Microsoft Small Basic

A la découverte de la programmation

Avant de commencer

Le mot des traducteurs

Ce document d'une soixantaine de pages est peut-être l'un de vos premiers pas dans le monde du développement logiciel. Vous devrez certainement lire d'autres ouvrages et écrire un certain nombre de programmes (petits et grands) pour vous sentir complètement à l'aise avec ce nouveau domaine. Arrivé à la fin de ce document, vous aurez réalisé des programmes intéressants, tel que un jeu de casse-briques ou bien encore un programme changeant aléatoirement votre fond d'écran.

SmallBasic – A la découverte de la programmation est le titre original de ce document. Il a été traduit en français par une équipe de MSP – Microsoft Student Partners – français : Jean-François Boulière, Sylvain Bruyère, Sébastien Courtois, Thomas De Toledo, Pierre Lasvigne et Christopher Maneu.

Nous espérons que la lecture de ce document vous donnera envie de continuer votre découverte du développement et de vous initier à de nouveaux langages.

Bienvenue dans le merveilleux monde de la programmation !
L'équipe des traducteurs

Chapitre 1

Introduction

Small Basic et la programmation logicielle

La programmation logicielle est définie comme le processus de création de logiciel informatique en utilisant des langages de programmation. De même que nous parlons et comprenons l'anglais, l'espagnol ou le français, les ordinateurs peuvent comprendre des programmes écrits dans certains langages. Ces derniers sont appelés langages de programmation. Au commencement, il n'y avait que quelques langages de programmation et ils étaient vraiment faciles à apprendre et à comprendre. Mais au fur et à mesure que les ordinateurs et les logiciels devinrent de plus en plus sophistiqués, les langages évoluèrent rapidement, rassemblant davantage de concepts complexes au fur et à mesure de cette évolution. Il en résulte que la plupart des langages modernes de programmation et leurs concepts sont assez difficiles à appréhender par un débutant. Cet état de fait a commencé à décourager les gens d'apprendre ou de s'essayer à la programmation logicielle.

Small Basic est un langage de programmation conçu pour rendre la programmation extrêmement simple d'approche et amusante pour les débutants. L'intention de Small Basic est de lever les barrières et d'apporter une pierre angulaire à l'extraordinaire monde de la programmation logicielle.

L'environnement Small Basic

Commençons avec une rapide introduction à l'environnement de Small Basic. Quand vous lancerez pour la première fois SmallBasic, vous verrez une fenêtre ressemblant à la figure suivante.

