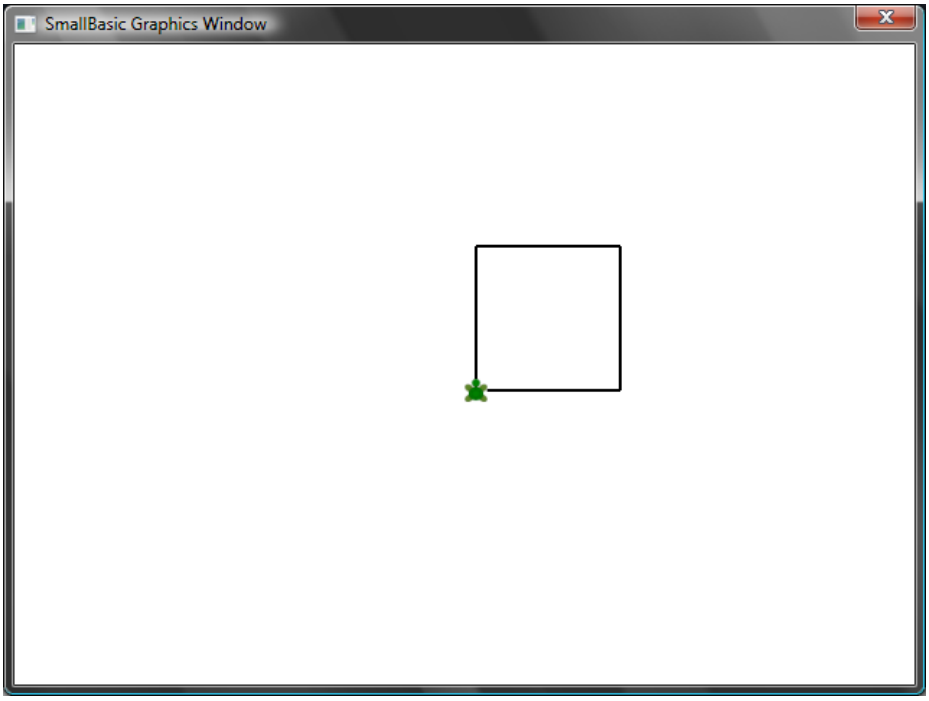

Figura 39 – La Tartaruga disegna un quadrato

È interessante notare che stiamo eseguendo sempre le stesse due operazioni per quattro volte. Dato che abbiamo già imparato comandi ripetitivi, possono essere eseguiti utilizzando i cicli, possiamo modificare il precedente programma affinché utilizzi un ciclo **For…EndFor**, arrivando a ottenere un programma più semplice.

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

Cambiare colore

La Tartaruga disegna sulla Finestra Grafica vista nel capitolo precedente. Questo significa che tutte le operazioni imparate nel precedente capitolo sono ancora valide. Ad esempio, il seguente programma disegna il quadrato con ogni lato di colore differente.

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

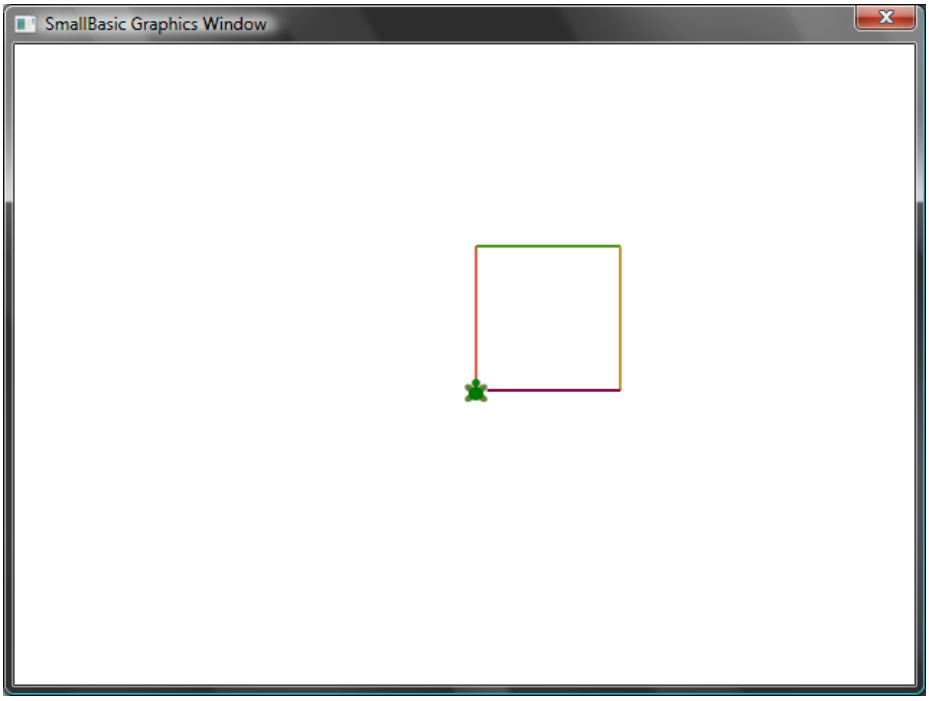

Figura 40 – Cambiare colore

Disegnare forme più complesse

La Tartaruga, in aggiunta alle operazioni **TurnRight** e **TurnLeft**, ha un'operazione **Turn**. Quest'operazione accetta un input che specifica l'angolo di rotazione. Utilizzando quest' operazione è possibile disegnare poligoni con numero arbitrario di lati. Il seguente programma disegna un esagono (un poligono di sei lati).

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

Prova questo programma per vedere se realmente disegna un esagono. Osserva che dato che l'angolo tra i lati è di 60 gradi, utilizziamo l'operazione **Turn(60)**. Per quei poligoni, i cui lati sono tutti uguali, l'angolo tra i lati può essere semplicemente ottenuto dividendo 360 per il numero dei lati. Con questa informazione, e utilizzando le variabili, possiamo scrivere un programma generico che possa disegnare poligoni con qualunque numero di lati.

```
sides = 12

length = 400 / sides
angle = 360 / sides
```

```
For i = 1 To sides
  Turtle.Move(length)
  Turtle.Turn(angle)
EndFor
```

Utilizzando questo programma, puoi disegnare qualunque poligono semplicemente modificando la variabile **sides**. Mettendo 4 ecco che otteniamo un quadrato. Mettendo un valore sufficientemente grande, diciamo 50 otterremo un risultato indistinguibile da un cerchio.

