Microsoft Small Basic

Uma introdução à Programação

Capítulo 1

Introdução

Small Basic e a Programação

A Programação de Computadores é definida como o processo de criação de programas de computador utilizando linguagens de programação. Tal como falamos e entendemos Inglês, Espanhol ou Francês, os computadores podem entender programas escritos em certas linguagens. Estas são chamadas de linguagens de programação. Nos primeiros tempos, havia apenas algumas linguagens de programação e elas eram fáceis de aprender e de compreender. Mas, conforme os computadores e os programas começaram a ser cada vez mais sofisticados, as linguagens de programação evoluíram rapidamente, juntando conceitos cada vez mais complexos pelo caminho. Como resultado, a maioria das linguagens de programação modernas e os seus conceitos são um enorme desafio para os principiantes. Este facto começou a desencorajar as pessoas de aprender, ou tentar aprender, programação de computadores.

Small Basic é uma linguagem de programação que foi desenhada para tornar a programação extremamente simples, próxima e divertida para os principiantes. A intenção do Small Basic é derrubar as barreiras e de servir como introdução ao maravilhoso mundo da programação de computadores.

O Ambiente de Trabalho do Small Basic

Vamos começar por uma pequena introdução ao Ambiente de Trabalho do Small Basic. Quando se abre o Small Basic pela primeira vez, verá uma janela parecida com a seguinte figura:

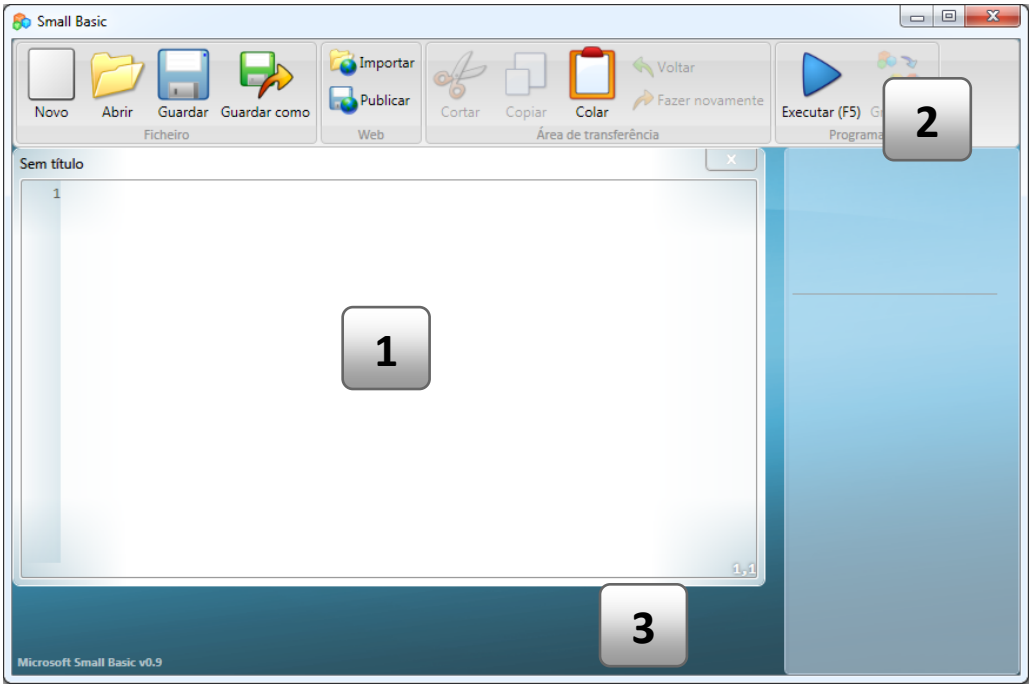


Figura 1 – O Ambiente de Trabalho do Small Basic

Este é o Ambiente de Trabalho do Small Basic, onde iremos escrever e correr os nossos programas de Small Basic. Este ambiente tem vários elementos distintos que estão identificados por números.

O **Editor**, identificado por [1] é onde iremos escrever os nossos programas de Small Basic. Quando se abre um programa de exemplo ou um programa previamente gravado, ele irá aparecer neste editor. Podemos depois modificá-lo e gravá-lo para usar mais tarde.

Podemos também abrir e trabalhar com mais do que um programa ao mesmo tempo. Cada programa em que estamos a trabalhar será mostrado num editor separado. O editor que contém o programa em que estamos actualmente a trabalhar é chamado de *editor activo*.

A **Barra de Ferramentas**, identificada por [2], é usada para emitir comandos seja no *editor activo* ou no ambiente de trabalho. Iremos aprender mais sobre os vários comandos na barra de ferramentas ao longo deste manual.

A **Superfície**, identificada por [3] é o lugar para onde todas as janelas do editor vão.

O nosso primeiro programa

Agora que estás familiarizado com o Ambiente de Trabalho do Small Basic, vamos prosseguir e começar a programar nele. Como foi referido acima, o editor é o lugar onde nós escrevemos os nossos programas. Então vamos começar e escrever a seguinte linha no editor.

```
TextWindow.WriteLine("Olá Mundo")
```

Este é o nosso primeiro programa de Small Basic. Se o escreveu correctamente, deverá conseguir ver algo semelhante à figura abaixo.

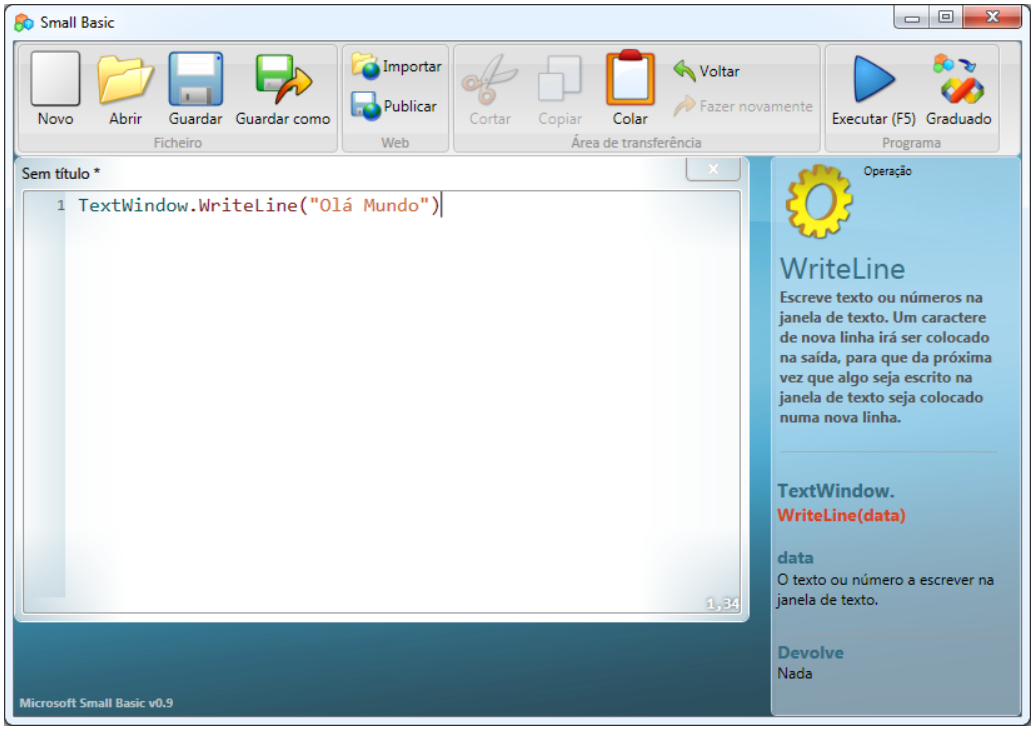


Figura 2 - Primeiro Programa

Agora que escrevemos o nosso programa, vamos prosseguir e corrê-lo para ver o que acontece. Podemos correr o nosso programa clicando no botão "Executar" na barra de ferramentas ou usando a tecla de atalho F5 no teclado. Se tudo correr bem, o nosso programa deverá correr com o resultado demonstrado abaixo.

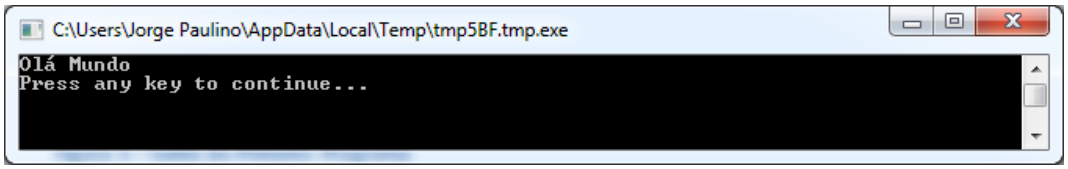


Figura 3 – Saída do Primeiro Programa

Parabéns! Acabamos de escrever e correr o nosso primeiro programa de Small Basic. Um programa pequeno e muito simples, mas ainda assim um grande passo no sentido de se tornar

Conforme escreves o teu primeiro programa, deverás ter notado que uma janela apareceu com uma lista de itens (Figura 4). Isto é chamado de "intellisense" e ajuda a escrever o programa mais rapidamente. Podes correr essa lista pressionando as teclas das setas Cima/Baixo e, quando encontras alguma coisa que pretendes, podes pressionar a tecla Enter para inserir esse item no teu programa.

um programador de computadores. Agora há só mais um detalhe para cobrir antes de irmos criar programas maiores. Nós temos que entender o que acabou de acontecer – o que dissemos exactamente ao computador e como o computador sabia o que fazer? No próximo capítulo, vamos analisar o programa que escrevemos, de modo a percebermos isso.

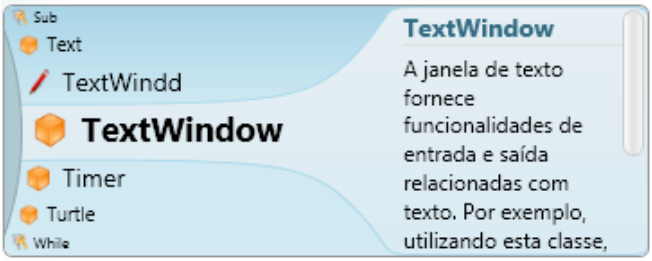


Figura 4 – "Intellisense"

Guardar o programa

Se pretendermos fechar o Small Basic e voltar mais tarde a trabalhar no programa que escrevemos, podes guardar o programa. De facto, é uma boa prática guardar os programas de tempos a tempos de modo a que não percamos informação no caso de haver um fecho acidental ou uma falha de energia. Podes guardar o programa corrente clicando no ícone "Guardar" na barra de ferramentas ou usando o atalho "Ctrl+G" (pressiona a tecla G enquanto mantém em baixo a tecla Ctrl).

Capítulo 2

Entender o nosso Primeiro Programa

O que é na realidade um programa de computador?

Um programa é um conjunto de instruções para o computador. Estas instruções dizem ao computador, com precisão, o que fazer e o computador segue sempre essas instruções. Tal como as pessoas, os computadores apenas podem seguir instruções se as mesmas forem especificadas numa linguagem que elas possam entender. Estas são chamadas de linguagens de programação. Existem muitas linguagens que o computador pode entender e o Small Basic é uma delas.

Imagine uma conversa entre ti e um teu amigo. Tu e o teu amigo usariam palavras, organizadas em frases para transmitir informação num sentido e no outro. Da mesma maneira, as linguagens de programação contêm colecções de palavras que podem ser organizadas em frases para transmitir informação ao computador. Os programas são basicamente conjuntos de frases (umas vezes são apenas algumas, outras vezes são milhares) que juntas fazem sentido tanto para o programador como para o computador da mesma forma.

Existem muitas linguagens que o computador pode entender. Java, C++, Python, VB, etc. são todas linguagens de programação potentes e modernas que são usadas para desenvolver programas, do mais simples aos mais complexos.

Programas Small Basic

Um programa de Small Basic típico consiste numa série de *instruções*. Cada linha do programa representa uma instrução e cada instrução é uma ordem para o computador. Quando pedimos ao computador para executar um programa de Small Basic, ele pega no programa e lê a primeira instrução. Ele entende o que lhe estamos a tentar dizer e executa essa instrução. Quando acabar de executar a

nossa primeira instrução, ele volta ao programa e lê e executa a segunda linha. Ele continua a repetir estes passos até que chegue ao fim do programa. É aí que o nosso programa termina.

De volta ao nosso Primeiro Programa

Este foi o primeiro programa que escrevemos:

```
TextWindow.WriteLine("Olá Mundo")
```

Este é um programa muito simples que consiste numa *frase*. Esta frase diz ao computador para escrever uma linha de texto que é **Olá Mundo**, na Janela de Texto.

Ela traduz-se literalmente na mente do computador em:

```
Escrever Olá Mundo
```

Poderá já ter reparado que a instrução pode por sua vez ser separada em segmentos mais pequenos, tal como as frases podem ser separadas em palavras. Na nossa primeira instrução temos 3 segmentos distintos:

a) TextWindow
b) WriteLine
c) "Olá Mundo"

O ponto, o parêntese ou as aspas são todos pontuações que têm que ser colocados nas posições apropriadas na instrução, para que o computador entenda a nossa intenção.

Provavelmente ainda se lembra da janela preta que apareceu quando corremos o nosso primeiro programa. Essa janela preta é chamada de Janela de Texto ou por vezes referida como Consola. É nela que o resultado do programa surge. A **Janela de Texto** no nosso programa é chamada de objecto. Existe um certo número destes objectos disponíveis para utilizar nos nossos programas. Podemos executar diversas *operações* sobre estes objectos. Já utilizámos a instrução *WriteLine* no nosso programa. Poderá também ter reparado que a instrução WriteLine é seguida de **Olá Mundo** dentro de aspas. Este texto é passado com entrada para a instrução WriteLine, que por sua vez imprime para o utilizador. Isto é chamado de *entrada* para a instrução. Algumas instruções têm uma ou mais entradas enquanto outras não têm nenhuma.

Pontuações como as aspas, espaços e parêntese são muito importantes num programa de computador. Baseado na sua posição e na sua quantidade, eles podem alterar o sentido do que está a ser expressado.

O nosso Segundo Programa

Agora que compreendemos o nosso primeiro programa, vamos continuar e torna-lo mais bonito ao adicionando-lhe algumas cores.

```
TextWindow.ForegroundColor = "Yellow"
TextWindow.WriteLine("Olá Mundo")
```

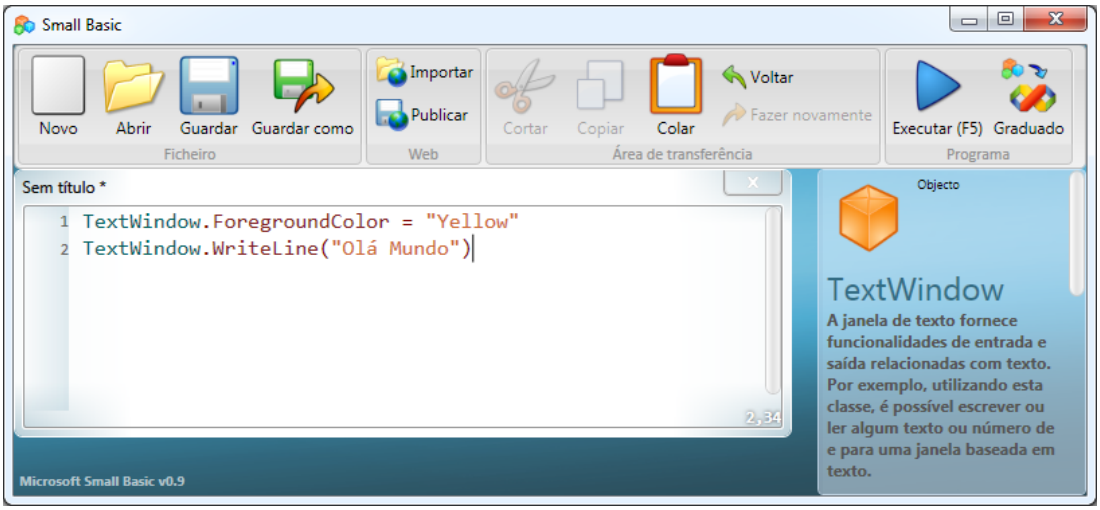


Figura 6 – Adicionando Cores

Quando corremos o programa acima, irá notar que ele imprime a mesma frase "Olá Mundo" dentro da Janela de Texto, mas desta vez imprime a amarelo em vez do cinzento em que imprimiu anteriormente.