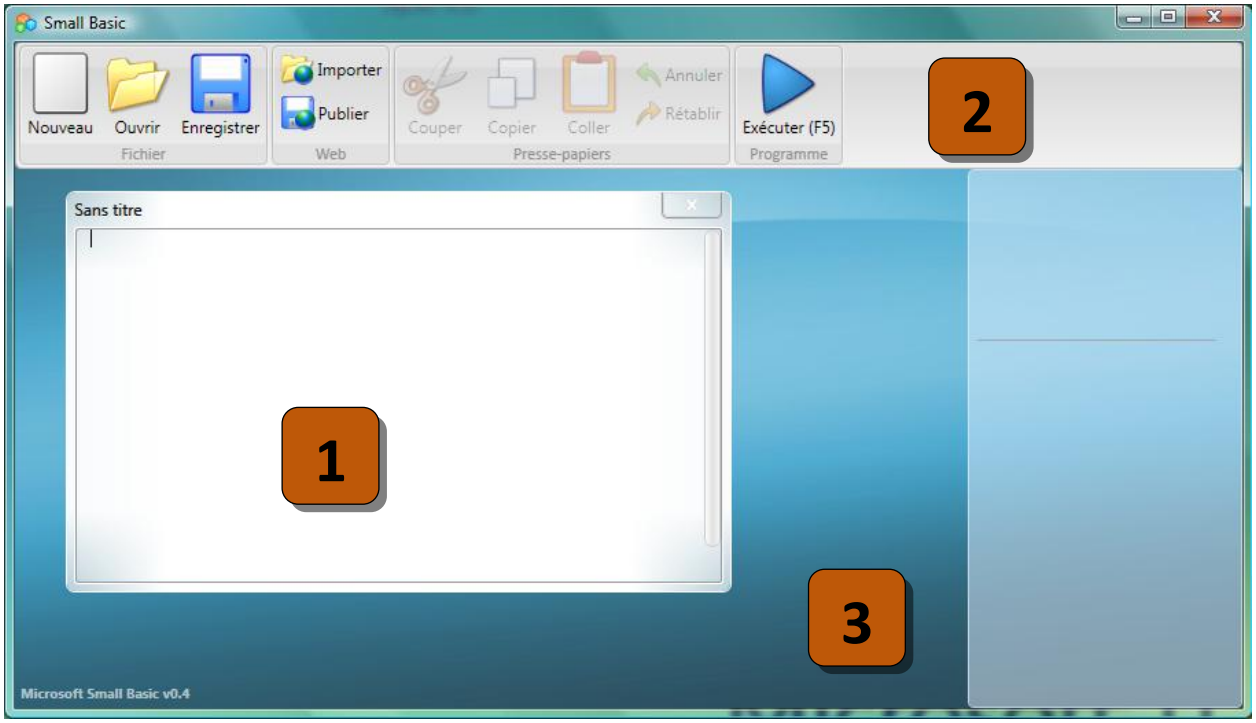

Figure 1 – L'environnement Small Basic

Ceci est l'environnement Small Basic, où on va écrire et exécuter nos programmes Small Basic. Cet environnement a plusieurs éléments distincts identifiés par des numéros.

L'**Editeur**, identifié par [1], est l'endroit où on écrit les programmes Small Basic. Quand vous ouvrez un programme sauvegardé précédemment, il apparaîtra dans cet éditeur. Vous pouvez ensuite modifier le programme et le sauvegarder pour un usage ultérieur.

Vous pouvez aussi ouvrir et travailler sur différents programmes simultanément. Chaque programme sur lequel vous travaillez sera affiché dans un éditeur différent. L'éditeur qui contient le programme sur lequel vous êtes actuellement en train de travailler est appelé *éditeur actif*.

La **Barre d'outils**, identifiée par [2], est utilisée pour commander à la fois l'éditeur actif et l'environnement. Nous ferons connaissance avec les nombreuses commandes au fur et à mesure de ce guide.

La **Surface**, identifiée par [3], est l'endroit où se trouvent toutes les fenêtres d'éditeur.

Notre premier programme

Maintenant que vous êtes familier avec l'environnement Small Basic, nous allons commencer à programmer. Comme nous l'avons remarqué plus haut, l'éditeur est l'endroit où on écrit les programmes. Alors continuons et tapons la ligne suivante dans l'éditeur.

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

C'est notre premier programme en Small Basic et si vous l'avez tapé correctement, vous devriez obtenir quelque chose de similaire à la figure ci-dessous.