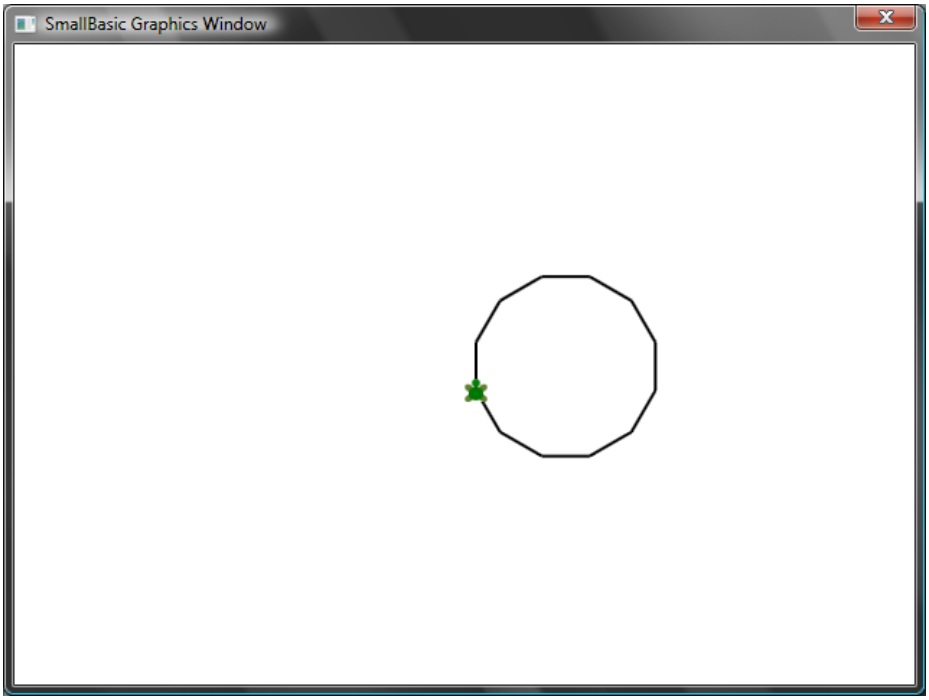

Figura 41 – Disegnare un poligono di 12 lati

Utilizzando le tecniche appena apprese, possiamo far disegnare alla Tartaruga cerchi multipli, ogni volta con un piccolo scivolamento, ottenendo un risultato interessante.

```
sides = 50
length = 400 / sides
angle = 360 / sides

Turtle.Speed = 9

For j = 1 To 20
  For i = 1 To sides
    Turtle.Move(length)
    Turtle.Turn(angle)
```

```
  EndFor
  Turtle.Turn(18)
EndFor
```

Il programma precedente ha due cicli **For…EndFor**, uno dentro l'altro. Il ciclo interno (i = 1 to sides) è simile a quello del programma del poligono ed è responsabile del disegno del cerchio. Il ciclo esterno (j = 1 to 20) è responsabile della rotazione della tartaruga di una piccola quantità per ogni cerchio che viene disegnato e dice alla tartaruga di disegnare 20 cerchi. Messi insieme, il programma produce un motivo interessante, come quello seguente.

Nel precedente programma abbiamo fatto andare più velocemente la Tartaruga impostando Speed a 9. È possibile impostare questa proprietà a qualunque valore tra 1 e 10 per far andare la Tartaruga alla velocità desiderata.

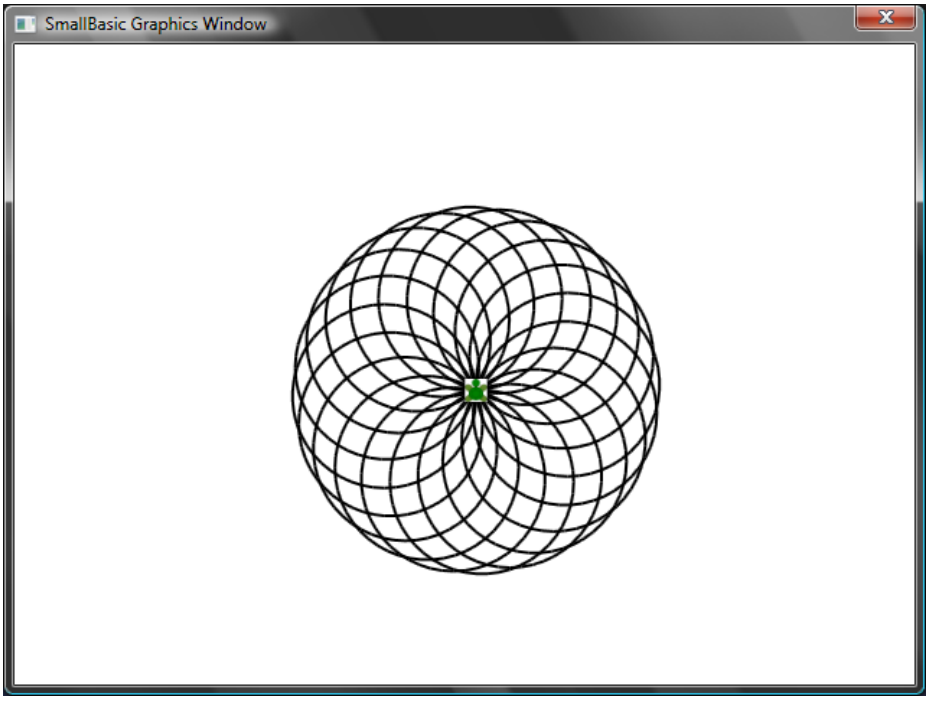

Figura 42 – Di cerchio in cerchio

Spostamenti

È possibile fare in modo che la Tartaruga non disegni chiamando l'operazione **PenUp**. Ciò permette di spostare la Tartaruga ovunque sullo schermo senza disegnare alcuna linea. Chiamando **PenDown** la Tartaruga tornerà a disegnare ancora. Questo può essere utilizzato per ottenere degli effetti interessanti, come linee tratteggiate. Ecco un programma che utilizza quest'operazione per disegnare un poligono con linee tratteggiate.

```
sides = 6

length = 400 / sides
angle = 360 / sides

For i = 1 To sides
  For j = 1 To 6
    Turtle.Move(length / 12)
    Turtle.PenUp()
    Turtle.Move(length / 12)
    Turtle.PenDown()
  EndFor
  Turtle.Turn(angle)
EndFor
```

Ancora una volta, questo programma ha due cicli. Il ciclo interno disegna una singola linea tratteggiata, mentre il ciclo esterno indica quante linee disegnare. Nel nostro esempio, utilizziamo 6 per la variabile **sides** e di conseguenza otteniamo un esagono con linee tratteggiate, come sotto.