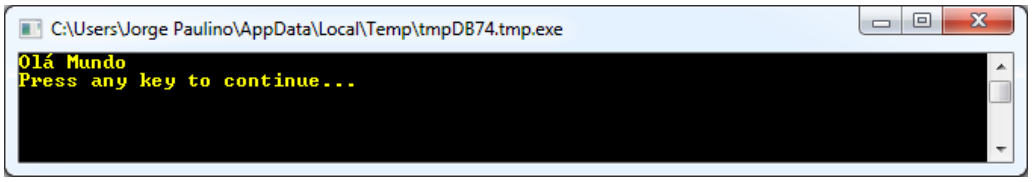


Figura 5 – Olá Mundo a amarelo

Repare na nova instrução que adicionamos ao nosso programa original. Ele usa uma nova palavra, *ForegroundColor* à qual atribuímos o valor "*Yellow*". Isto significa que atribuímos o valor "*Yellow*" a *ForegroundColor*. Agora a diferença entre ForegroundColor e a instrução WriteLine é que ForegroundColor não recebeu entradas nem precisou de parênteses. Em vez disso, foi seguida por um sinal *igual a* e uma palavra. Nós definimos *ForegroundColor* como uma *propriedade* da Janela de Texto. Aqui fica uma lista dos valores que são válidos como propriedade para a instrução ForegroundColor. Experimente substituir "Yellow" por outro destes valores e veja os resultados – não se esqueça das aspas, elas são pontuações obrigatórias.

```
Black
```

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

Capitulo 3

Introdução às Variáveis

Utilizando Variáveis no nosso programa

Não era bom que o nosso programa pudesse dizer "Olá" com o nome to utilizador em vez de dizer apenas um "Olá Mundo"? De modo a conseguirmos isso, temos primeiro que perguntar ao utilizador pelo seu nome e guardá-lo algures e depois imprimir "Olá" juntamente com o nome do utilizador. Vamos então ver como podemos fazer isso:

```
TextWindow.Write("Insira o seu nome: ")
nome = TextWindow.Read()
TextWindow.WriteLine("Olá " + nome)
```

Quando escrevemos e executamos o programa, iremos ver um resultado como o seguinte:

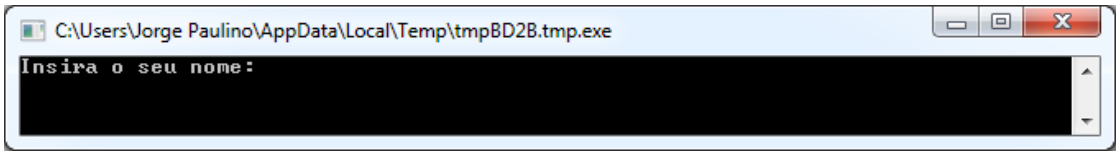


Figura 6 – Pergunta o nome do utilizador

E quando inserimos o nosso nome e pressionamos a tecla ENTER, iremos ver o seguinte resultado:

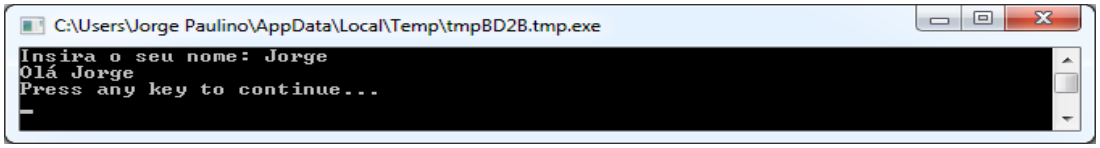


Figure 7 – Um Olá personalizado

Agora, se corrermos o programa de novo, será questionado de novo pelo mesmo. Poderá inserir um nome diferente e o computador dirá Olá com esse nome.

Análise do programa

No programa que acabámos de correr, a linha que lhe deverá ter chamado a atenção será provavelmente esta:

```
nome = TextWindow.Read()
```

Read() parece o mesmo que *WriteLine()*, mas sem entradas. É uma operação e basicamente diz ao computador para esperar que o utilizador escreva qualquer coisa e pressione a tecla ENTER. Assim que o utilizador pressiona a tecla ENTER, ele pega no que o utilizador escreveu e retorna-o para o programa. O mais interessante é que qualquer que seja o texto que o utilizador escreva, é agora guardado numa *variável* chamada **nome**. A *variável* é definida como um lugar onde podemos guardar valores temporariamente e usá-los mais tarde. Na linha abaixo, **nome** foi usado para guardar o nome do utilizador.

A próxima linha também é interessante:

```
TextWindow.WriteLine("Olá " +
nome)
```

Este é o lugar onde nós usamos o valor guardado na nossa variável **nome**. Pegamos no valor guardado em **nome** e juntamo-lo a "Olá" e escrevemos para a Janela de Texto.

Write, tal como WriteLine é outra operação na Janela da Consola. Write permite-lhe escrever alguma coisa para a Janela da Consola mas permite que texto consecutivo esteja na mesma linha que o texto corrente.

Uma vez o valor de uma variável definido, podemos reutilizá-la qualquer número de vezes. Por exemplo, podemos fazer o seguinte:

```
TextWindow.Write("Insira o seu nome: ")
nome = TextWindow.Read()
TextWindow.Write("Olá " + nome + ".  ")
TextWindow.WriteLine("O que estás a fazer " + nome + "?")
```

E vermos o seguinte resultado:

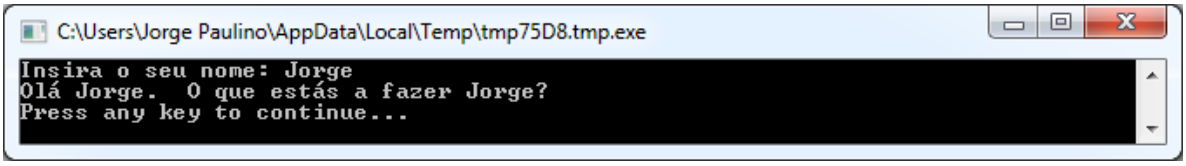


Figura 8 - Reutilizando a Variável

Regras para nomes de Variáveis

As variáveis têm nomes associados e é dessa maneira que as conseguimos identificar. Existem algumas regras simples e algumas normas para definir os nomes dessas variáveis. Elas são:

1. O nome tem que começar por uma letra e não deverá coincidir com nenhuma palavra-chave como **if**, **for**, **then**, etc.
2. O nome poderá conter qualquer combinação de letras, dígitos e underscores.
3. É útil atribuir nomes significativos às variáveis uma vez que as variáveis podem ter nomes tão compridos quanto quiser, utilize os nomes das variáveis para descrever a sua intenção.

Brincando com os Números

Acabámos de ver como podemos usar variáveis para guardar o nome do utilizador. Nos próximos programas, iremos ver como podemos guardar e manipular números em variáveis. Vamos começar com um programa muito simples:

```
numero1 = 10
numero2 = 20
numero3 = numero1 + numero2
TextWindow.WriteLine(numero3)
```

Quando corremos o programa, iremos ter o seguinte resultado:

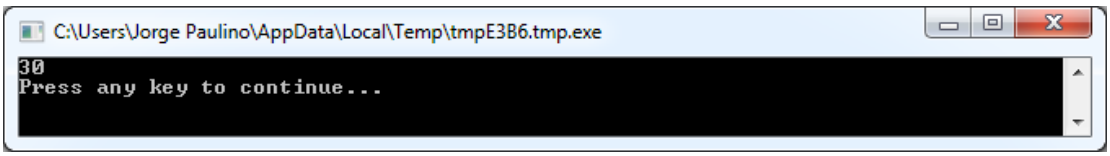


Figura 9 – Adicionando dois números

Na primeira linha do programa, estamos a atribuir o valor 10 à variável **numero1**. Na segunda, estamos a atribuir o valor 20 à variável **numero2**. Na terceira linha, estamos a somar **numero1** e **numero2** e a atribuir o resultado à variável

Repare que os números não têm aspas em volta deles. Para números, não são necessárias aspas. Apenas necessita de colocar aspas quando estamos a usar texto.

numero3. Neste caso, a variável **numero3** irá ter um valor de 30. E isto será o que iremos imprimir para a nossa TextWindow.

Agora vamos modificar o programa ligeiramente e ver os resultados:

```
numero1 = 10
numero2 = 20
numero3 = numero1 * numero2
TextWindow.WriteLine(numero3)
```

O programa acima irá multiplicar **numero1** com **numero2** e guardar o resultado em **numero3**. Poderás ver o resultado deste programa aqui:

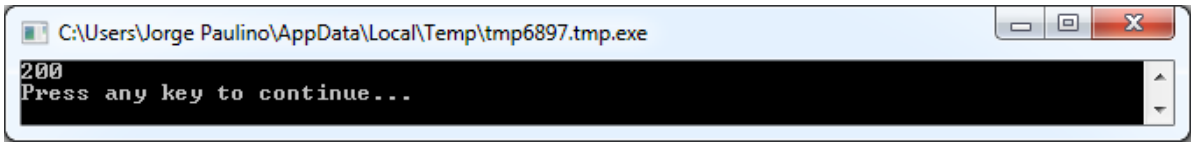


Figura 10 – Multiplicando dois Números

De igual modo, podemos subtrair ou dividir números. Aqui fica a subtracção:

```
numero3 = numero1 - numero2
```

E o símbolo para a divisão é '/'. O programa ficará assim:

```
numero3 = numero1 / numero2
```

E o resultado desta divisão deverá ser:

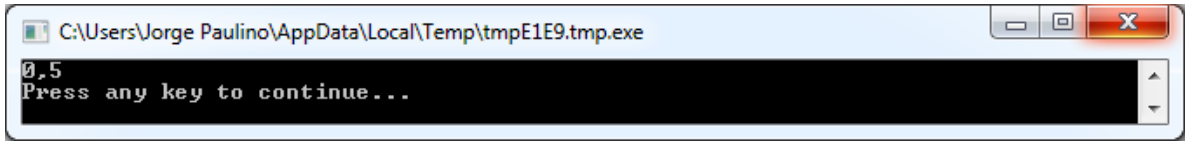


Figura 11 - Dividindo dois Números

Uma Simples conversão de Temperatura

Para o próximo programa, iremos usar a fórmula $℃ = \frac{5(°F - 32)}{9}$ para converter temperaturas em Fahrenheit para temperaturas em Celsius.

Primeiro, iremos recolher a temperatura em Fahrenheit do utilizador e guardá-la numa variável. Existe uma operação especial que nos permite ler números do utilizador que é a **TextWindow.ReadNumber**.

```
TextWindow.Write("Introduza a temperatura em Fahrenheit: ")
fahr = TextWindow.ReadNumber()
```

Logo que tenhamos a temperatura em Fahrenheit guardada numa variável, podemos convertê-la para Celsius assim:

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

Os parênteses dizem ao computador para calcular a parte **fahr – 32** primeiro e só depois processar o resto. Agora tudo o que temos de fazer é imprimir o resultado para o utilizador. Juntando tudo, ficamos com este programa:

```
TextWindow.Write("Introduza a temperatura em Fahrenheit: ")
fahr = TextWindow.ReadNumber()
celsius = 5 * (fahr - 32) / 9
TextWindow.WriteLine("Temperatura em Celsius é " + celsius)
```

Assim, o resultado deste programa será:

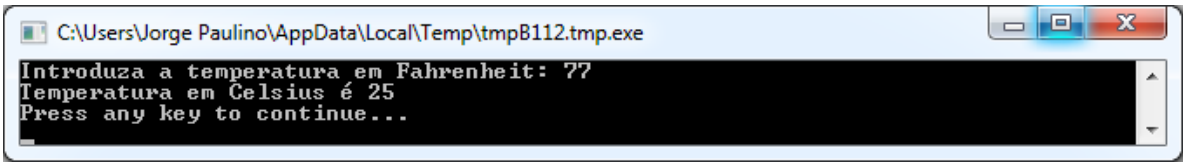


Figura 12 – Conversão de Temperatura

Capítulo 4

Condições e "Branching"

De volta ao nosso primeiro programa, não seria bom que em vez de dizer o genérico *Olá Mundo* que pudéssemos dizer *Bom Dia Mundo* ou *Boa Noite Mundo*, dependendo da hora do dia? Para o nosso próximo programa, iremos fazer o computador dizer *Bom Dia Mundo* se a hora for anterior às 12PM(antes do meio dia); e *Boa Noite Mundo* se a hora for posterior às 12PM(depois do meio dia).

```
If (Clock.Hour < 12) Then
  TextWindow.WriteLine("Bom Dia Mundo")
EndIf
If (Clock.Hour >= 12) Then
  TextWindow.WriteLine("Boa Noite Mundo")
EndIf
```

Dependendo de quando corrermos o programa, iremos ver um dos seguintes resultados:

Figura 13 – Bom Dia Mundo

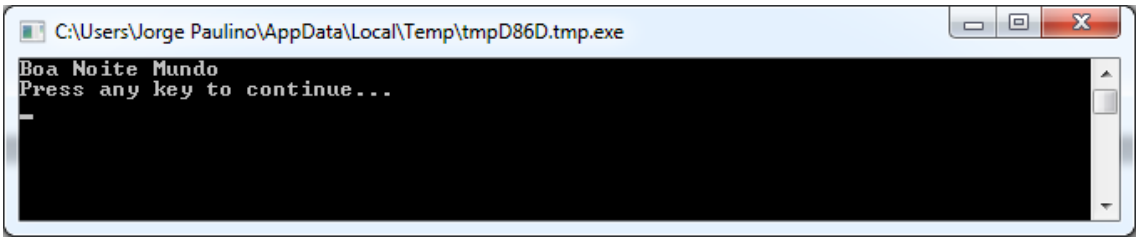


Figura 145 – Boa Noite Mundo

Vamos analisar as primeiras três linhas do nosso programa. Se calhar já viu que estas linhas dizem ao computador que, se Clock.Hour é menor do que 12, imprimir "Bom Dia Mundo." As palavras **If**, **Then** and **EndIf** são palavras reservadas que são interpretadas pelo computador quando o programa é executado. A palavra **If** é sempre seguida de uma condição, que neste caso é (**Clock.Hour < 12)**. Lembre-se que os parênteses são necessários para o computador entender quais são as suas intenções. A condição é seguida por **then** e pela operação a executar. Depois da operação vem **EndIf**. Isto diz ao computador que a execução das condições está terminada.

Em Small Basic, podemos usar o objecto Clock para aceder à data e hora actual. Ele também providencia uma série de propriedades que nos permitem aceder ao Dia, Mês, Ano, Hora, Minutos, Segundos separadamente.

Entre o **then** e o **EndIf**, poderá haver mais do que uma operação e o computador irá executá-las todas só se a condição for válida. Por exemplo, podes escrever qualquer coisa como isto:

```
If (Clock.Hour < 12) Then
  TextWindow.Write("Bom Dia. ")
  TextWindow.WriteLine("Como foi o pequeno-almoço?")
EndIf
```

Else

No programa no inicio deste capítulo, poderá ter reparado que a segunda condição é um pouco redundante. O valor de **Clock.Hour** poderá ser ou menor do que 12 ou não. Na realidade, não temos que efectuar a segunda verificação. Em alturas como esta, podemos encurtar os dois conjuntos de instruções **if..then..endif** para apenas um usando uma nova palavra, **else**.

Se quisermos rescrever o nosso programa usando **else**, ele ficaria assim:

```
If (Clock.Hour < 12) Then
  TextWindow.WriteLine("Bom Dia Mundo")
Else
  TextWindow.WriteLine("Boa Noite Mundo")
EndIf
```

E este programa irá fazer exactamente o mesmo que o outro o que nos trás para uma lição muito importante em termos de programação de computadores:

> *Em programação, existem normalmente várias maneiras de fazer a mesma coisa. Por vezes uma maneira faz mais sentido do que outra. As escolhas são deixadas para o programador. À medida que escreves mais programas e vais ficando mais experiente, começarás a notar estas diferentes técnicas e as suas vantagens e desvantagens.*

Tabulação

Em todos os exemplos, pode ver que as instruções *If, Else* e *EndIf* têm uma tabulação. Esta tabulação não é necessária. O computador irá entender o programa correctamente sem elas. No entanto, elas ajudam a ver e a entender a estrutura do programa mais facilmente. Daí, é normal considerarmos uma boa prática o uso de tabular as instruções entre blocos.