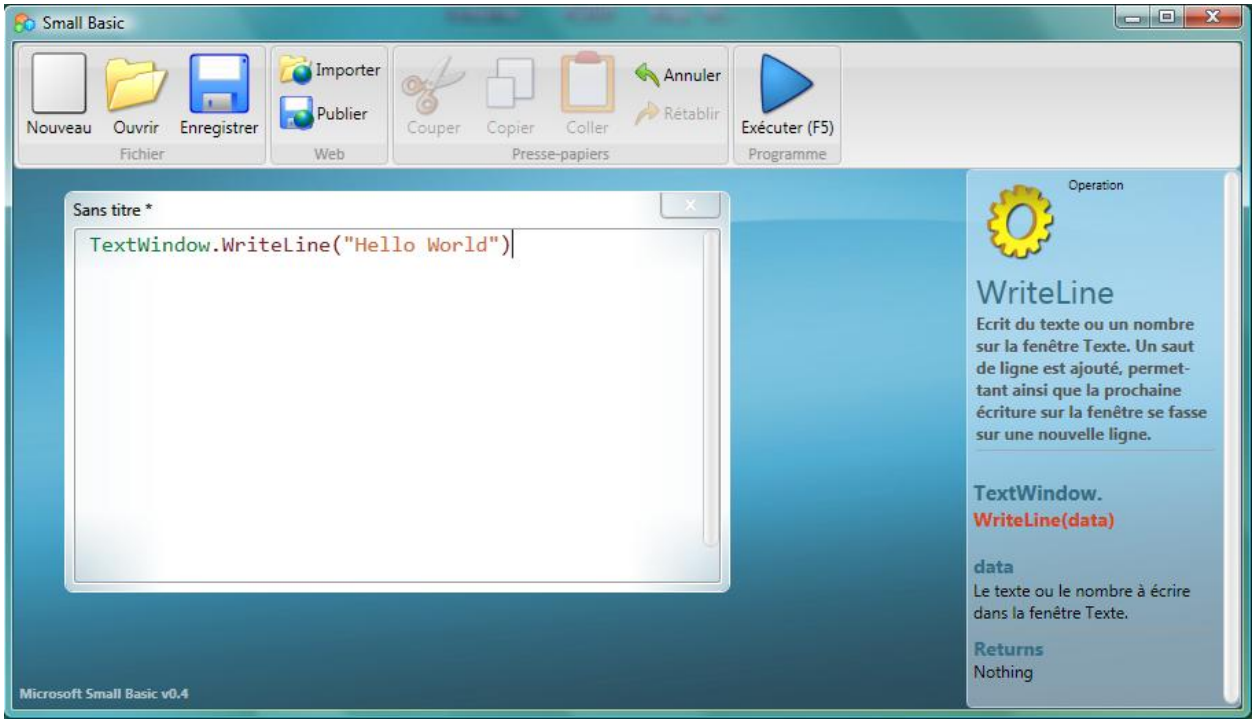

Figure 2 – Premier programme

Maintenant que nous avons entré notre nouveau programme, continuons et exécutons-le pour voir ce qui se passe. Nous pouvons exécuter notre programme soit en cliquant sur le bouton *Exécuter* de la barre d'outils soit en utilisant le raccourci clavier F5. Si tout se passe bien, notre programme devrait s'exécuter en donnant le résultat ci-dessous.

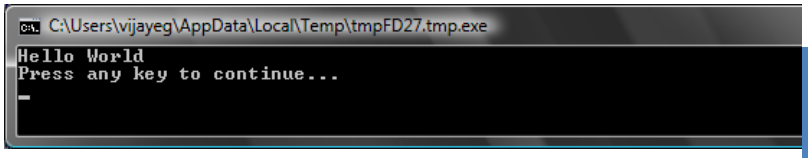

Figure 3 – Sortie de notre premier programme

Félicitations ! Vous venez d'écrire et d'exécuter votre premier programme en Small Basic. Un très petit et très simple programme, mais néanmoins un grand pas en avant dans la démarche de devenir un vrai programmeur ! Maintenant, il y a juste un détail supplémentaire à connaître avant de continuer à créer de plus gros programmes. Nous devons comprendre ce qui

*Quand vous avez tapé votre premier programme, vous avez pu remarquer qu'une fenêtre est apparue avec une liste d'objets (*Figure 4*). Elle est appelé "IntelliSense" et vous aide à taper plus vite votre programme. Vous pouvez parcourir cette liste en utilisant les flèches Haut/Bas de votre clavier, et quand vous trouvez quelque chose qui vous intéresse, vous pouvez presser la touche Entrée pour insérer l'élément sélectionné dans votre programme.*

s’est produit – ce que nous avons dit exactement à l’ordinateur et comment a-t-il su quoi faire ? Dans le chapitre suivant, nous analyserons le programme que nous venons d’écrire pour acquérir cette compréhension.

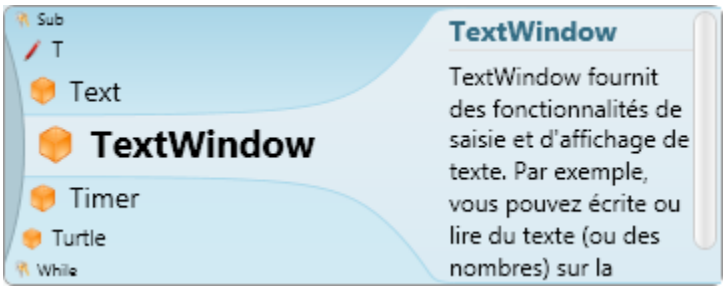

Figure 4 - IntelliSense

Sauvegardons notre programme

Si vous voulez fermer Small Basic et y revenir plus tard pour travailler sur le programme que vous venez de taper, vous devez enregistrer le programme. C’est en réalité une bonne pratique de sauvegarder le programme de temps en temps, afin de ne pas perdre d’information dans l’éventualité d’une extinction accidentelle de l’ordinateur ou d’une panne de courant. Vous pouvez sauvegarder votre programme actuel soit en cliquant sur l’icône *Enregistrer* de la barre d’outils soit en utilisant le raccourci clavier “Ctrl+S” (il vous suffit d’appuyer sur la touchée S pendant que vous appuyez sur la touche Ctrl).