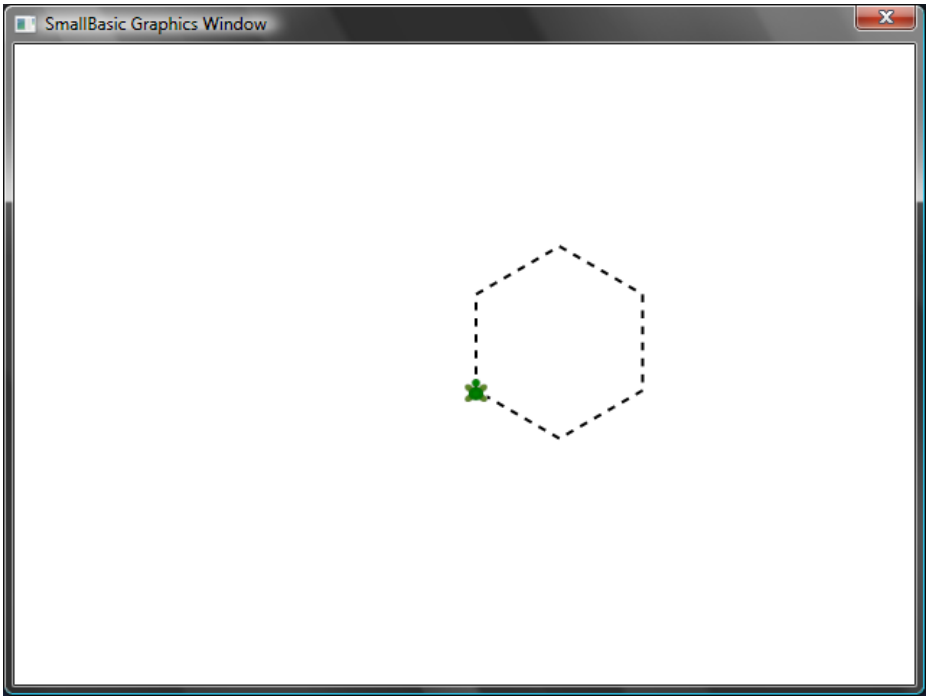

Figura 43 – Utilizzo di PenUp e PenDown

Capitolo 9

Subroutine

Molto spesso mentre scriviamo i programmi ricorrono dei casi in cui dobbiamo eseguire lo stesso insieme di passi ripetutamente. In questi casi, non ha probabilmente senso riscrivere le stesse istruzioni più volte. In queste occasioni sono utili le *Subroutine*.

Una *Subroutine* è una porzione di codice all'interno di un più ampio programma che fa qualcosa di specifico, e che può essere chiamata in qualunque punto del programma. Le subroutine sono identificate da un nome che segue la parola chiave **Sub** e sono terminare dalla parola chiave **EndSub**. Per esempio, la seguente porzione di codice rappresenta una subroutine il cui nome è *PrintTime*, e fa il lavoro di stampare l'ora corrente sulla Finestra Testo.

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

Quello che segue è un programma che include la subroutine e la chiama in vari punti.

```
PrintTime()
TextWindow.Write("Enter your name: ")
name = TextWindow.Read()
TextWindow.Write(name + ", the time now is: ")
PrintTime()

Sub PrintTime
  TextWindow.WriteLine(Clock.Time)
```

```
EndSub
```

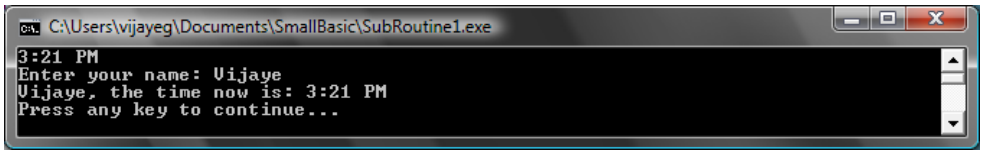

Figura 44 – Chiamata a una semplice Subroutine

Puoi eseguire una subroutine chiamando *SubroutineName(). Come al solito, i caratteri di punteggiatura () sono indispensabili per indicare al computer che vuoi eseguire una subroutine.*

Vantaggi nell'uso delle subroutine

Come abbiamo appena visto, le subroutine aiutano a ridurre la quantità di codice che occorre digitare. Una volta scritta la subroutine *PrintTime*, è possibile chiamarla in qualunque punto del programma e l'ora corrente verrà stampata.

Ricorda, puoi solo chiamare una subroutine Small Basic dall'interno dello stesso programma. Non puoi chiamare una subroutine da altri programmi.

In più, le subroutine possono aiutare a decomporre problemi complessi in pezzi più semplici. Diciamo che hai un'equazione complessa da risolvere, puoi scrivere parecchie subroutine che risolvono pezzi più piccoli dell'equazione complessa. Poi puoi mettere insieme i risultati per ottenere la soluzione dell'equazione complessa originale.

Le Subroutine possono anche aiutare a migliorare la leggibilità di un programma. In altre parole, se hai dei nomi chiari per le tue subroutine che eseguono porzioni del tuo programma, il programma nel suo complesso diventa più semplice da leggere e comprendere. Questo è molto importante se vuoi comprendere il programma di qualcun altro o se vuoi che il tuo programma sia compreso da altri. Qualche volta, è di aiuto anche quando vuoi rileggere i tuoi programmi più vecchi.

Utilizzo di Variabili

Puoi accede e utilizzare qualunque Variabile presente in un programma all'interno di una subroutine. Come esempio, il seguente programma accetta due numeri e stampa il più grande dei due. Si noti che la Variabile *max* è utilizzata sia dentro che fuori della subroutine.

```
TextWindow.Write("Enter first number: ")
num1 = TextWindow.ReadNumber()
TextWindow.Write("Enter second number: ")
num2 = TextWindow.ReadNumber()
```

```
FindMax()
TextWindow.WriteLine("Maximum number is: " + max)

Sub FindMax
  If (num1 > num2) Then
    max = num1
  Else
    max = num2
  EndIf
EndSub
```

E il risultato di questo programma è il seguente.

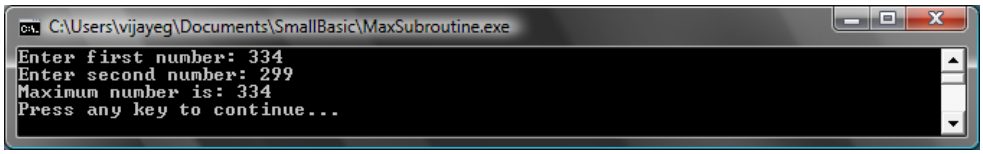

Figura 45 – Massimo di due numeri con la subroutine

Vediamo un altro esempio che illustra l'utilizzo delle subroutine. Questa volta useremo un programma grafico che calcola vari punti che memorizzerà nelle variabili x e y. Poi chiama una subroutine **DrawCircleUsingCenter** che disegna un cerchio utilizzando x e y come 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
  y = Math.Cos(i) * 100 + 200