Par ou Ímpar

Agora que temos as instruções **If..Then..Else..EndIf** no nosso saco de truques, vamos escrever um programa que, dado um número, nos diga se ele é par ou ímpar.

```
TextWindow.Write("Introduza um número: ")
num = TextWindow.ReadNumber()
remainder = Math.Remainder(num, 2)
If (remainder = 0) Then
  TextWindow.WriteLine("O número é Par")
Else
  TextWindow.WriteLine("O número é Ímpar")
EndIf
```

E, quando corrermos o programa, iremos ver este resultado:

Figura 15 – Par ou Ímpar

Neste programa introduzimos outra operação útil, **Math.Remainder**. E sim, como já deve ter percebido, **Math.Remainder** irá dividir o primeiro número pelo segundo e retornar o restante.

“Branching”

Lembre-se, no segundo capítulo aprendemos que o computador processa um programa, uma instrução de cada vez, de cima para baixo. No entanto, existe uma instrução especial que pode fazer o computador saltar para uma instrução fora de ordem. Vamos dar uma olhadela ao próximo programa.

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

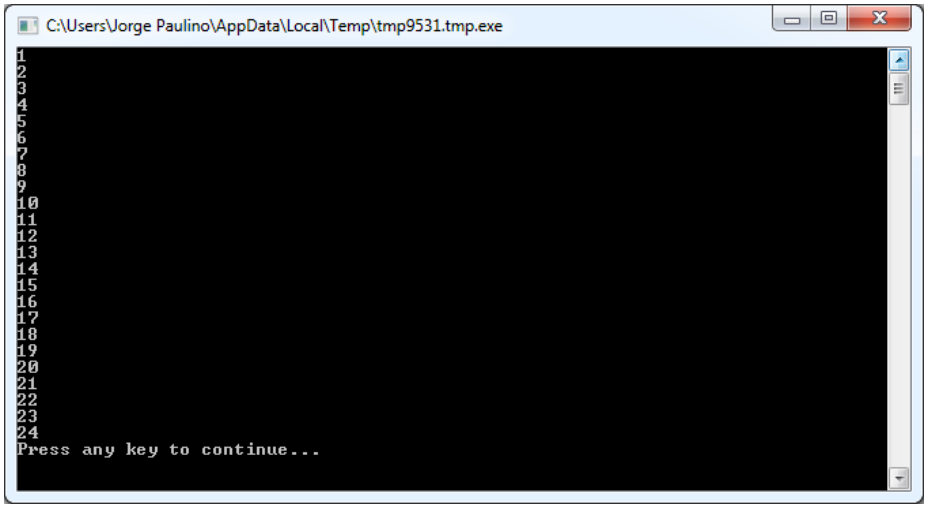


Figura 16 - Usando Goto

No programa acima, atribuímos um valor de 1 à variável **i**. Depois adicionámos uma nova instrução que acaba com dois pontos (:)

```
start:
```

Isto chama-se uma *etiqueta*. Etiquetas são uma espécie de marcadores que o computador entende. Pode dar-se o nome que se quiser ao marcador e podem adicionar-se as etiquetas que se quiserem no programa, desde que não se repitam os nomes.

Outra instrução interessante é:

```
i = i + 1
```

Isto apenas diz ao computador para somar 1 à variável **i** e atribuir o resultado de volta a **i**. Assim, se o valor de **i** era 1 antes desta instrução, ele será 2 depois da mesma ser executada.

E finalmente,

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

Esta é a parte que diz ao computador que, se o valor de **i** é menor que 25, começa a executar as instruções do marcador **start**.

Execução sem fim

Usando a instrução **Goto,** podemos fazer o computador repetir alguma coisa um certo número de vezes. Por exemplo, podemos pegar no programa Par ou Ímpar e modificá-lo conforme imagem em baixo, e o programa irá correr para sempre. Podemos parar o programa pressionando o botão (X) no canto superior direito da janela.

```
inicio:
TextWindow.Write("Introduza um número: ")
num = TextWindow.ReadNumber()
remainder = Math.Remainder(num, 2)
If (remainder = 0) Then
  TextWindow.WriteLine("O Número é Par")
Else
  TextWindow.WriteLine("O Número é Ímpar")
EndIf
Goto inicio
```

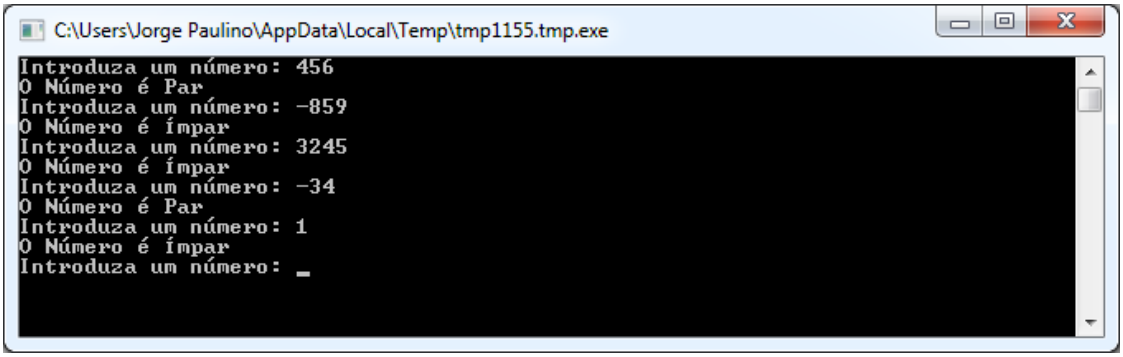


Figura 17 – Par ou Ímpar a executar sem fim

Capítulo 5

Ciclos

Ciclo For

Vamos pegar no programa que escrevemos no capítulo anterior.

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

Este programa imprime números ordenados de 1 a 24. Este processo de incrementar a variável, que é muito comum em programação, que normalmente as linguagens de programação têm um método mais fácil de fazer isto. O programa acima é o equivalente ao programa em baixo:

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

O seu resultado é:

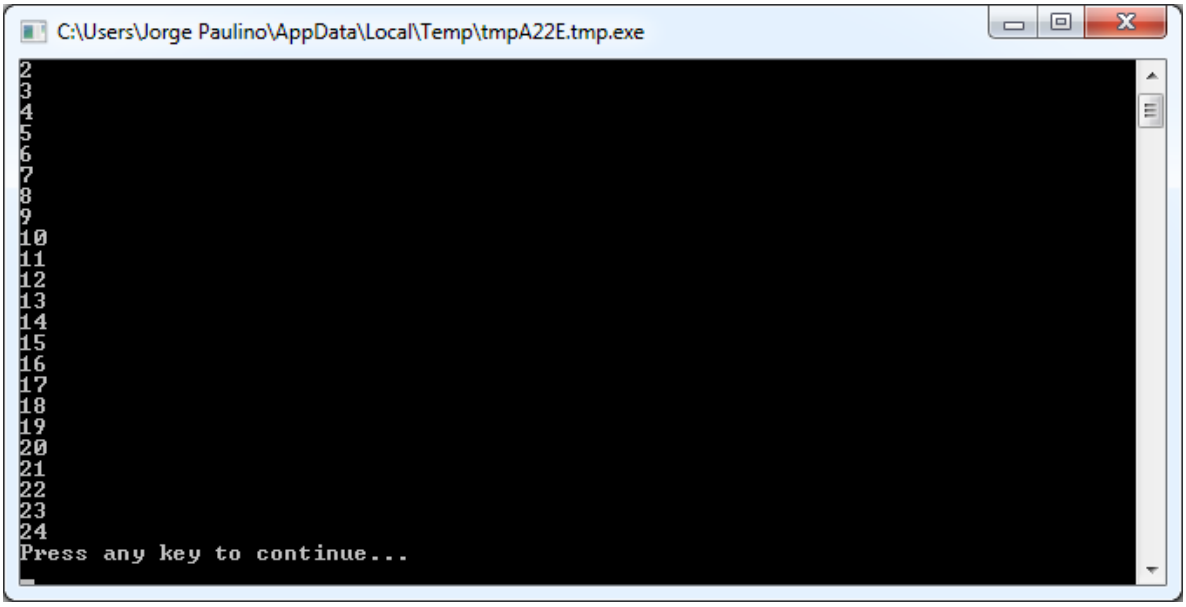


Figura 18 – Usando o ciclo For

Repare que reduzimos as 8 linhas do programa para 4, e ele ainda faz exactamente o mesmo que as 8 linhas! Lembra-se anteriormente de termos dito que existem normalmente várias maneiras de fazer a mesma coisa? Este é um bom exemplo!

For..EndFor é, em termos de programação, chamado de *ciclo*. Ele permite-lhe pegar numa variável, dar-lhe um valor inicial e um valor final e dizer ao computador para incrementar a variável por nós. De cada vez que o computador incrementa a variável, ele executa as instruções entre os comandos **For** e **EndFor**.

Mas se quisesse que a variável fosse incrementada em 2 em vez de 1 – útil se quiser por exemplo imprimir todos os números ímpares entre 1 e 24, pode usar um ciclo para fazer isso também.

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

Figura 19 – Só os Números Ímpares

A parte **Step 2** da instrução **For** diz ao computador para incrementar o valor de **i** em 2 em vez do 1 habitual. Ao usar **Step** podemos especificar qual o incremento que queremos. Podemos até especificar um valor negativo para o passo e fazer o computador contar para trás, como no exemplo em baixo:

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

Figura 20 – Contar para trás

Ciclo While

O ciclo **While** é ainda outro método de ciclo que é usado especificamente quando a contagem do ciclo não é ainda conhecida previamente. Enquanto o ciclo For corre por uma pré-definida quantidade de vezes, o ciclo While corre enquanto uma condição é verdadeira. No exemplo em baixo, dividimos um número por 2 enquanto o resultado é maior do que 1.

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

```
EndWhile
```

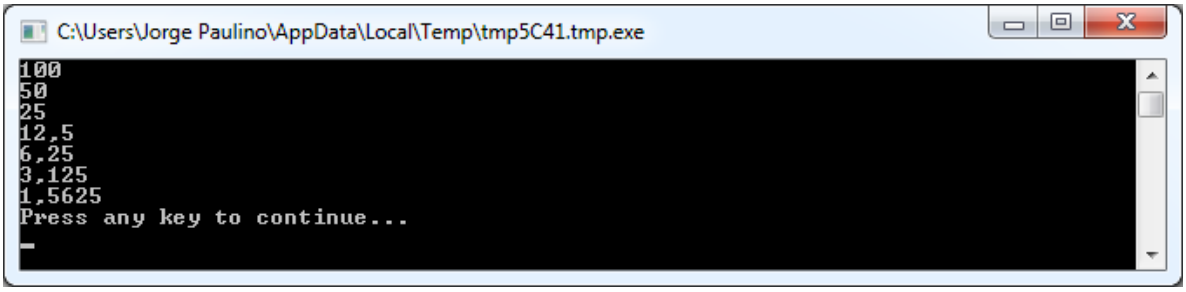


Figura 21 – Ciclo dividindo

No programa acima, atribuímos o valor de 100 à variável *numero* e corremos o ciclo while enquanto o valor de *numero* é maior do que 1. Dentro do ciclo, imprimimos o número e depois dividimo-lo por 2. Tal como esperado, o resultado do programa são números que vão sendo progressivamente sendo divididos ao meio.

Seria muito difícil de escrever este programa usando um ciclo For porque não sabemos quantas vezes o ciclo irá correr. Com um ciclo While é fácil de verificar a condição e pedir ao computador para continuar o ciclo ou parar.

É interessante notar que cada ciclo While pode ser embrulhado dentro de uma instrução If...Then. Por exemplo, o programa acima pode ser reescrito da seguinte forma, sem afectar o resultado final.

De facto, internamente, o computador rescreve cada ciclo While em instruções que usam If..Then juntamente com uma ou mais instruções Goto.

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

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

Capítulo 6

Introdução aos Gráficos

Até agora, em todos os nossos exemplos, usámos a TextWindow para explicar os fundamentos da linguagem Small Basic. No entanto, o Small Basic vem com um poderoso conjunto de capacidades gráficas que iremos começar a explorar neste capítulo.

Introdução à GraphicsWindow

Tal como temos a TextWindow que nos permite trabalhar com Texto e Números, o Small Basic também providencia uma **GraphicsWindow** que podemos usar para desenhar coisas. Vamos começar por mostrar a GraphicsWindow.

```
GraphicsWindow.Show()
```

Quando correr este programa, irá notar que, em vez da habitual caixa de texto preta, irá aparecer uma janela branca, como a que é mostrada em baixo. Por enquanto, não há muito a fazer nesta janela. Mas esta será a janela base na qual iremos trabalhar neste capítulo. Pode fechar esta janela clicando no botão 'X' no canto superior direito.

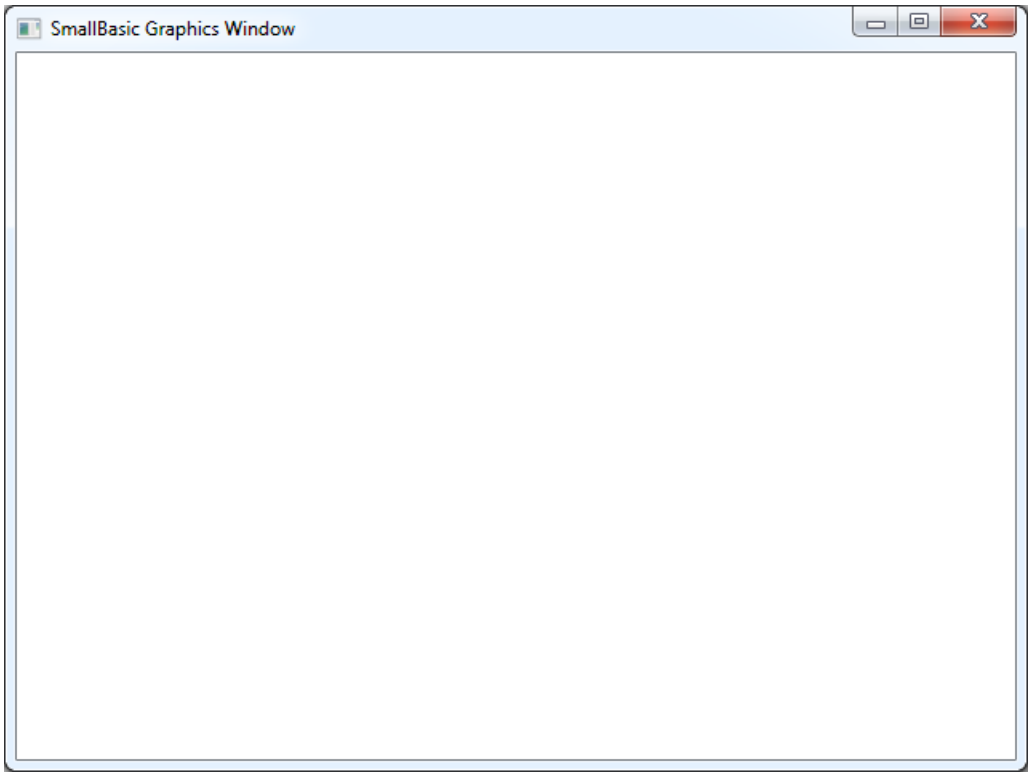


Figura 22 – Uma Graphic Window vazia

Formatando a Graphics Window

A janela gráfica pode ser personalizada a seu gosto. Pode alterar o título, o fundo e o seu tamanho. Vamos avançar e modificá-la um pouco, só para se familiarizar com a janela.

```
GraphicsWindow.BackgroundColor = "SteelBlue"
GraphicsWindow.Title = "A Minha Janela Gráfica"
GraphicsWindow.Width = 320
GraphicsWindow.Height = 200
GraphicsWindow.Show()
```

Isto é como a janela personalizada fica. Pode mudar a cor do fundo para um dos muitos valores listados no Anexo B. Brinque com estas propriedades para ver como pode modificar a aparência da janela.