  DrawCircleUsingCenter()
EndFor

Sub DrawCircleUsingCenter
  startX = x - 40
  startY = y - 40

  GraphicsWindow.DrawEllipse(startX, startY, 120, 120)
EndSub
```

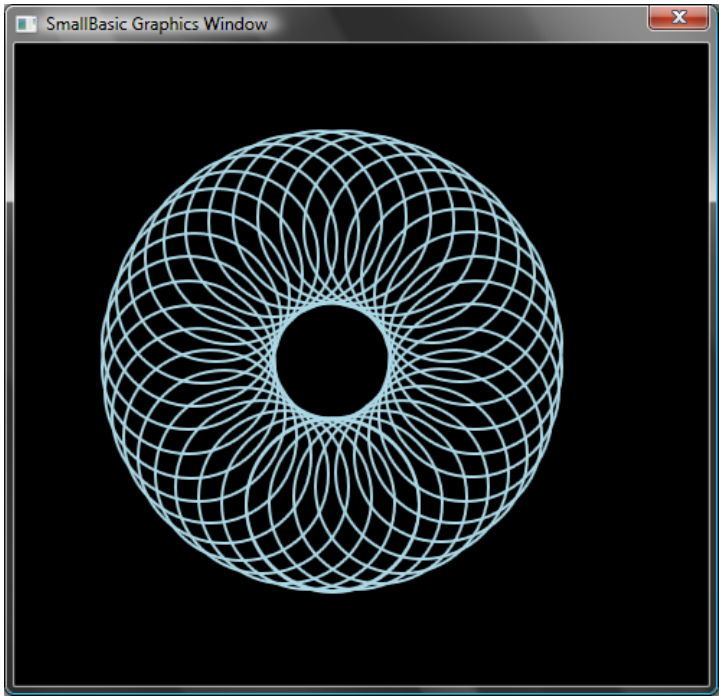

Figura 46 – Esempio grafico di subroutine

Chiamata di subroutine all'interno dei cicli

A volte le subroutine vengono chiamate all'interno dei cicli, in queste occasioni queste eseguono lo stesso insieme di istruzioni ma con valori differenti in una o più variabili. Ad esempio, diciamo di aver una subroutine chiamata *PrimeCheck* che determina se un numero è primo o no. Puoi scrivere un programma che permette all'utente di inserire un valore e poi dice se questo è un numero primo o meno, utilizzando la subroutine. Quello che segue è il programma in questione.

```
TextWindow.Write("Enter a number: ")
i = TextWindow.ReadNumber()
isPrime = "True"
PrimeCheck()
If (isPrime = "True") Then
  TextWindow.WriteLine(i + " is a prime number")
Else
  TextWindow.WriteLine(i + " is not a prime number")
EndIf

Sub PrimeCheck
  For j = 2 To Math.SquareRoot(i)
    If (Math.Remainder(i, j) = 0) Then
      isPrime = "False"
      Goto EndLoop
```

```
    EndIf
  Endfor
EndLoop:
EndSub
```

La subroutine *PrimeCheck* prende il valore di *i* e cerca di dividerlo per numeri più piccoli. Se un numero divide *i* e non da resto, allora *i* non è un numero primo. A questo punto la subroutine imposta il valore di *isPrime* a "False" (falso) ed esce. Se il numero fosse indivisibile per un numero più piccolo allora il valore di *isPrime* rimane impostato a "True" (vero).

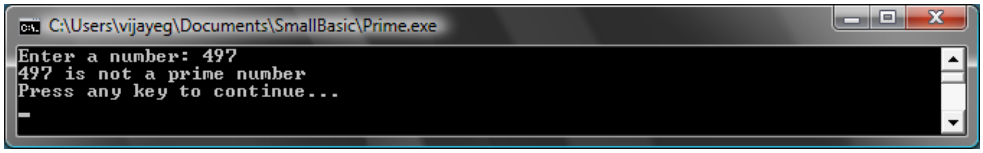

Figura 47 – Controllo numeri primi

Ora che hai una subroutine che può fare il controllo dei numeri primi, potresti volerla utilizzare per elencare tutti i numeri primi fino a 100, ad esempio. È davvero semplice modificare il precedente programma e fare la chiamata a *PrimeCheck* dall'interno di un ciclo. In questo modo la subroutine ha utilizza un valore differente ad ogni esecuzione del ciclo. Vediamo come fare nell'esempio seguente.

```
For i = 3 To 100
  isPrime = "True"
  PrimeCheck()
  If (isPrime = "True") Then
    TextWindow.WriteLine(i)
  EndIf
EndFor

Sub PrimeCheck
  For j = 2 To Math.SquareRoot(i)
    If (Math.Remainder(i, j) = 0) Then
      isPrime = "False"
      Goto EndLoop
    EndIf
  Endfor
EndLoop:
EndSub
```

Nel programma qui sopra, il valore di *i* è aggiornato ogni volta che il ciclo viene eseguito. All'interno del ciclo, una chiamata alla subroutine *PrimeCheck* prende il valore di *i* e calcola se è un numero primo o meno. Il risultato è memorizzato nella variabile *isPrime* a cui si accede nel ciclo fuori della subroutine. Il

valore di *i* viene visualizzato se risulta essere un numero primo. E mentre il ciclo inizia da 3 e va fino a 100, noi otteniamo una lista di tutti i numeri primi che sono compresi tra 3 e 100. Segue il risultato del programma.

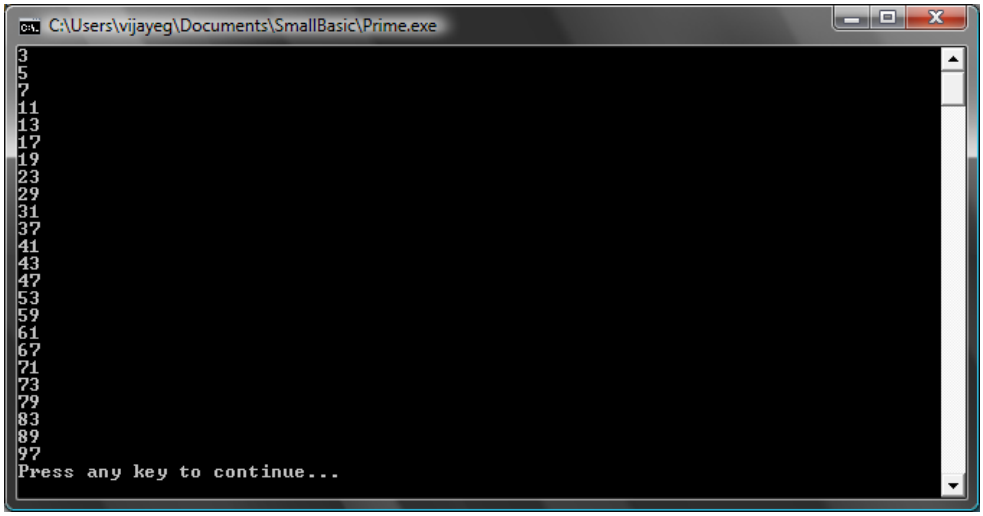

Figura 48 – Numeri primi

Capitolo 10

Vettori

A questo punto hai una buona conoscenza delle variabili e del loro utilizzo – dopo tutto sei arrivato fin qui e c'è ancora da divertirsi!

Rivisitiamo, per un momento, il primo programma che faceva uso delle variabili:

```
TextWindow.Write("Enter your Name: ")
name = TextWindow.Read()
TextWindow.WriteLine("Hello " + name)
```

In questo programma, abbiamo ricevuto e memorizzato il nome dell'utente in una variabile chiamata **name**. Poi abbiamo detto "Hello" all'utente. Adesso, diciamo che ci sia più di un utente – ad esempio 5. Come possiamo memorizzare tutti i loro nomi? Un modo per farlo è questo:

```
TextWindow.Write("User1, enter name: ")
name1 = TextWindow.Read()
TextWindow.Write("User2, enter name: ")
name2 = TextWindow.Read()
TextWindow.Write("User3, enter name: ")
name3 = TextWindow.Read()
TextWindow.Write("User4, enter name: ")
name4 = TextWindow.Read()
TextWindow.Write("User5, enter name: ")
name5 = TextWindow.Read()
```

```
TextWindow.Write("Hello ")
TextWindow.Write(name1 + ", ")
TextWindow.Write(name2 + ", ")
TextWindow.Write(name3 + ", ")
TextWindow.Write(name4 + ", ")
TextWindow.WriteLine(name5)
```

Quando esegui questo programma ottieni il seguente risultato:

Figura 49 – Senza l'Uso dei Vettori

Chiaramente ci deve esserer un modo migliore per scrivere un programma così semplice, giusto? Specialmente dato che il computer è un ottimo strumento per lo svolgimento di compiti ripetitivi, perché dovremmo perdere tempo con la scrittura dello stesso codice per ogni nuovo utente? Il trucco sta nel memorizzare e recuperare più di un nome utente utilizzando la stessa variabile. Se potessimo farlo, potremo poi utilizzare un ciclo **For** appreso nei precedenti capitoli. È in queste occasioni che i vettori ci vengono in aiuto.

Cos'è un Vettore?

Un vettore è uno speciale tipo di variabile che può contenere più di un valore per volta. Fondamentalmente, significa avere al posto di **name1, name2, name3, name4, name5**, una sola variabile di nome **name** in cui memorizzare tutti e cinque i nomi utente. Il modo per memorizzare valori multipli è di utilizzare una cosa chiamata *"Indice"*. Per esempio, **name[1], name[2], name[3], name[4] e name[5]** possono tutti memorizzare un valore ciascuno. I numeri 1, 2, 3, 4 e 5 sono chiamati *indici* del vettore.

Sebbene **name[1]**, **name[2]**, **name[3]**, **name[4]** e **name[5]** possano sembrare variabili diverse, esse sono in realtà una variabile sola. E qual è il vantaggio di tutto ciò, potresti chiedere. La parte migliore del memorizzare i valori in vettori è che puoi specificare l'indice utilizzando un'altra variabile – il che permette di accedere con facilità ai vettori all'interno di cicli.

Adesso vediamo come mettere in pratica questa nuova conoscenza per riscrivere il nostro precedente programma con i vettori.

```
For i = 1 To 5
```

```
  TextWindow.Write("User" + i + ", enter name: ")
  name[i] = TextWindow.Read()
EndFor

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

Molto più semplice da leggere, non è vero? Nota le due righe in grassetto. La prima memorizza un valore nel vettore e la seconda lo legge dal vettore. Il valore memorizzato in **name[1]** non interferisce con quello memorizzato in **nome[2].** Quindi per molti aspetti puoi trattare **name[1]** e **name[2]** come due variabili differenti pur avendo lo stesso nome.

Figura 50 – Utilizzo dei vettori

Il programma precedente da esattamente lo stesso risultato di quello senza i vettori, eccetto per la virgola alla fine di *Mantis*. Possiamo aggiustarlo riscrivendo il ciclo di stampa come segue:

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

Indicizzare un Vettore

Nel nostro precedente programma abbiamo utilizzato i numeri come indici per memorizzare e recuperare i valori dal vettore. Scopriamo che gli indici non sono ristretti solo ai numeri e in pratica è anche molto utile utilizzare indici testuali. Ad esempio, nel seguente programma, richiediamo e memorizziamo varie informazioni su un utente e poi visualizziamo le informazioni che l'utente chiede.

```
TextWindow.Write("Enter name: ")
user["name"] = TextWindow.Read()
TextWindow.Write("Enter age: ")
user["age"] = TextWindow.Read()
TextWindow.Write("Enter city: ")
user["city"] = TextWindow.Read()
TextWindow.Write("Enter zip: ")
user["zip"] = TextWindow.Read()

TextWindow.Write("What info do you want? ")
index = TextWindow.Read()
TextWindow.WriteLine(index + " = " + user[index])
```

Figura 51 – Utilizzo di indici non numerici

Più di una dimensione

Supponiamo di voler memorizzare il nome ed il numero di telefono di tutti i tuoi amici e poi di poter cercare il loro numero ogni volta che ne hai bisogno – tipo una rubrica. Come dovremmo scrivere tale programma?

In questo caso ci sono due insiemi di indici (anche conosciuti come dimensioni del vettore) coinvolti.

Supponiamo di identificare ogni amico con il proprio soprannome. Questo sarà il primo indice del vettore. Utilizziamo il primo indice per ottenere la variabile coi dati del nostro amico, e il

Gli indici dei vettori non sono case sensitive. Proprio come le variabili regolari, gli indici dei vettori non devono corrispondere esattamente per quel che riguarda il maiuscolo/minuscolo.

secondo indice, **name** e **phone number** ci aiuteranno ad ottenere nome e numero di telefono dell'amico. Il modo in cui memorizziamo questi dati potrebbe essere questo:

```
friends["Rob"]["Name"] = "Robert"
friends["Rob"]["Phone"] = "555-6789"

friends["VJ"]["Name"] = "Vijaye"
friends["VJ"]["Phone"] = "555-4567"

friends["Ash"]["Name"] = "Ashley"
friends["Ash"]["Phone"] = "555-2345"
```

Dato che abbiamo due indici sullo stesso vettore, **friends**, questo vettore è chiamato *vettore a due dimensioni.*

Una volta impostato questo programma, possiamo accettare in ingresso il soprannome di un amico e visualizzare le sue informazioni. Ecco il programma completo che fa ciò:

```
friends["Rob"]["Name"] = "Robert"
friends["Rob"]["Phone"] = "555-6789"

friends["VJ"]["Name"] = "Vijaye"
friends["VJ"]["Phone"] = "555-4567"

friends["Ash"]["Name"] = "Ashley"
friends["Ash"]["Phone"] = "555-2345"

TextWindow.Write("Enter the nickname: ")
nickname = TextWindow.Read()

TextWindow.WriteLine("Name: " + friends[nickname]["Name"])
TextWindow.WriteLine("Phone: " + friends[nickname]["Phone"])
```

Figura 52 – Una semplice rubrica

Utilizzare i Vettori per rappresentare le griglie

Un utilizzo molto comune dei vettori a più dimensioni è per rappresentare griglie e tabelle. Le griglie hanno righe e colonne, che possono opportunamente riempire un vettore a due dimensioni. Un semplice programma che dispone dei riquadri in una griglia è il seguente:

```
rows = 8
columns = 8
size = 40

For r = 1 To rows
  For c = 1 To columns
    GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
    boxes[r][c] = Shapes.AddRectangle(size, size)
    Shapes.Move(boxes[r][c], c * size, r * size)
  EndFor
EndFor
```

Questo programma aggiunge dei rettangoli e li posiziona a formare una griglia 8x8. Oltre alla disposizione dei riquadri, li memorizza anche in un vettore. In questo modo è semplice tenere traccia dei riquadri e utilizzarli ancora quando necessario.