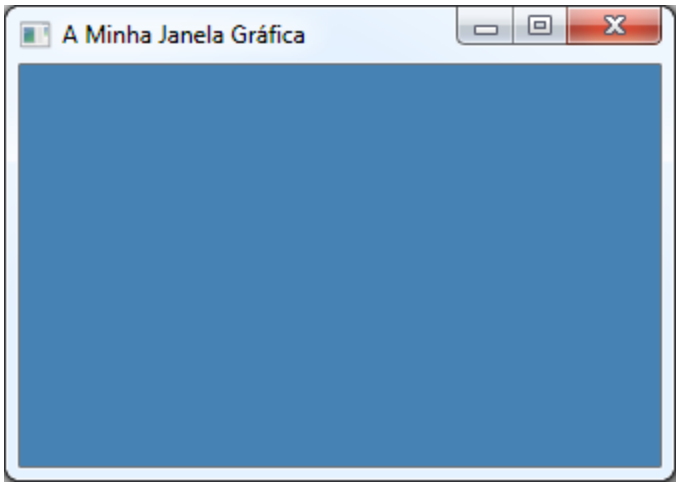
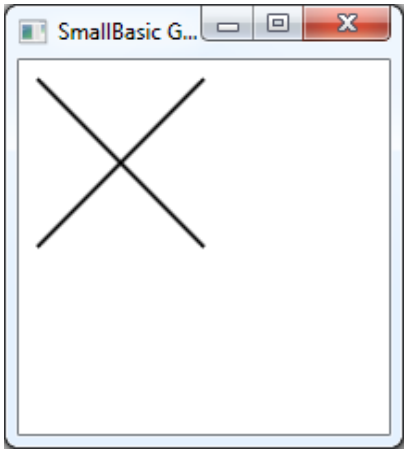


Figura 23 – Uma Graphics Window Personalizada

Desenhando Linhas

Uma vez que tenha a GraphicsWindow aberta, pode desenhar formas, texto e até fotos nela. Vamos começar por desenhar algumas formas simples. Aqui fica um programa que desenha um par de linhas na Graphics Window.

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



Em vez de usar nomes para as cores, pode usar a notação de cores da internet (#RRGGBB). Por exemplo, #FF0000 representa Vermelho, #FFFF00 para Amarelo, e por aí a fora. Iremos aprender mais acerca das cores no Capítulo das Cores.

Figura 24 – Linhas Cruzadas

As duas primeiras linhas do programa, definem a janela, as restantes duas desenham as linhas cruzadas. Os dois primeiros números a seguir a *DrawLine* especificam as coordenadas iniciais x e y e os outros dois especificam as coordenadas finais de x e y. O interessante nos gráficos em computador, é que as coordenadas (0, 0) começam no canto superior esquerdo da janela. Aliás, nas coordenadas de espaciais a janela é considerada como estando no 2º quadrante.

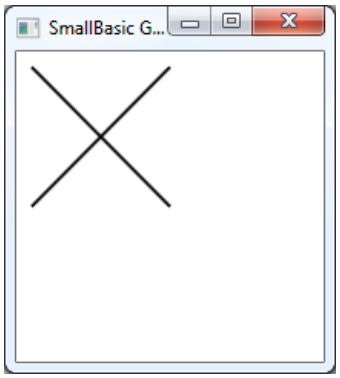


Figura 25 – O mapa das Coordenadas

De volta ao programa, é interessante notar que o Small Basic permite-lhe modificar as propriedades da linha, tais como cor e espessura. Primeiro, vamos modificar a cor das linhas, tal como mostrado no programa em baixo.

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

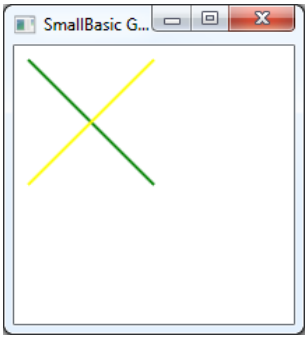


Figura 26 – Modificando a Cor da Linha

Agora, vamos modificar o também o tamanho. No programa em baixo, modificámos a espessura da linha para ser 10, em vez da espessura por defeito que é 1.

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

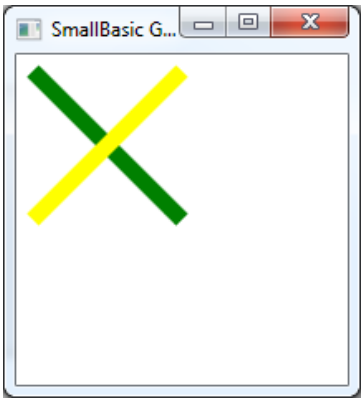


Figura 27 – Linhas Coloridas Espessas

PenWidth e *PenColor* modificam a caneta com a qual estas linhas são desenhadas. Elas não afectam apenas as linhas mas também qualquer forma que seja desenhada depois destas propriedades serem alteradas.

Ao usar as instruções de ciclos, que aprendemos nos capítulos anteriores, podemos facilmente escrever programas que desenhem múltiplas linhas com espessura da caneta 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
```

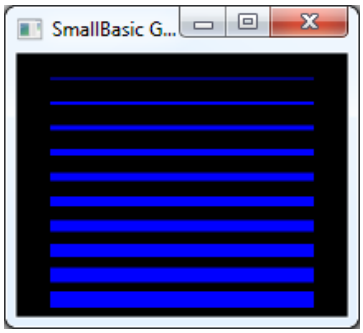


Figura 28 - Múltiplas Espessuras Caneta

O interessante neste programa é o ciclo, onde nós incrementamos a espessura da caneta (*PenWidth*), de cada vez que o ciclo corre, e depois desenha uma nova linha por baixo da anterior.

Desenhando e Pintando Formas

Quando se trata de desenhar formas, existem normalmente dois tipos de operações para cada uma. Elas são operações Desenho (*Draw*) e Preenchimento (*Fill*). As operações de Desenho, desenham o contorno da forma, usando a caneta, e as operações de Preenchimento, pintam a forma usando o pincel. Por exemplo, no programa abaixo, existem dois rectângulos, um é desenhado usando uma caneta Vermelha e o outro é pintado usando um pincel 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)
```

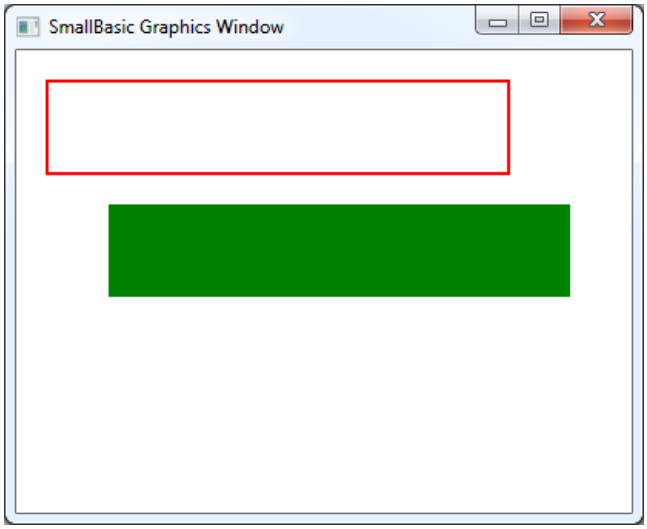


Figura 29 – Desenhando e Pintando

Para desenhar ou pintar um rectângulo, necessita de quatro números. Os dois primeiros números representam as coordenadas X e Y para o canto superior esquerdo do rectângulo. O terceiro número especifica a largura do rectângulo enquanto o quarto especifica a sua altura. De facto, o mesmo se aplica para desenhar e pintar elipses, tal como mostrado no programa em baixo.

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

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

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

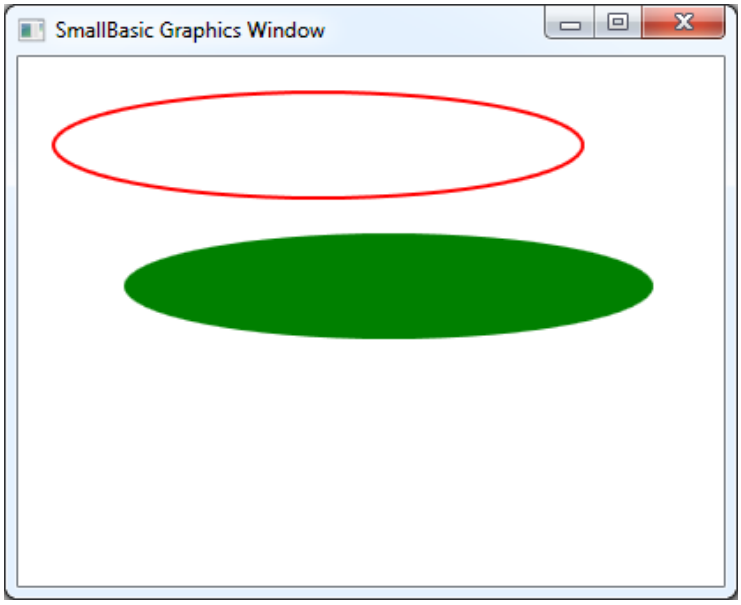


Figura 30 – Desenhando e Pintando Elipses

Elipses são apenas uma espécie de círculos. Se pretende desenhar círculos, tem de especificar a mesma altura e largura.

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

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

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

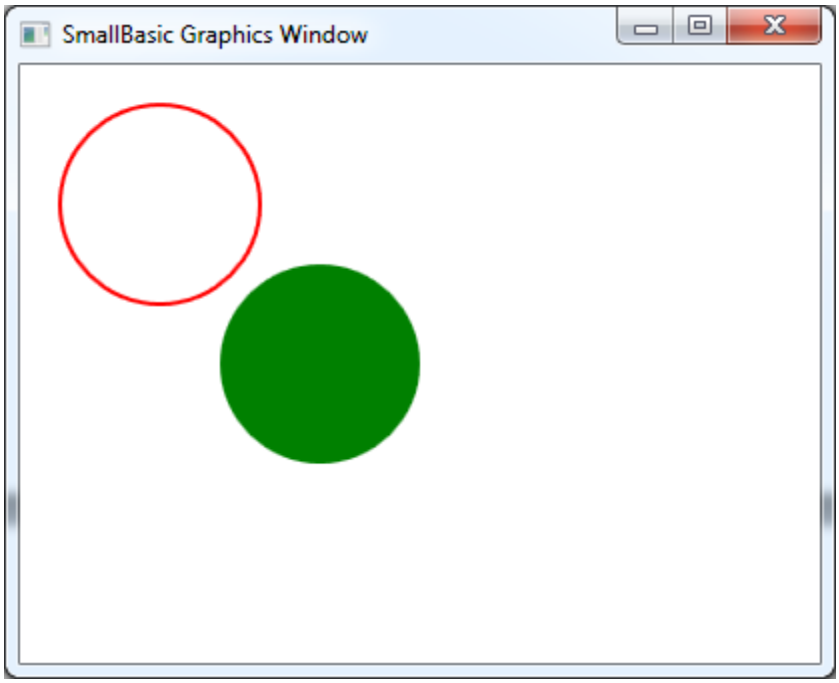


Figura 31 – Círculos

Capitulo 7

Diversão com Formas

Neste capítulo vamo-nos divertir com o que já aprendemos até agora. Este capítulo contém exemplos que mostram algumas formas interessantes de combinar o que aprendemos até agora para criar alguns programas com uma boa apresentação.

Rectangalore

Aqui vamos desenhar múltiplos rectângulos num ciclo, aumentando o tamanho do mesmo.

```
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
```

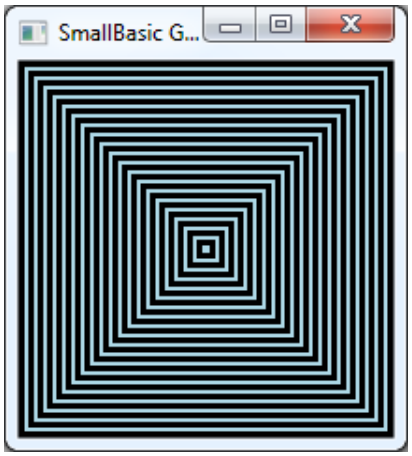


Figura 32 - Rectangalore

Circtacular

Uma variação do programa anterior, mas desta vez desenhando círculos em vez de rectângulos.

```
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
```

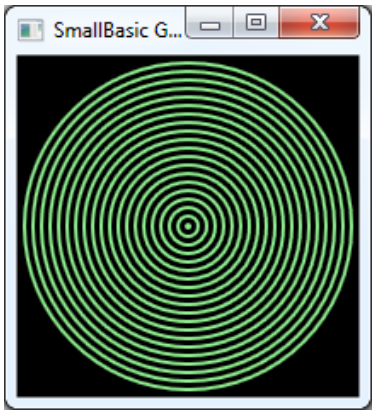


Figura 33 – Circtacular

Randomize

Este programar usa a operação *GraphicsWindow.GetRandomColor* para definir cores aleatórias para a linha e depois usa *Math.GetRandomNumber* para definir as coordenadas x e o y para os círculos. Estas duas operações podem ser combinadas de maneiras interessantes para criar programas interessantes que mostram resultados diferentes cada vez que a aplicação corre.

```
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
```

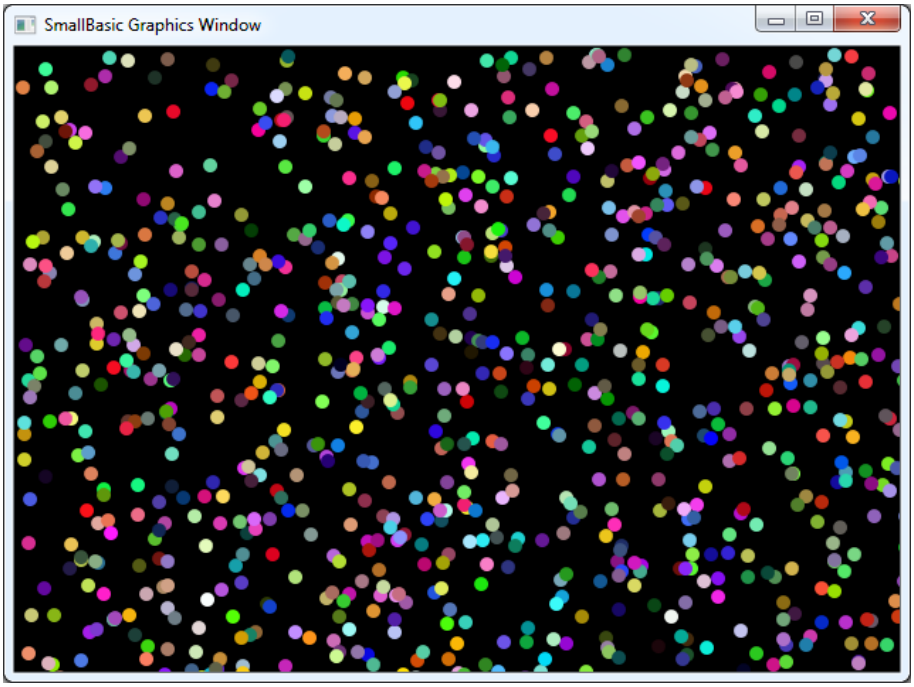


Figura 34 – Randomize

Fractals

O programa seguinte desenha um simples triângulo fractal usando números aleatórios. Um fractal é uma figura geométrica que pode se subdividida em partes, em que cada parte se assemelha com precisão à forma principal. Neste caso, o programa desenha centenas de triângulos em que todos se assemelham ao triângulo. E como o programa correr durante alguns segundos, pode-se ver os triângulos a formarem-se lentamente de alguns pontos. A lógica no entanto é um pouco difícil de descrever e será um bom exercido para explorar.

```
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
```

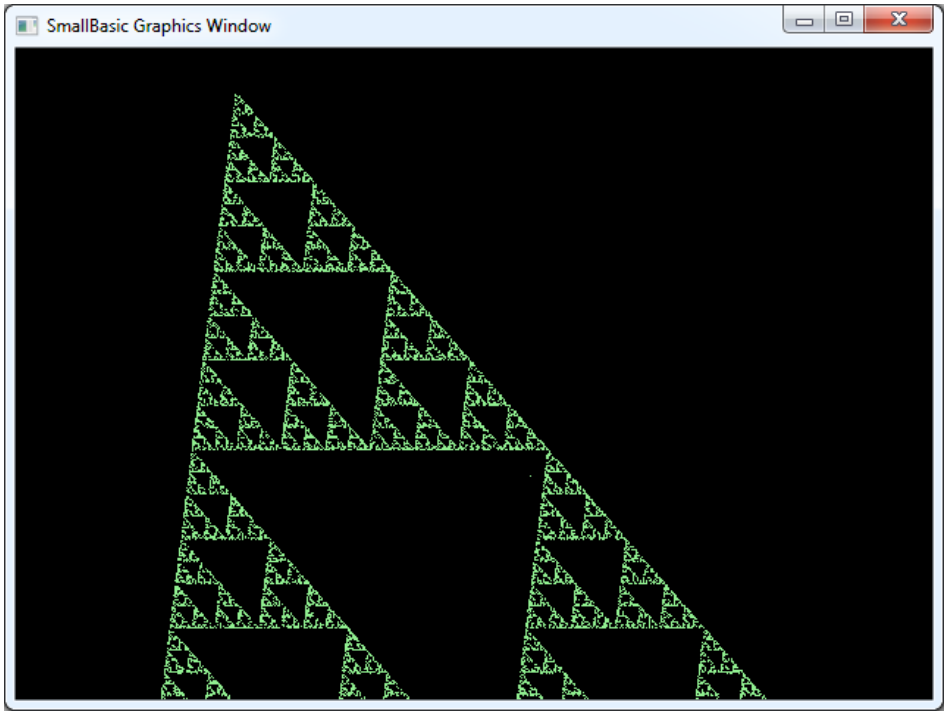


Figura 4 - Triângulo Fractal

Se quiser realmente ver os pontos formarem-se lentamente, pode fazer uma pausa no ciclo usando a operação *Program.Delay*. Esta operação utiliza um número, em milissegundos, que irá indicar o tempo a parar a execução do programa. Aqui está o programa modificado, com a alteração a negrito.

```
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
```

Aumentar a pausa irá tornar o programa mais lento. Experimente outros valores para escolher como mais gosta de ver o programa.