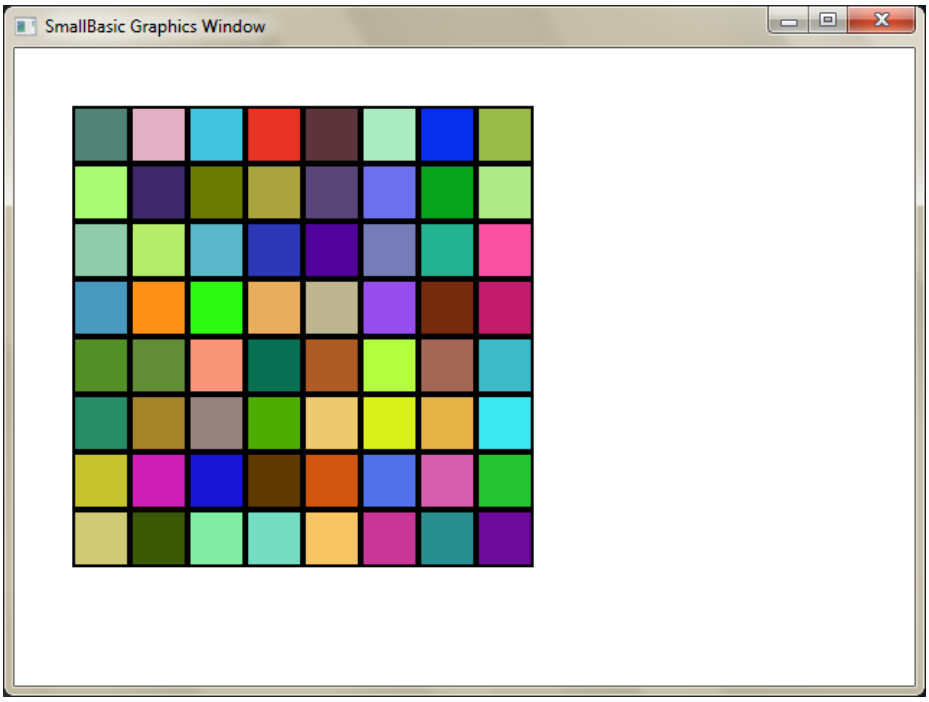

Figura 53 – Disposizione di riquadri in una griglia

Per esempio, aggiungendo il seguente codice alla fine del precedente programma animeremo i riquadri verso l'angolo superiore sinistro.

```
For r = 1 To rows
  For c = 1 To columns
    Shapes.Animate(boxes[r][c], 0, 0, 1000)
    Program.Delay(300)
  EndFor
EndFor
```

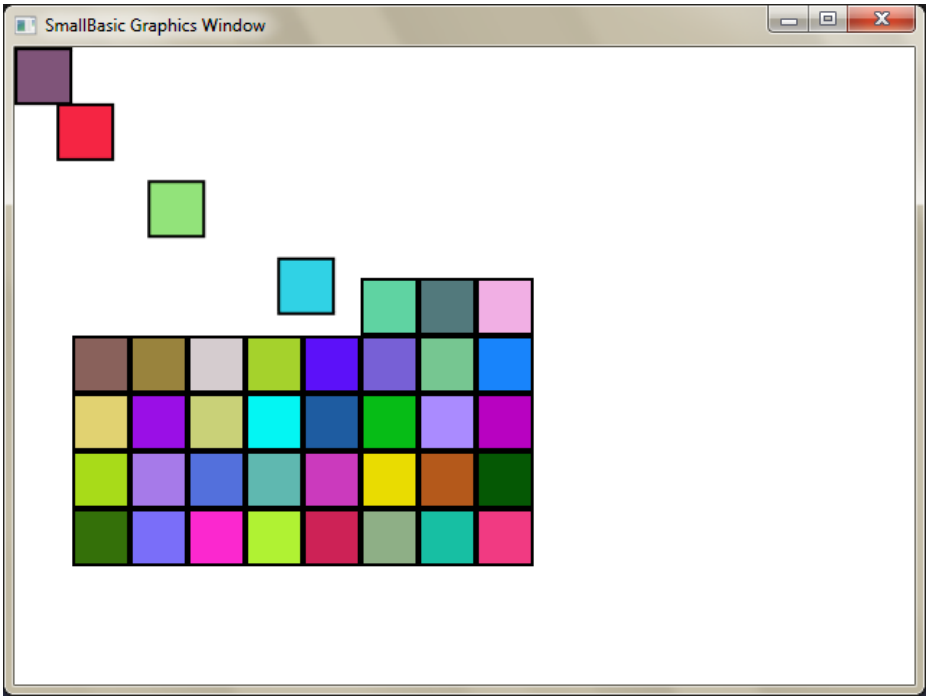

Figura 54 – Tenere traccia dei riquadri nella griglia

Capitolo 11

Eventi e interattività

Nei primi due capitoli, abbiamo introdotto gli oggetti che hanno *Proprietà* e *Operazioni*. In aggiunta alle proprietà e operazioni, qualche oggetto ha ciò che chiameremo ***Eventi***. Gli eventi sono come segnali che sono sollevati, per esempio, in risposta alle azioni dell'utente, come muovere il mouse o cliccarlo. In un certo senso gli eventi sono l'opposto delle operazioni. Nel caso delle operazioni, tu come programmatore le richiami per far fare qualcosa al computer; d'altra parte, nel caso degli eventi, il computer ti fa sapere quando avviene qualcosa di interessante.

Sono utili gli eventi?

Gli eventi sono essenziali per introdurre interattività in un programma. Se vuoi permettere all'utente di interagire con il tuo programma, gli eventi sono ciò di cui hai bisogno. Diciamo tu stia scrivendo il programma del Tris. Vorrai permettere all'utente di scegliere il proprio simbolo, giusto? È qui che gli eventi entrano in gioco – ricevi l'input dell'utente nel tuo programma utilizzando gli eventi. Se ti sembra difficile da comprendere, non preoccuparti, daremo uno sguardo ad un semplice esempio che ti aiuterà a capire cosa sono gli eventi e come possono essere utilizzati.

Sotto trovi un programma molto semplice che ha appena un'istruzione e una subroutine. La subroutine utilizza l'operazione *ShowMessage* dell'oggetto GraphicsWindow per visualizzare un messaggio all'utente.

```
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  GraphicsWindow.ShowMessage("You Clicked.", "Hello")
EndSub
```

La parte interessante del programma qui sopra, è la linea in cui assegniamo il nome della subroutine all'evento **MouseDown** dell'oggetto Window. Noterai che MouseDown assomiglia molto a una proprietà – eccetto che invece di assegnare un valore, le stiamo assegnando la subroutine *OnMouseDown*. Questo è ciò che rende speciali gli eventi – quando si verifica l'evento, la subroutine viene chiamata automaticamente. In questo caso, la subroutine *OnMouseDown* vien chiamata ogni volta che l'utente clicca il mouse sulla GraphicsWindow. Procedi eseguendo il programma e provandolo. Ogni volta che clicchi sulla GraphicsWindow con il mouse, vedrai un messaggio proprio come quello mostrato nella figura seguente.

Figura 55 – Risposta agli eventi

Questo tipo di gestione degli eventi è molto potente e permette la creazione di programmi creativi e interessanti. I programmi scritti in questo modo sono spesso chiamati *programmi event-driven*.

Puoi modificare la subroutine *OnMouseDown* per fare altre cose invece che visualizzare un messaggio. Ad esempio, come nel programma seguente, puoi disegnare dei grossi punti blu nel punto in cui l'utente fa clic.

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

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

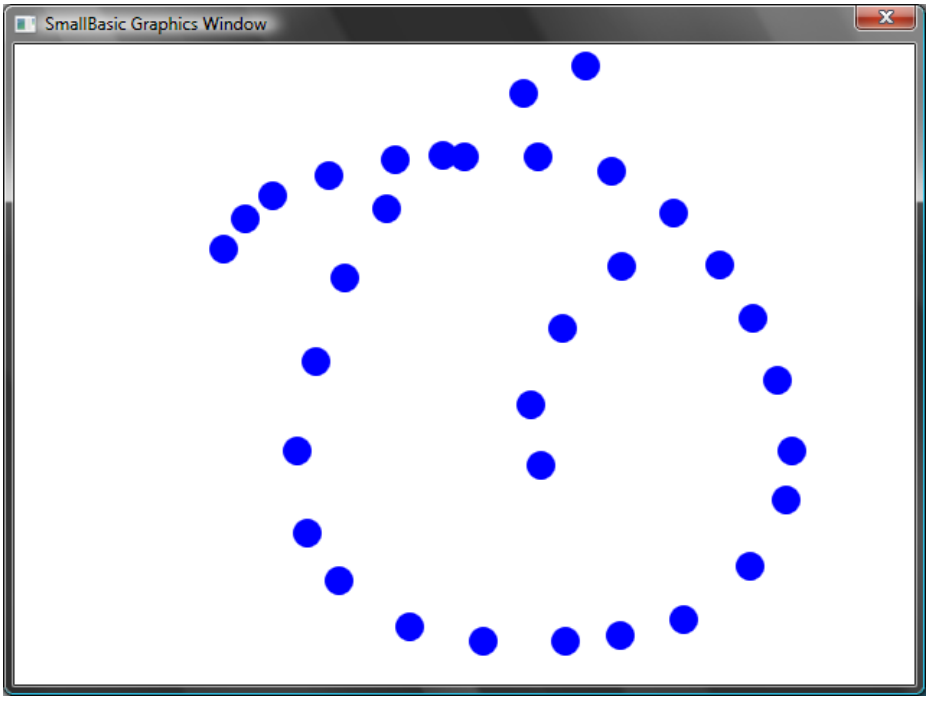

Figura 56 – Gestione dell'evento MouseDown

Si noti che nel programma precedente, abbiamo utilizzato *MouseX e* Mouse Y per ottenere le coordinate del mouse. Abbiamo poi utilizzato queste ultime per disegnare un cerchio utilizzandole come centro.

Gestire eventi multipli

Non ci sono davvero limiti alla quantità di eventi che si possono gestire. Puoi anche avere una subroutine per gestire eventi multipli. Puoi comunque gestire un evento una sola volta. Se provi ad assegnare due subroutine allo stesso evento, la seconda sostituisce la prima.

Per chiarire il concetto, prendiamo il precedente esempio e aggiungiamo una subroutine che gestisce la pressione dei tasti. In più, questa nuova subroutine cambia il colore del pennello di modo che quando clicchi, otterrai un punto di colore diverso.

```
GraphicsWindow.BrushColor = "Blue"
GraphicsWindow.MouseDown = OnMouseDown
GraphicsWindow.KeyDown = OnKeyDown

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

Sub OnMouseDown
  x = GraphicsWindow.MouseX - 10
```

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

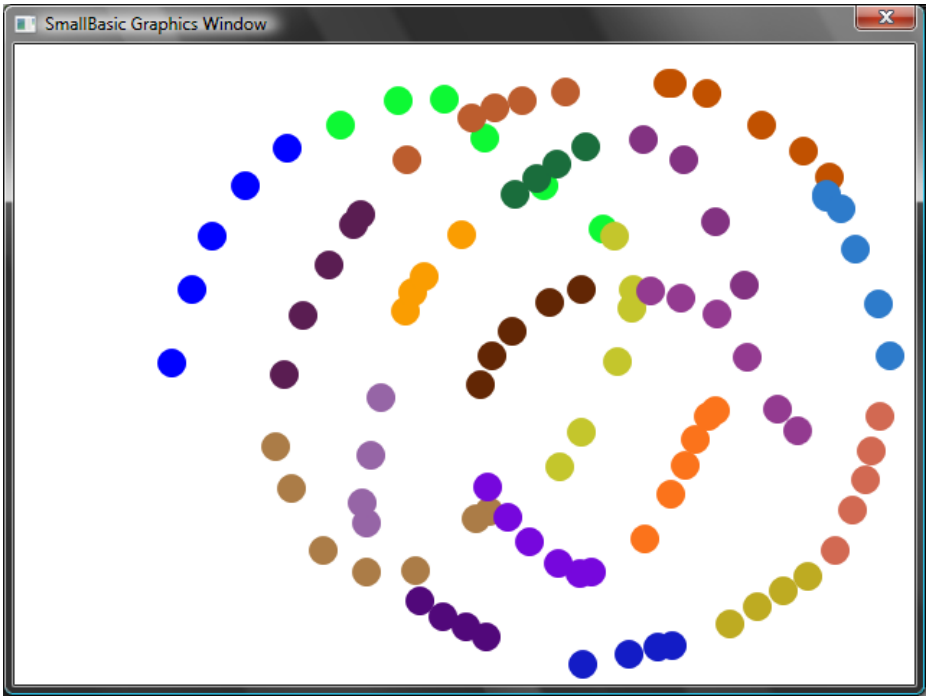

Figura 57 – Gestire eventi multipli

Se esegui il programma e fai clic sulla finestra, otterrai un punto blu. Se poi premi un qualunque tasto e fai ancora clic, otterrai un punto di colore differente. Quello che accade quando premi un tasto è che la subroutine *OnKeyDown* viene eseguita cambiando il colore del pennello con uno casuale. Dopo di che, quando fai clic col mouse, viene disegnato un cerchio utilizzando il nuovo colore – generando punti di colore casuale.

Un programma per il disegno

Dotato di eventi e subroutine, possiamo ora scrivere un programma che permette all'utente di disegnare nella finestra. È sorprendentemente semplice scrivere un programma del genere, potendo spezzettare il problema in altri più semplici. Per prima cosa, scriviamo un programma che permette all'utente di muovere il mouse ovunque sulla GraphicsWindow, lasciando una traccia dello spostamento.

```
GraphicsWindow.MouseMove = OnMouseMove

Sub OnMouseMove
  x = GraphicsWindow.MouseX
  y = GraphicsWindow.MouseY
```

```
  GraphicsWindow.DrawLine(prevX, prevY, x, y)
  prevX = x
  prevY = y
EndSub
```

Quando esegui questo programma, comunque la prima linea inizia sempre dal bordo superiore sinistro della finestra (0,0). Possiamo correggere questo problema gestendo l'evento *MouseDown* e catturando i valori *prevX* e *prevY* quando si verifica l'evento.