Outra modificação que pode fazer a este programa é substituir a seguinte linha:

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

por

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

Esta alteração irá fazer com que o programa desenhe os pixéis do triângulo de cores aleatórias.

Capitulo 8

Gráficos de Tartaruga

Logo

Nos anos 70, havia uma simples mas poderosa linguagem de programação, chamada *Logo* que era usada por alguns investigadores. Isto foi até alguém ter adicionado o que é chamado Turtle Graphics (Gráficos de Tartaruga) à linguagem e tornado disponível a *Turtle* (Tartaruga) que estava visível no ecrã e respondia a comandos como Avançar, Virar à Direita, Virar à Esquerda, etc. Usando a Tartaruga, as pessoas eram capazes de desenhar gráficos interessantes no ecrã. Isto tornou a linguagem acessível e apelativa a pessoas de todas as idades, e foi responsável pela sua grande popularidade nos anos 80s.

O Small Basic vem com um objecto **Turtle** com vários comandos que podem ser chamados pelos programas em Small Basic. Neste capítulo, vamos usar a *Turtle* para desenhar gráficos no ecrã.

A Tartaruga

Para começar, necessitamos de tornar a *Turtle* visível no ecrã. Isto pode ser feito com uma simples linha de código.

```
Turtle.Show()
```

Quando corre o programa vai surgir uma janela branca, como a que vimos no capítulo anterior, excepto que esta tem uma tartaruga (Turtle) no centro. É esta tartaruga que irá seguir as nossas instruções e desenhar o que lhe pedirmos.

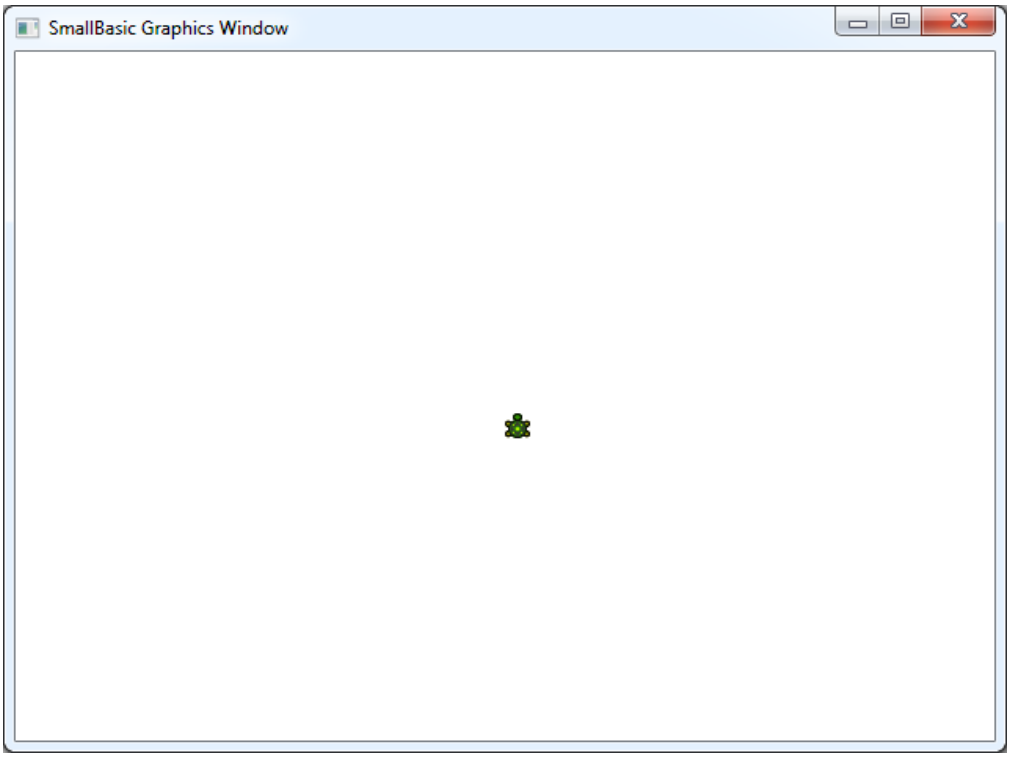


Figura 35 – Tartaruga visível

Movendo e Desenhando

Uma das instruções que a Tartaruga conhece é a **Move** (Mover). Esta operação usa um número como entrada. Este número diz à Tartaruga a distância que deverá mover. Por exemplo, no exemplo abaixo, vamos pedir à Tartaruga para mover 100 pixéis.

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

Quando correr o programa, pode ver a tartaruga a movimentar-se devagar, 100 pixéis para cima. À medida que se move, vai notar que vai desenhando uma linha atrás dela. Quando a tartaruga acabar de se mover, o resultado será semelhante ao da figura seguinte.

Quando usar operações na Tartaruga, não é necessário chamar o método Show(). A Tartaruga ficará automaticamente visível quando alguma operação for executada.

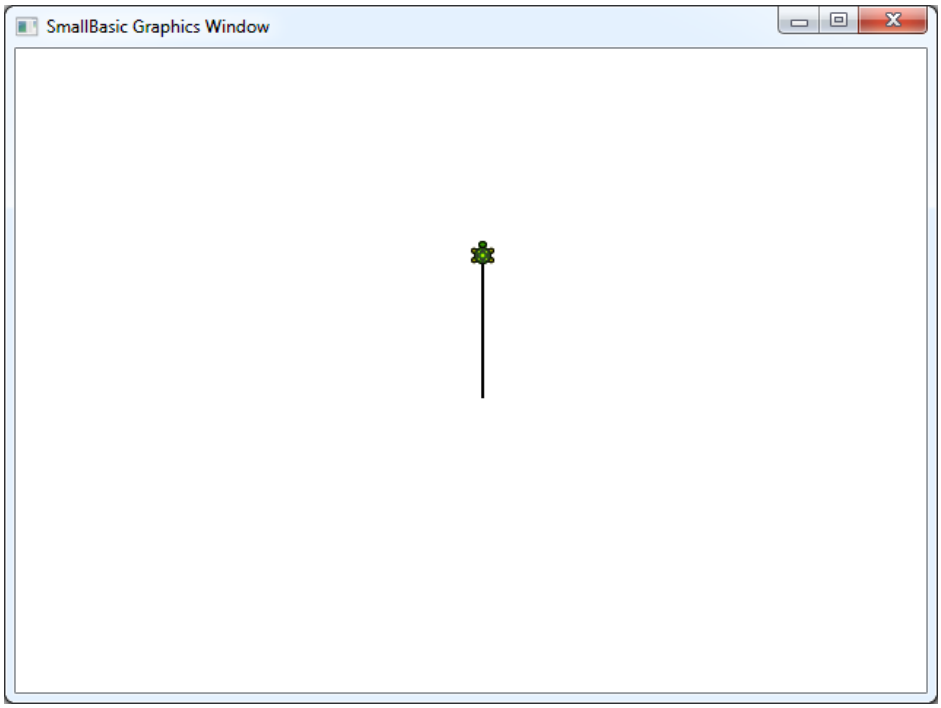


Figura 36 – Mover uma centena de pixéis

Desenhando um Quadrado

Um quadrado tem quatro lados, dois verticais e dois horizontais. Para desenharmos um quadrado nós precisamos de fazer a tartaruga desenhar uma linha, virar à direita e desenhar outra linha, e continuar isto até que os quatro lados estejam terminados. Se traduzirmos isto para o programa, é assim que que ficará:

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

Quando correr o programa consegue ver a tartaruga desenhar o quadrado, linha após linha, e o resultado será semelhante ao da seguinte figura:

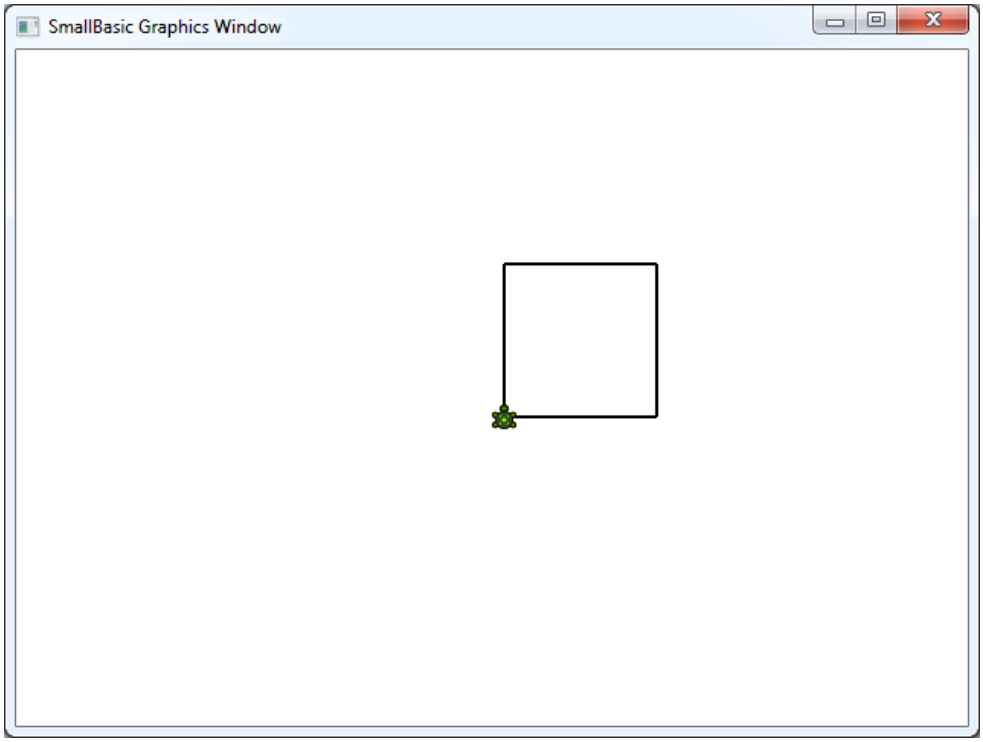


Figura 37 – Tartaruga desenhando o quadrado

Será interessante notar que estamos a utilizar as mesmas duas instruções várias vezes – quatro vezes para ser preciso. E já aprendemos que tais comandos repetitivos podem ser executados usando ciclos. Então, se pegarmos no programa acima e modifica-lo para usar um ciclo **For..EndFor**, acabamos por tornar o programa muito mais simples.

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

Mudando Cores

A tartaruga desenha na mesma janela gráfica (GraphicsWindow) que vimos no capítulo. Isto significa que todas as operações que aprendemos no capítulo anterior são válidas aqui. Por exemplo, o programa seguinte irá desenhar um quadrado com cada lado de uma cor diferente.

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

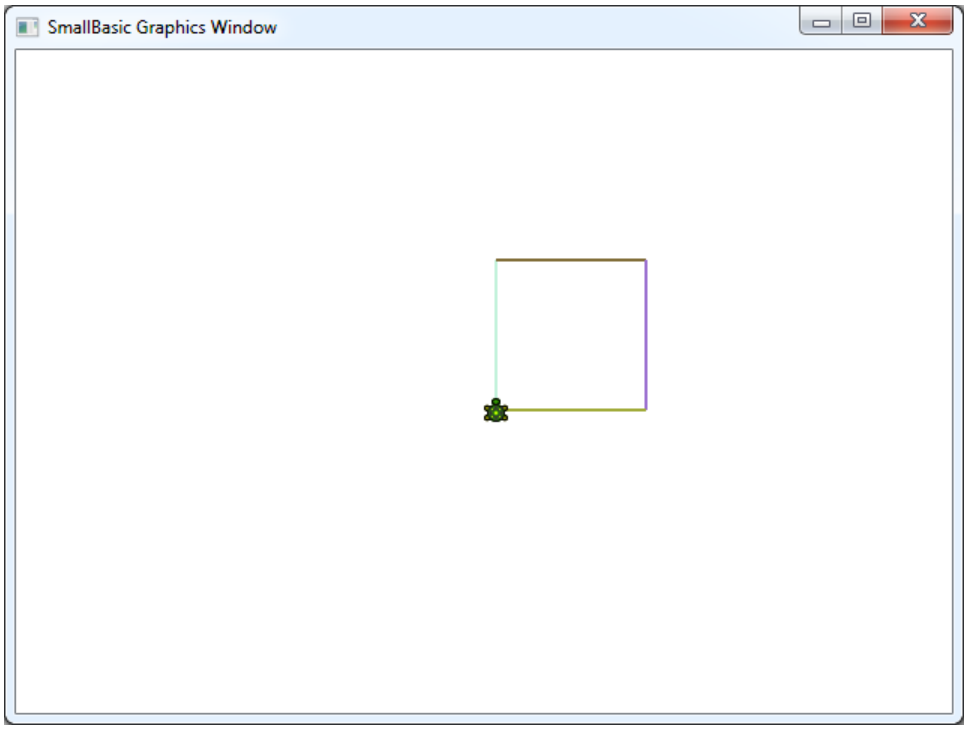


Figura 38 – Mudando Cores

Desenhando formas mais complexas

A Tartaruga, em adição às operações **TurnRight** (Virar Direita) e **TurnLeft** (Virar Esquerda), tem a operação **Turn** (Virar). Esta operação usa um parâmetro que especifica o ângulo da rotação. Usando esta operação, é possível desenhar qualquer polígono. O seguinte programa desenha um hexágono (um polígono de seis lados).

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

Tente este programa para ver se desenha realmente um hexágono. Observe que uma vez que o ângulo entre os lados é 60 graus, usamos **Turn(60)**. Para tal polígono, em que os lados são todos iguais, o ângulo entre todos os lados pode ser obtido dividindo 360 pelo número de lados. Com esta informação e usando variáveis, podemos escrever programa genéricos que podem desenhar qualquer tipo de polígonos.

```
lados = 12

comprimento = 400 / lados
```

```
angulo = 360 / lados

For i = 1 To lados
  Turtle.Move(comprimento)
  Turtle.Turn(angulo)
EndFor
```

Usando este programa, pode-se desenhar um polígono diferente apenas modificando a variável **lados**. Definindo 4 dar-nos-á um quadrado que começamos por fazer. Definindo um valor maior, por exemplo 50, fará quase um círculo perfeito.