In oltre, abbiamo bisogno della traccia quando l'utente ha il pulsante del mouse premuto. Nelle altre occasioni, non dovemmo disegnare le linee. Al fine di ottenere questo comportamento, utilizzeremo la proprietà *IsLeftButtonDown* dell'oggetto **Mouse**. Questa proprietà dice se il pulsante sinistro è premuto o no. Se questo valore è vero, allora disegneremo la linea, altrimenti la saltiamo.

```
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
```

Appendice A

Esempi divertenti

I frattali della Tartaruga

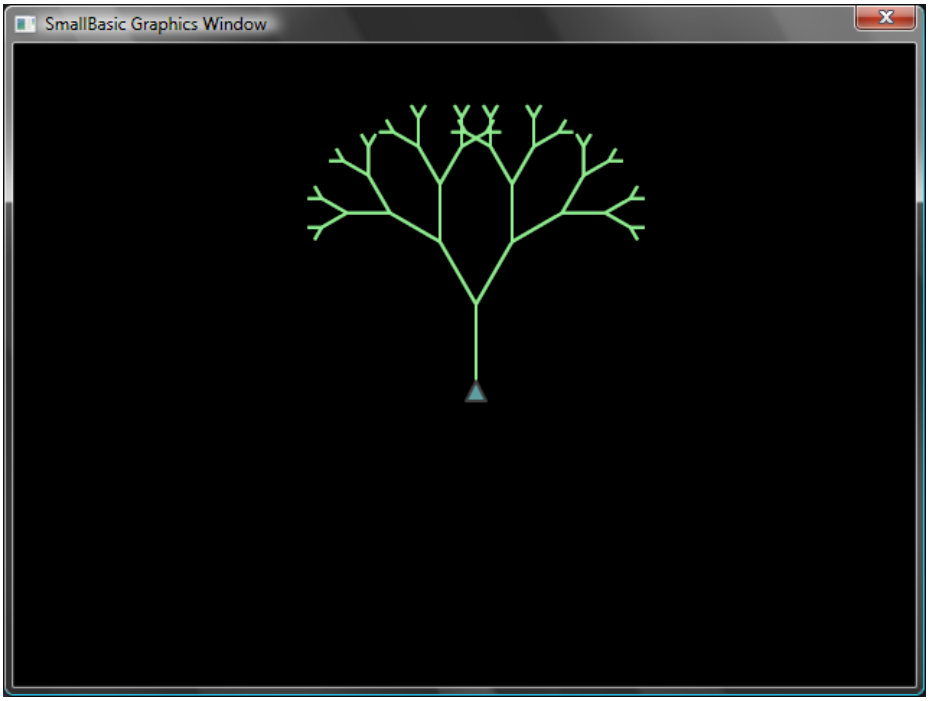

Figura 58 – La Tartaruga disegna il frattale di un albero

```
angle = 30
delta = 10
distance = 60
Turtle.Speed = 9
```

```
GraphicsWindow.BackgroundColor = "Black"
GraphicsWindow.PenColor = "LightGreen"
DrawTree()

Sub DrawTree
  If (distance > 0) Then
    Turtle.Move(distance)
    Turtle.Turn(angle)

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

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

Foto da Flickr

Figura 59 – Recuperare immagini da Flickr

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

Sub OnMouseDown
    pic = Flickr.GetRandomPicture("mountains, river")
    GraphicsWindow.DrawResizedImage(pic, 0, 0, 640, 480)
EndSub
```

Sfondo del desktop dinamico

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

Il gioco Paddle

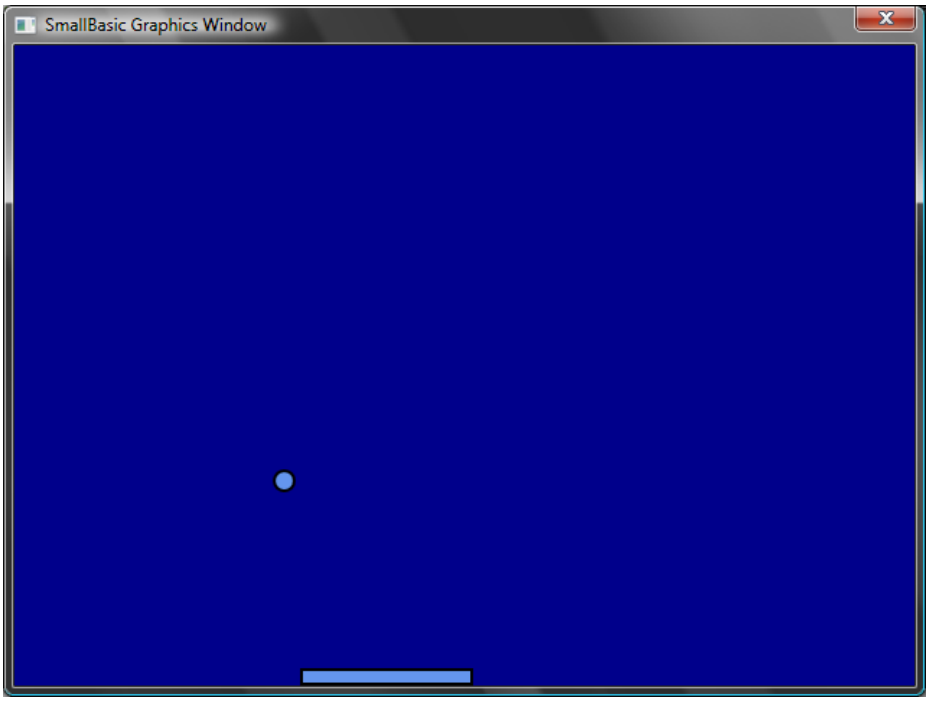

Figura 60 – Il gioco Paddle

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

```
GraphicsWindow.MouseMove = OnMouseMove

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

RunLoop:
  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(ball, x, y)
  Program.Delay(5)

  If (y < gh) Then
    Goto RunLoop
  EndIf

GraphicsWindow.ShowMessage("You Lose", "Paddle")

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

Appendice B

I colori

Ecco un elenco dei colori che hanno un nome in Small Basic, raggruppati per tipologia di colore.

Rossi

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

Rosa

Pink	#FFC0CB
LightPink	#FFB6C1
HotPink	#FF69B4
DeepPink	#FF1493
MediumVioletRed	#C71585
PaleVioletRed	#DB7093

Arancioni

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

Gialli

Gold	#FFD700
Yellow	#FFFF00
LightYellow	#FFFFE0
LemonChiffon	#FFFACD
LightGoldenrodYellow	#FAFAD2
PapayaWhip	#FFEFD5
Moccasin	#FFE4B5
PeachPuff	#FFDAB9

PaleGoldenrod	#EEE8AA
Khaki	#F0E68C
DarkKhaki	#BDB76B

Viola

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

Verdi

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

Azzurri

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

Marroni

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

Bianchi

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

Grigi

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