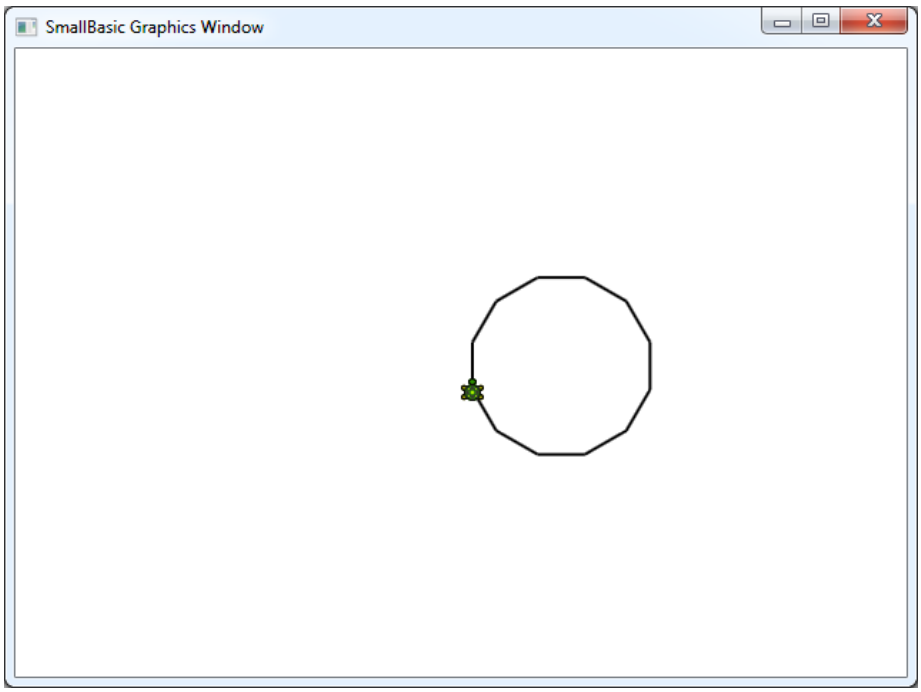


Figura 39 – Desenhando um polígono de 12 lados

Usando esta técnica que acabamos de aprender, podemos fazer a Tartaruga desenhar múltiplos círculos, com uma pequena mudança, tendo um resultado muito interessante.

```
lados = 50
comprimento = 400 / lados
angulo = 360 / lados

Turtle.Speed = 9

For j = 1 To 20
  For i = 1 To lados
    Turtle.Move(comprimento)
```

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

No programa acima, fizemos a Tartaruga avançar mais rapidamente ao alterar a velocidade para 9. Pode definir esta propriedade para qualquer valor entre 1 e 10 para fazer a Tartaruga andar com a velocidade que quiser que ela ande.

O programa acima tem dois ciclos **For..EndFor** , um dentro do outro. O ciclo interno (*i = 1 to lados*) é semelhante ao programa do polígono e é responsável por desenhar o círculo. O ciclo exterior (*j = 1 to 20*) é responsável por girar um pouco a Tartaruga por cada círculo que é desenhado. Isto diz à tartaruga para desenhar 20 círculos. Quando juntos, tornam o resultado deste programa num padrão muito interessante, como mostra a imagem seguinte.

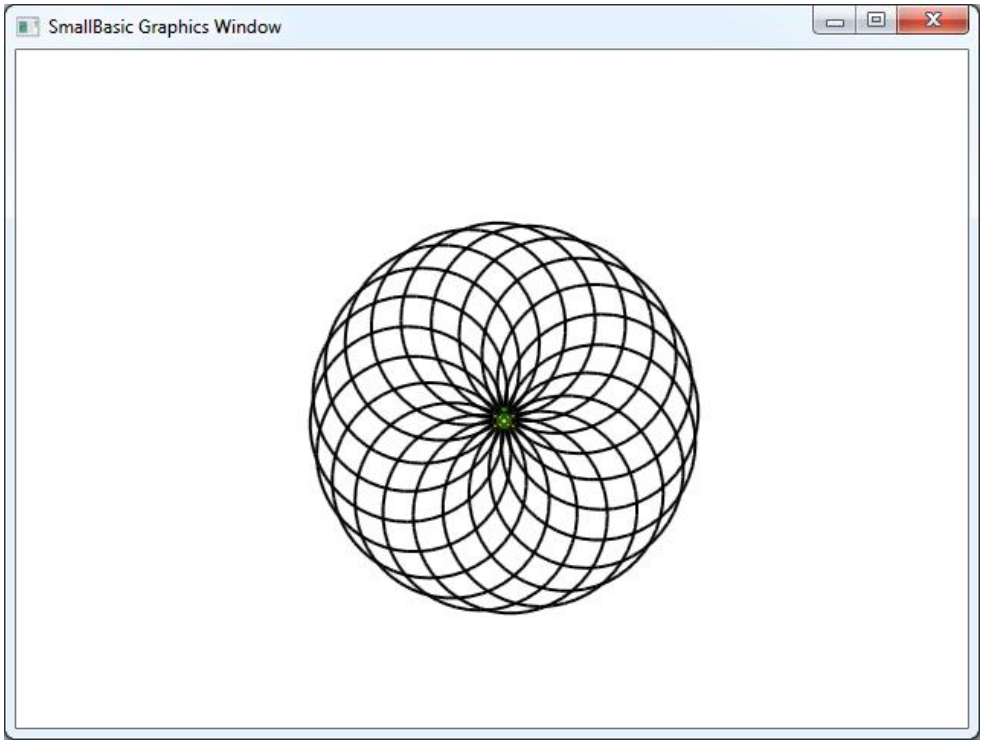


Figura 40 – Andando às voltas

Movimentar-se

Pode-se fazer a tartaruga não desenhar definido a operação **PenUp** (Caneta para Cima). Isto permite mover a tartaruga para qualquer lugar do ecrã sem desenhar uma linha. Chamar **PenDown** (Caneta para Baixo) irá fazer com que a tartaruga desenhe outra vez. Isto pode ser usado para se conseguir alguns efeitos interessantes, com por exemplo, linhas ponteadas. Aqui está um programa que usa isto para desenhar um polígono com linhas ponteadas.

```
lados = 6

comprimento = 400 / lados
angulo = 360 / lados

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

Novamente, este programa tem dois ciclos. O ciclo interior desenha uma linha única ponteada, enquanto o ciclo exterior expecifica quantas linhas desenhar. Neste exemplo, usamos 6 para a variável **lados** e portanto, conseguimos um hexágono com uma linha a ponteado, como a imagem em baixo.

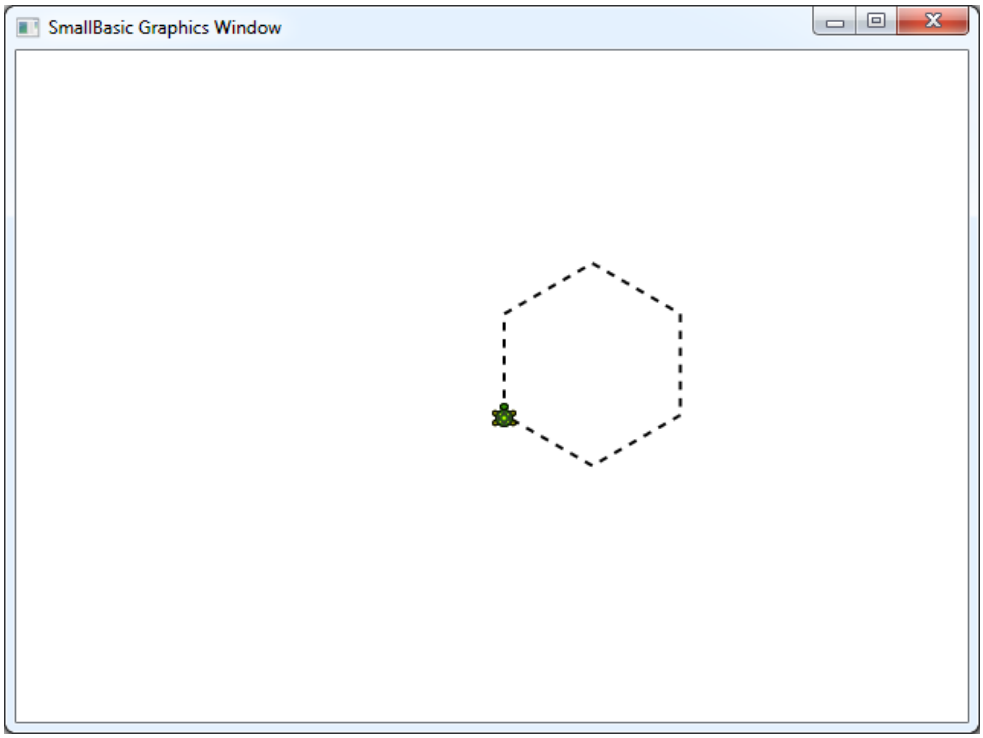


Figura 41 – Usando PenUp e PenDown

Capitulo 9

Sub-rotinas

Muito frequentemente, quando escrevemos programas, deparamo-nos com situações em que temos de executar os mesmos passos várias vezes. Nesses casos, provavelmente não faz sentido escrever o mesmo código várias vezes. É aí que as sub-rotinas são úteis.

Uma sub-rotina é um conjunto de código, num programa maior, que usualmente faz algo muito específico e que pode ser chamado em qualquer lado do programa. Sub-rotinas são identificadas pelo nome que se segue à *keyword* **Sub** e que são terminadas pela *keyword* **EndSub**. Por exemplo, o seguinte código representa a sub-rotina com o nome *ImprimeHora,* e imprime a hora actual em uma TextWindow.

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

O seguinte programa inclui uma sub-rotina que é chamada em vários locais.

```
ImprimeHora()
TextWindow.Write("Inserir o nome: ")
nome = TextWindow.Read()
TextWindow.Write(nome + ", a hora actual é: ")
ImprimeHora()

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

```
EndSub
```

Figura 42 – Chamar uma simples Sub-rotina

A sub-rotina é executada chamando *NomeDaSubRotina()*. Como usual, os parênteses "()" são necessários para dizer ao computador que se quer executar uma sub-rotina.

Vantagens de usar Sub-rotinas

Como já vimos anteriormente, as sub-rotinas ajudam a reduzir o tamanho do código que necessitamos de digitar. Uma vez que se tem a rotina *ImprimeHora* escrita, podemos chama-la em qualquer parte do programa e irá imprimir a hora actual.

Não esquecer que podemos chamar uma subrotina dentro do mesmo programa. Não podemos chamar uma subrotina a partir de outro programa.

Adicionalmente, as sub-rotinas podem ajudar a decompor problemas complexos em partes mais simples. Por exemplo, se temos uma equação complexa para resolver, pode-se escrever várias sub-rotinas que resolvem pequenas partes da equação complexa. Então pode-se juntar os resultados para voltar a ter a complexa equação original.

As sub-rotinas podem também ajudar a melhorar a legibilidade do programa. Por outras palavras, se tivermos sub-rotinas com bons nomes atribuídos para as diferentes partes do programa, o programa torna-se mais fácil de ler e mais compreensivo. Isto é muito importante se queremos compreender o programa ou se queremos que o nosso programa seja entendido por outros. Às vezes ajuda mesmo quando queremos ler o nosso próprio programa uma semana depois de o escrever.

Usando variáveis

Podemos aceder e usar as variáveis que temos no programa dentro das sub-rotinas. Como exemplo, o seguinte programa aceita dois números e imprime o maior dos dois. Reparem que a variável *max* é usada dentro e fora da sub-rotina.

```
TextWindow.Write("Inserir o primeiro número: ")
num1 = TextWindow.ReadNumber()
TextWindow.Write("Inserir o segundo número: ")
num2 = TextWindow.ReadNumber()
```

```
EncontraMaior()
TextWindow.WriteLine("O número maior é: " + max)

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

E o resultado do programa parece-se com isto:

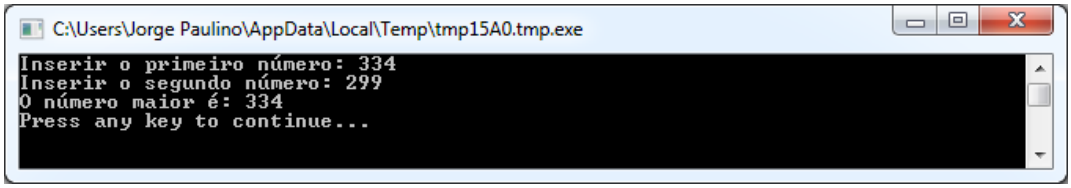


Figura 43 - Maior de dois números usando uma Sub-rotina

Vamos olhar para outro exemplo que irá ilustrar o uso de sub-rotinas. Desta vez iremos usar um programa gráfico que gera vários pontos que serão guardados nas variáveis *x* e *y*. Depois chama a sub-rotina **DrawCircleUsingCenter** que é responsável por desenhar o círculo usando *x* e *y* como o 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
```

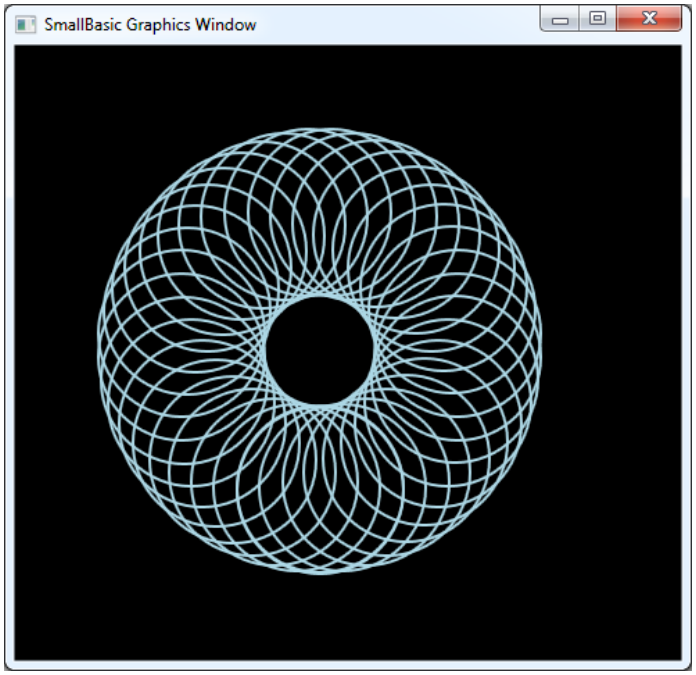


Figura 44 – Exemplo Gráfico usando uma Sub-rotina

Chamando sub-rotinas dentro dos ciclos

Às vezes as sub-rotinas são chamadas dentro de um ciclo, período durante o qual eles executam o mesmo conjunto de instruções mas com valores diferentes em uma ou mais variáveis. Por exemplo, imagine que tem uma sub-rotina chamada *PrimeCheck* e esta sub-rotina determina se um número é primo ou não. Pode-se escrever um programa que deixa o utilizador inserir um número e verifica se ele é primo ou não, usando esta sub-rotina. O seguinte programa ilustra isto.

```
TextWindow.Write("Inserir um número: ")
i = TextWindow.ReadNumber()
isPrime = "True"
PrimeCheck()
If (isPrime = "True") Then
  TextWindow.WriteLine(i + " é um número primo ")
Else
  TextWindow.WriteLine(i + " não é um número primo")
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
```

A sub-rotina *PrimeCheck* agarra no valor de *i* e tenta dividir por números mais pequenos. Se um número divide *i* e deixa um resto diferente de zero, então *i* não é um número primo. Nessa altura a sub-rotina define o valor da variável *isPrime* como "False" e sai. Se o número for indivisível por um valor mais pequeno então o valor de *isPrime* mantem-se como "True."

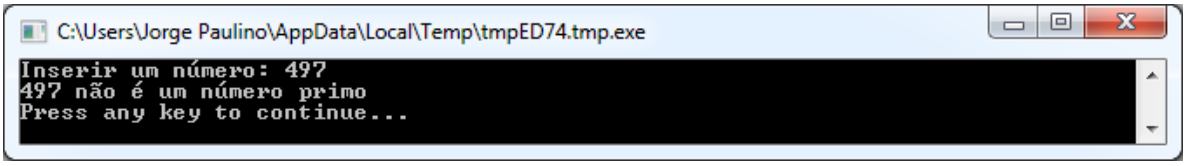


Figura 45 – Verificação de número Primos

Agora que já temos a sub-rotinaque pode testar os números Primos, podemos usa-la para listar todos os números primos abaixo de 100. É muito simples modificar o programa acima, de modo a fazer a chamada ao *PrimeCheck* dentro do ciclo. Isto dará à sub-rotina um valor diferente cada vez que o ciclo corre. Vamos ver como isto é feito com o exemplo seguinte.

```
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
```

Neste programa, o valor de *i* é actualizado cada vez que o ciclo corre. Dentro do ciclo, é feita uma chamada para a sub-rotina *PrimeCheck*. A sub-rotina *PrimeCheck* então agarra no valor de *i* e verifica se *i* é um número primo ou não.

O resultado é guardado na variável *isPrime* que é então acedida pelo ciclo fora da sub-rotina. O valor de *i* é então impresso se for um número primo. E como o ciclo começa em 3 e vai até 100, nós conseguimos a lista de todos os números primos entre 3 e 100. Este é o resultado do programa.

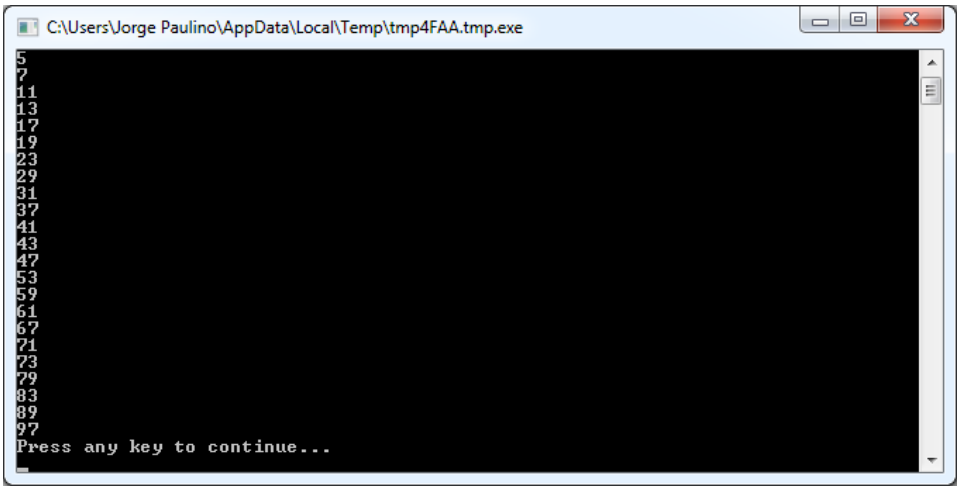


Figura 46 – Números Primos

Capitulo 10

Matrizes

Neste momento já deve estar bem treinado com o uso de variáveis – depois de tudo o que já se fez ainda se está a divertir, certo?

Vamos de momento voltar ao programa que escrevemos sobre variáveis:

```
TextWindow.Write("Inserir o Nome: ")
nome = TextWindow.Read()
TextWindow.WriteLine("Olá " + nome)
```

Neste programa, recebemos e guardamos o nome do utilizador numa variável chamada **nome**. Mais tarde dizemos "Olá" ao utilizador. Se existir mais do que um utilizador – por exemplo 5 utilizadores. Como podemos guardar os seus nomes? Uma forma de fazer isto é a seguinte:

```
TextWindow.Write("Utilizador1, inserir o nome: ")
nome1 = TextWindow.Read()
TextWindow.Write("Utilizador2, inserir o nome: ")
nome2 = TextWindow.Read()
TextWindow.Write("Utilizador3, inserir o nome: ")
nome3 = TextWindow.Read()
TextWindow.Write("Utilizador4, inserir o nome: ")
nome4 = TextWindow.Read()
TextWindow.Write("Utilizador5, inserir o nome: ")
nome5 = TextWindow.Read()
```

```
TextWindow.Write("Olá ")
TextWindow.Write(nome1 + ", ")
TextWindow.Write(nome2 + ", ")
TextWindow.Write(nome3 + ", ")
TextWindow.Write(nome4 + ", ")
TextWindow.WriteLine(nome5)
```

Quando corremos isto, iremos obter o seguinte resultado:

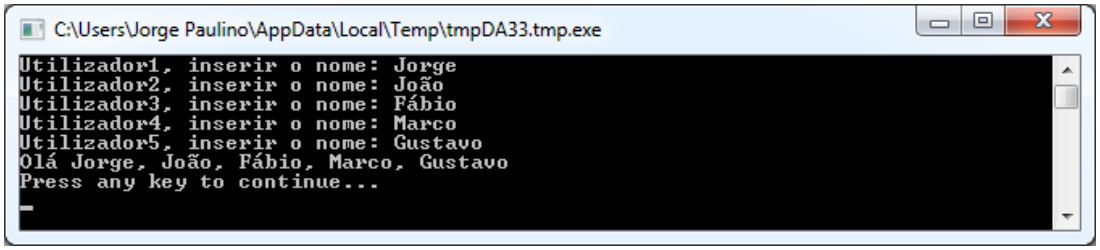


Figura 47 – Não usando matrizes

Certamente deve haver uma melhor forma de escrever este simples programa, certo? Especialmente quando o computador é muito bom em fazer tarefas repetitivas, então porque nos preocuparmos em escrever o mesmo código vezes e vezes sem conta para cada utilizador? O truque aqui é guardar e ler mais do que um nome de utilizador na mesma variável. Se podermos fazer isto então podemos usar um ciclo **For** que aprendemos nos capítulos anteriores. Aqui é onde as matrizes (Arrays) podem ajudar.

O que é uma matriz?

Uma matriz é uma variável especial que pode ter mais do que um valor. Basicamente o que significa é que em vez de criar **nome1, nome2, nome3, nome4** e **nome5** de modo a guardar os cinco nomes, podemos usar apenas o **nome** para guardar os cinco nomes dos utilizadores. A forma de guardar múltiplos valores é algo chamado "indice" (index). Por exemplo, **nome[1], nome[2], nome[3], nome[4]** e **nome[5]** podem guardar todos os valores. Os números 1, 2, 3, 4 e 5 são chamados *índices* da matriz.

Embora o **nome[1]**, **nome[2]**, **nome[3]**, **nome[4]** e **nome[5]** pareçam diferentes variáveis, eles são na realidade apenas uma variável. E qual é a vantagem disto, podemos perguntar. A melhor parte de utilizarmos variáveis em matrizes é que podemos especificar o índice usando outra variável, o que nos permite aceder às matrizes dentro de ciclos.

Agora vamos ver como podemo aplicar os nossos novos conhecimentos para reescrever o código anterior usando matrizes.

```
For i = 1 To 5
  TextWindow.Write("Utilizador" + i + ", inserir o nome: ")
  nome[i] = TextWindow.Read()
```

```
EndFor

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

Muito mais simples de ler, não é verdade? Repare nas linhas a negrito. A primeira guarda o valor na matriz e a segunda lê o valor da mesma. O valor que guardamos em **nome[1]** não irá afectar o que guardamos em **nome[2]**. Assim, para a maioria dos objectivos, pode-se tratar **nome[1]** e **nome[2]** como duas diferentes variáveis com a mesma identidade.

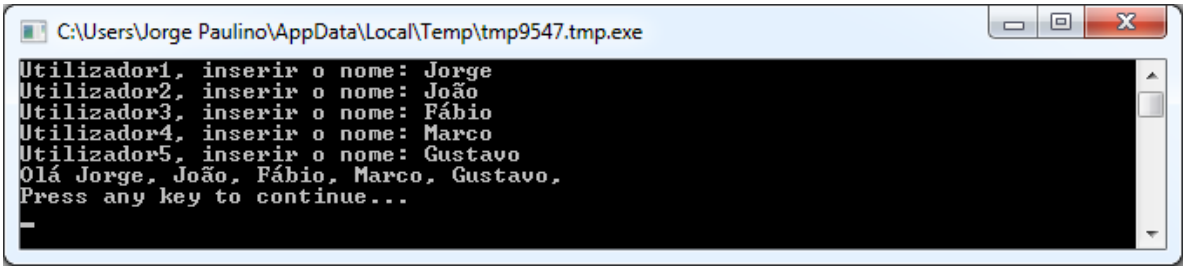


Figura 48 – Usando matrizes

O programa acima dá-nos quase o mesmo resultado do programa que não usa matrizes, excepto a vigula no final do nome *Mantis*. Podemos resolver isto reescrevendo o ciclo de impressão:

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

Indexando matrizes

No programa anterior podemos ver que foram usados números como índices para guardar e ler valores da matriz. Acontece que os índices não estão restritos a apenas números e na prática é muito útil utilizar índices textuais. Por exemplo, no programa seguinte, pedimos e armazenamos várias informações sobre um utilizador e, em seguida, imprimimos a informação que o utilizador solicita.

```
TextWindow.Write("Inserir o nome: ")
user["nome"] = TextWindow.Read()
TextWindow.Write("Inserir a idade: ")
user["idade"] = TextWindow.Read()
TextWindow.Write("Inserir a cidade: ")
user["cidade"] = TextWindow.Read()
TextWindow.Write("Inserir o código postal: ")
user["código postal"] = TextWindow.Read()

TextWindow.Write("Que informação quer? ")
index = TextWindow.Read()
TextWindow.WriteLine(index + " = " + user[index])
```

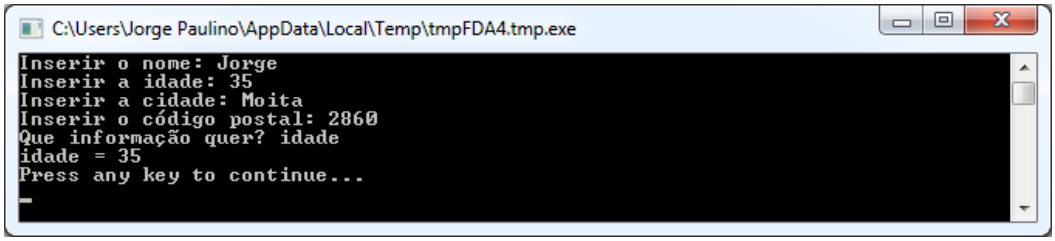


Figura 49 – Usando indices não numéricos

Mais do que uma dimensão

Digamos que você deseja armazenar o nome eo telefone de todos os seus amigos e, em seguida, ser capaz de pesquisa os seus números de telefone sempre que precisar - tipo uma agenda. Como iremos escrever um programa desse tipo?

Neste caso, existem dois conjuntos de índices (também conhecida como a dimensão da matriz) envolvidos. Suponha que identificam cada amigo pela sua alcunha. Isto torna-se o nosso primeiro índice da matriz. Uma vez que usamos o primeiro índice para obter a variável do nosso amigo, o segundo dos índices, **nome** e **número de telefone** irá ajudar-nos a chegar ao verdadeiro nome e número de telefone daquele amigo.

A forma como armazenar esses dados seria assim:

```
amigos["Rob"]["Nome"] =
"Robert"
amigos["Rob"]["Telefone"] =
"555-6789"

amigos["VJ"]["Nome"] = "Vijaye"
amigos["VJ"]["Telefone"] = "555-4567"
```

Índices de matriz não são caso sensível. Assim como as variáveis regulares, os índices das matrizes não têm que corresponder à capitalização.

```
amigos["Ash"]["Nome"] = "Ashley"
amigos["Ash"]["Telefone"] = "555-2345"
```

Como temos dois índices na mesma matriz, **amigos**, essa matriz é chamada de matriz bidimensional.

Uma vez definido o programa acima, então podemos tomar como entrada o apelido de um amigo e, em seguida, imprimir as informações que temos sobre eles armazenadas. Aqui está o programa completo que faz isso:

```
amigos["Rob"]["Nome"] = "Robert"
amigos["Rob"]["Telefone"] = "555-6789"

amigos["VJ"]["Nome"] = "Vijaye"
amigos["VJ"]["Telefone"] = "555-4567"

amigos["Ash"]["Nome"] = "Ashley"
amigos["Ash"]["Telefone"] = "555-2345"

TextWindow.Write("Inserir a alcunha: ")
nickname = TextWindow.Read()

TextWindow.WriteLine("Nome: " + amigos[nickname]["Nome"])
TextWindow.WriteLine("Telefone: " + amigos[nickname]["Telefone"])
```

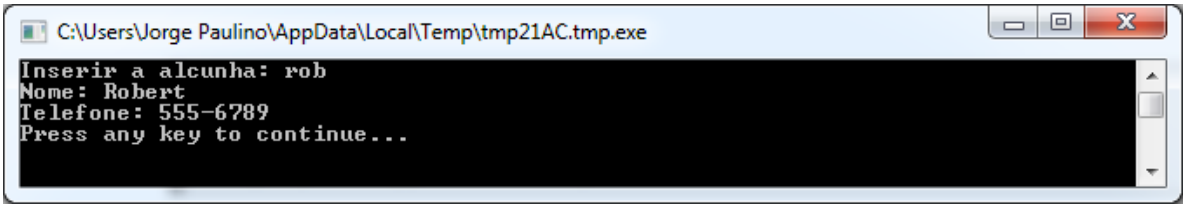


Figura 50 – Uma simples agenda telefónica

Usando matrizes para representar grelhas

Um uso muito comum de matrizes multi-dimensionais é representar grelhas/tabelas. As grelhas têm linhas e colunas, que se podem encaixar perfeitamente numa matriz bidimensional. Um programa simples que estabelece as caixas em uma grelha é mostrado de seguida:

```
linhas = 8
colunas = 8
tamanho = 40
```

```
For r = 1 To linhas
  For c = 1 To colunas
    GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
    boxes[r][c] = Shapes.AddRectangle(tamanho, tamanho)
    Shapes.Move(boxes[r][c], c * tamanho, r * tamanho)
  EndFor
EndFor
```

Este programa adiciona rectângulos e posiciona-os de formar a elaborar uma grelha de 8x8. Além de alinhar estas caixas, também armazena-as numa matriz. Isso torna fácil de controlar as caixas e usá-las novamente quando precisarmos delas.

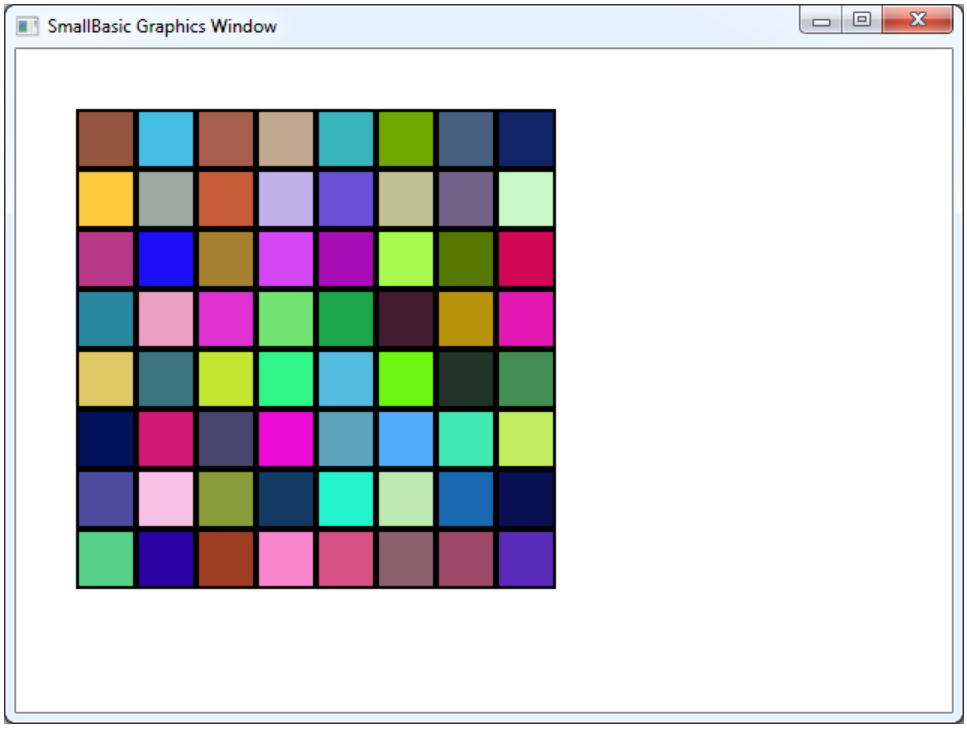


Figura 51 – Criando uma grelha de caixas

Por exemplo, adicionando o seguinte código ao final do programa anterior faria essas caixas animar para o canto superior esquerdo.

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

```
  EndFor
EndFor
```

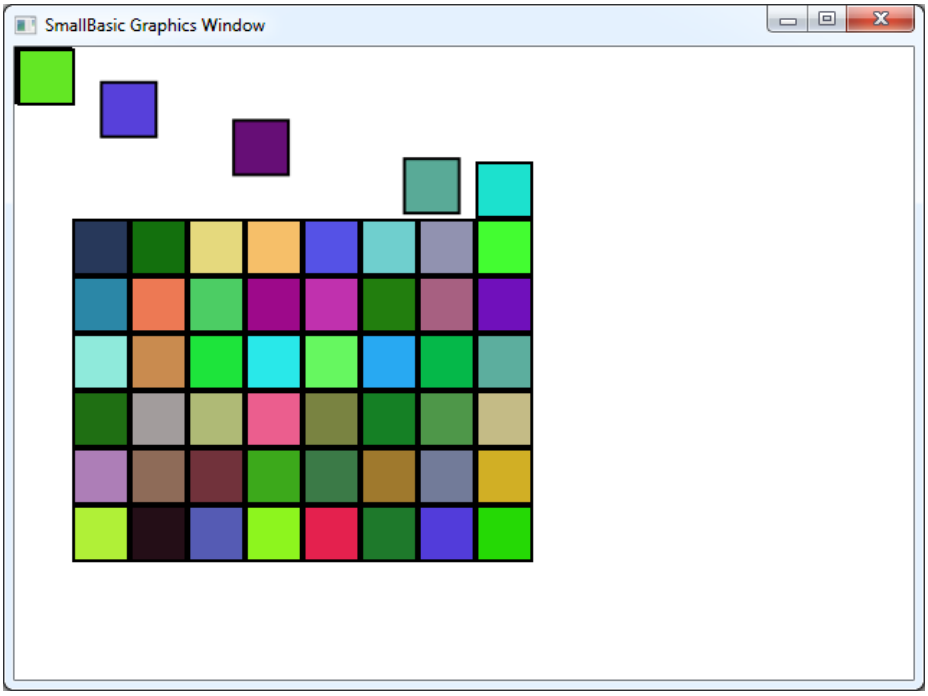


Figura 52 - Manter o controle de caixas na grelha

Capitulo 11

Eventos e Interatividade

Nos dois primeiros capítulos, foram apresentados objectos que têm propriedades e operações. Além das propriedades e operações, alguns objectos têm o que são chamados de Eventos. Os eventos são como os sinais que são gerados, por exemplo, em resposta às acções do utilizador, como mover o rato ou clicar nele. De uma certa forma os eventos são o oposto das operações. No caso das operações, como programador, chama-as para fazer o computador executar alguma coisa, e, no caso de eventos, o computador avisa quando algo interessante aconteceu.

Como são os eventos úteis?

Os eventos são centrais na introdução para a interatividade de um programa. Se quiser permitir que um utilizador interaja com o programa, os eventos são o que terá de usar. Digamos que está a escrever um jogo do galo. Você vai querer permitir que o utilizador escolha o jogo, certo? É aí que entram em eventos - que receber a entrada do utilizador dentro de seu programa usando eventos. Se isso parece difícil de entender, não se preocupe, vamos ver um exemplo muito simples que ajudará a entender quais são os eventos e como eles podem ser usados.

Abaixo está um programa muito simples que tem apenas uma declaração e uma sub-rotina. A sub-rotina usa a operação *ShowMessage* sobre o objeto GraphicsWindow para exibir uma caixa de mensagem ao utilizador.

```
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  GraphicsWindow.ShowMessage("Você clicou.", "Olá")
EndSub
```

A parte interessante de ver no programa acima é a linha onde vamos atribuir o nome de sub-rotina para o evento **MouseDown** do objecto GraphicsWindow. Você notará que MouseDown se parece muito com uma propriedade - só que em vez de atribuir algum valor, estamos a atribuir-lhe a sub-rotina *OnMouseDown*. Isso é especial sobre os acontecimentos - quando o evento acontece, a sub-rotina é chamada automaticamente. Neste caso, a sub-rotina *OnMouseDown* é chamada cada vez que o utilizador clica com o rato, sobre a GraphicsWindow. Execute o programa e teste-o. Quando você clicar no GraphicsWindow com o rato, você verá uma mensagem como a mostrada na imagem abaixo.

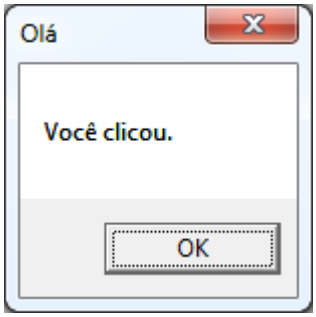


Figura 53 – Resposta a um evento

Esse tipo de manipulação de eventos é muito poderoso e permite programas muito criativos e interessantes. Programas escritos desta forma são chamados programas orientados a eventos.

Você pode modificar a sub-rotina *OnMouseDown* para fazer outras coisas além de aparecer uma mensagem. Por exemplo, como no programa abaixo, você pode desenhar grandes pontos azuis quando o utilizador clica com o rato.

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

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

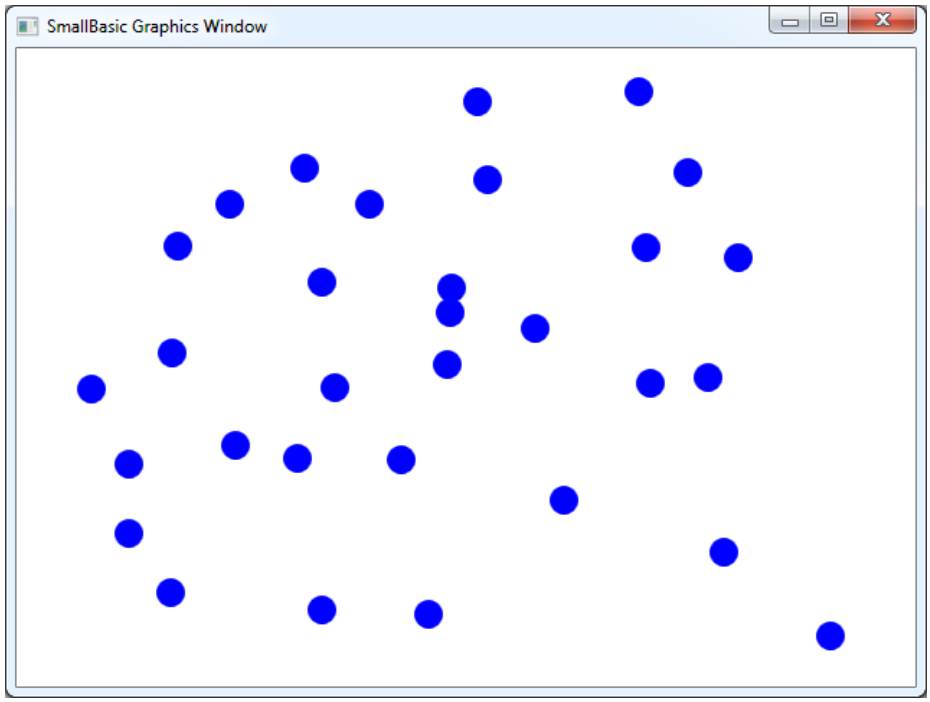


Figura 54 – Manipulando o evento MouseDown

Observe que no programa acima, nós utilizamos *MouseX* e *MouseY* para obter as coordenadas do rato. Em seguida, usamos isso para desenhar um círculo usando as coordenadas do rato como o centro do círculo.

Manipulação de vários eventos

Realmente não há limites para quantos eventos pode manipular. Pode até ter uma sub-rotina para lidar com vários eventos. No entanto, você pode manipular um evento apenas uma vez. Se tentar atribuir duas sub-rotinas para o mesmo evento, a segunda ganha.

Para ilustrar isto, vamos ver o exemplo anterior e adicionar uma sub-rotina que vai processa as teclas pressionadas. Além disso, vamos fazer essa nova sub-rotina alterar a cor do pincel, para que quando se clique com o rato, apareça um ponto de cor diferente.

```
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
```

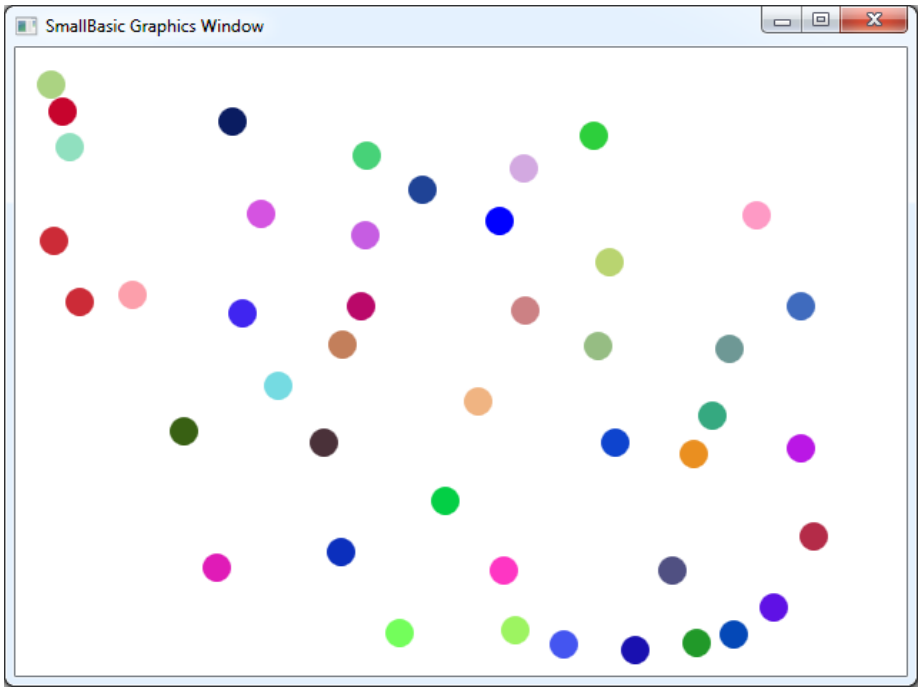


Figura 55 – Manipulando múltiplos eventos

Se executou o programa e clicou na janela, você vai ter um ponto azul. Agora, se você pressionar qualquer tecla e clicar novamente, vai ter um ponto com uma cor diferente. O que acontece, quando pressiona uma tecla, é que a sub-rotina que é executada, OnKeyDown, muda a cor do pincel para uma cor aleatória. Depois disso, quando clicar com o rato, é desenhado um círculo com a cor recentement definida - dando os pontos de cor aleatória.

Um programa de desenho

Através de eventos e sub-rotinas, podemos agora escrever um programa que permite aos utilizadores desenhar na janela. É surpreendentemente fácil escrever um programa destes, dividindo o mesmo em pequenas partes. Como primeiro passo, vamos escrever um programa que permitirá aos utilizadores mover o rato em qualquer lugar da janela gráfica, deixando um rasto onde quer que se mova o rato.

```
GraphicsWindow.MouseMove = OnMouseMove

Sub OnMouseMove
  x = GraphicsWindow.MouseX
```

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

No entanto, quando executar este programa, a primeira linha começa sempre a partir do canto superior esquerdo da janela (0, 0). Podemos corrigir este problema ao manipular o evento *MouseDown* e capturar os valores *prevX* e *prevY* quando esse evento acontece.

Além disso, nós só precisamos do rasto quando o utilizador tem o botão do rato em baixo. Outras vezes, não devemos traçar a linha. Para obter este comportamento, vamos usar a propriedade *IsLeftButtonDown* do objecto **Mouse**. Esta propriedade diz se o botão esquerdo está pressionado ou não. Se esse valor for verdadeiro, então nós vamos desenhar a linha, se não vamos saltar a linha.

```
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
```

Anexo A

Exemplos Divertidos

Tartaruga Fractal

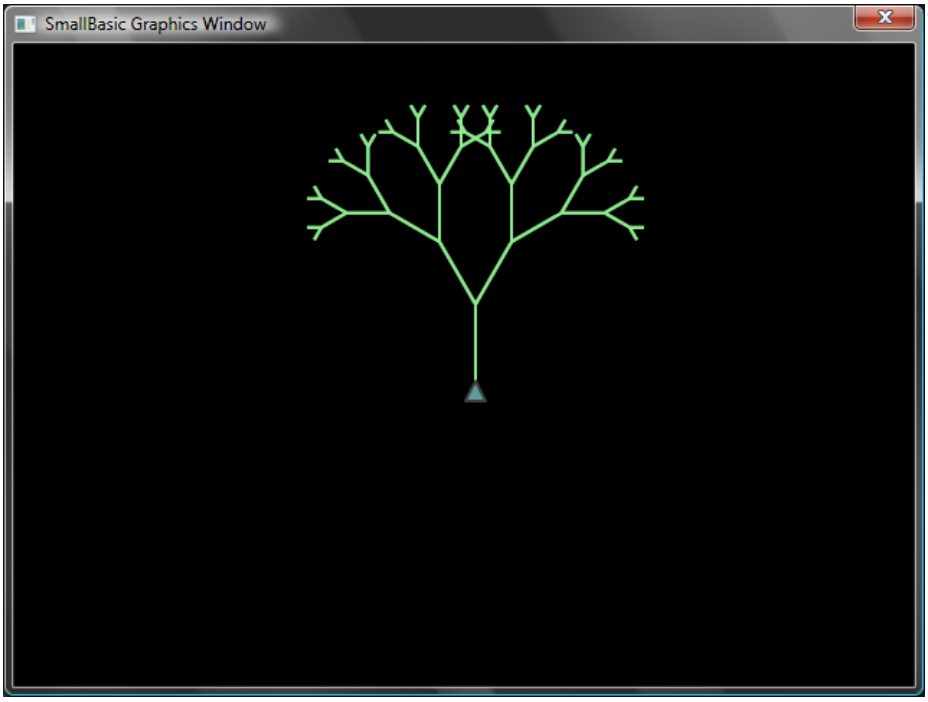


Figura 56 - Tartaruga desenho de uma árvore fractal

```
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
```

Fotos do Flickr

Figura 57 – Usando imagens do Flickr

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

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

Papéis de Parede Dinâmicos

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

Jogo do Remo

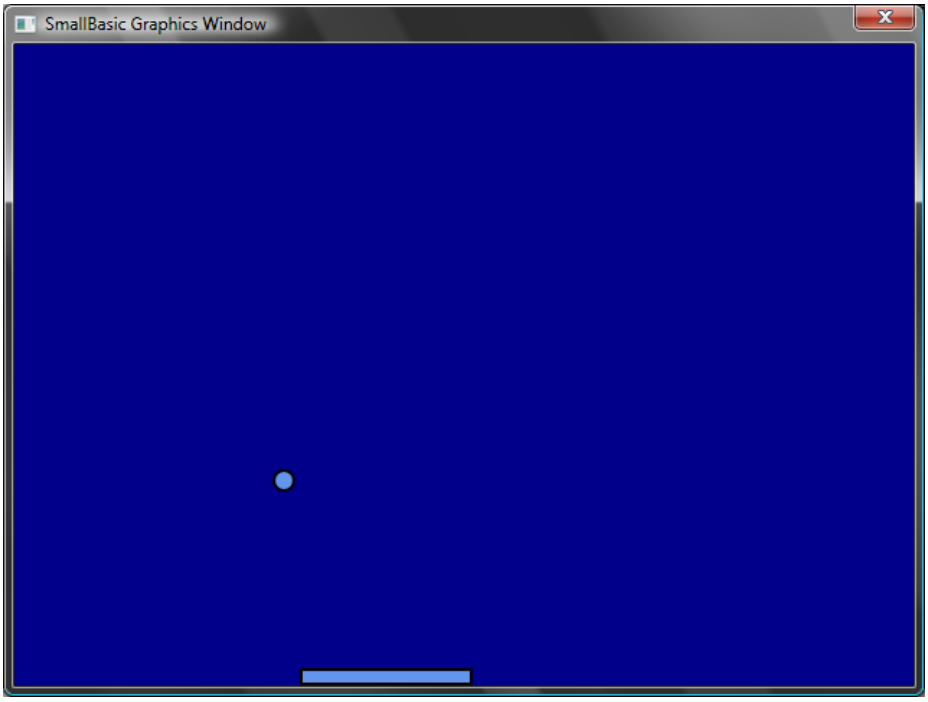


Figura 58 – Jogo do Remo

```
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
```

Anexo B

Cores

Aqui está uma lista de cores suportadas pelo Small Basic, agrupadas pela sua cor base.

Vermelhos

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

Cor-de-rosa

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

Cores de Laranja

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

Amarelos

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

PaleGoldenrod	#EEE8AA
Khaki	#F0E68C
DarkKhaki	#BDB76B

Púrpura

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

Azuis

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

Castanhos

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

Brancos

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

Cinzentos

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