Chapitre 2

Comprendre notre premier programme

Qu'est réellement un programme d'ordinateur?

Un programme est un ensemble d'instructions pour l'ordinateur. Ces instructions indiquent précisément à l'ordinateur ce qu'il doit faire, et l'ordinateur suit toujours ces instructions. Comme les êtres humains, les ordinateurs ne peuvent suivre les instructions que si elles sont spécifiées dans un langage qu'ils peuvent comprendre. Ce sont les langages de programmation. Il y a de nombreux langages de programmation que l'ordinateur peut comprendre et **Small Basic** est l'un deux.

Imaginez une conversation entre vous et votre ami. Vous utiliseriez des mots, organisés en phrases pour véhiculer des informations entrantes et sortantes. De la même façon, les langages de programmation contiennent des bibliothèques de mots qui peuvent être organisées en phrases qui véhiculent les informations vers l'ordinateur. Les programmes sont simplement des ensembles de phrases (parfois juste quelques unes et parfois plusieurs milliers) qui réunies acquièrent un sens à la fois pour le programmeur et l'ordinateur.

Il y a de nombreux langages informatiques modernes tels que Java, C++, Python, VB, etc., qui sont utilisés pour développer des programmes simples ou complexes, compréhensibles par l'ordinateur.

Les programmes en Small Basic

Un programme en Small Basic typique consiste en un ensemble de *commandes*. Chaque ligne du programme représente une *commande* et chaque *commande* correspond à un ensemble d'instructions exécutables par l'ordinateur. Quand on demande à l'ordinateur d'exécuter un programme en Small Basic, il prend le programme et lit la première commande. Il comprend ce que nous essayons de dire et exécute nos commandes. Une fois qu'il a exécuté notre première commande, il revient au programme, lit et exécute la seconde ligne. Il continue ainsi jusqu'à ce qu'il atteigne la fin du programme. C'est là que se termine notre programme.

Retour à notre premier programme

Voici le premier programme que nous avons écrit :

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

C'est un programme très simple constitué d'une seule commande. Cette commande dit à l'ordinateur d'écrire une ligne de texte qui est **Hello World** dans la fenêtre texte (*Text Window*), appelée aussi Console.

Littéralement, cela se traduit dans l'esprit de l'ordinateur par :

```
Ecrire Hello World
```

Vous pourriez avoir déjà remarqué que la commande peut se découper en plusieurs segments de taille inférieure de la même façon que la phrase peut se découper en mots. Dans la première commande, nous avons 3 segments distincts :

a) La Console (*Text Window*)
b) La commande WriteLine
c) Le texte "Hello World"

Les points, parenthèses et guillemets sont tous des ponctuations qui doivent être placées au bon endroit dans votre code pour que l'ordinateur comprenne notre intention.

Vous vous rappelez peut-être l'écran noir qui est apparu lorsque vous avez exécuté votre premier programme. Cette fenêtre est appelée *Text Window* mais on s'y réfère généralement comme la Console. C'est là que s'affiche le résultat du programme. La console, dans notre exemple, est en réalité un *objet*. Nous pouvons utiliser un certain nombre d'objets dans nos programmes. On peut réaliser plusieurs opérations différentes sur ces objets. Nous avons déjà utilisé la commande *WriteLine* dans notre programme. Vous avez aussi pu remarquer que la commande *WriteLine* est suivie par **Hello World** dans des guillemets. Ce texte est traité comme une *entrée* par la commande *WriteLine*, et est ensuite affiché pour l'utilisateur. Il est appelé *argument* de la commande (ou paramètre). Certaines commandes prennent un ou plusieurs arguments alors que certaines n'en prennent aucun.

> *Les éléments de ponctuation comme les guillemets, les espaces et les parenthèses sont très importants dans un programme informatique. En fonction de leur position et de leur nombre, ils peuvent changer la signification de ce qui est exprimé.*

Notre deuxième programme

Maintenant que vous avez compris notre premier programme, avançons dans la leçon et ajoutons quelques couleurs pour le rendre plus attrayant.

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

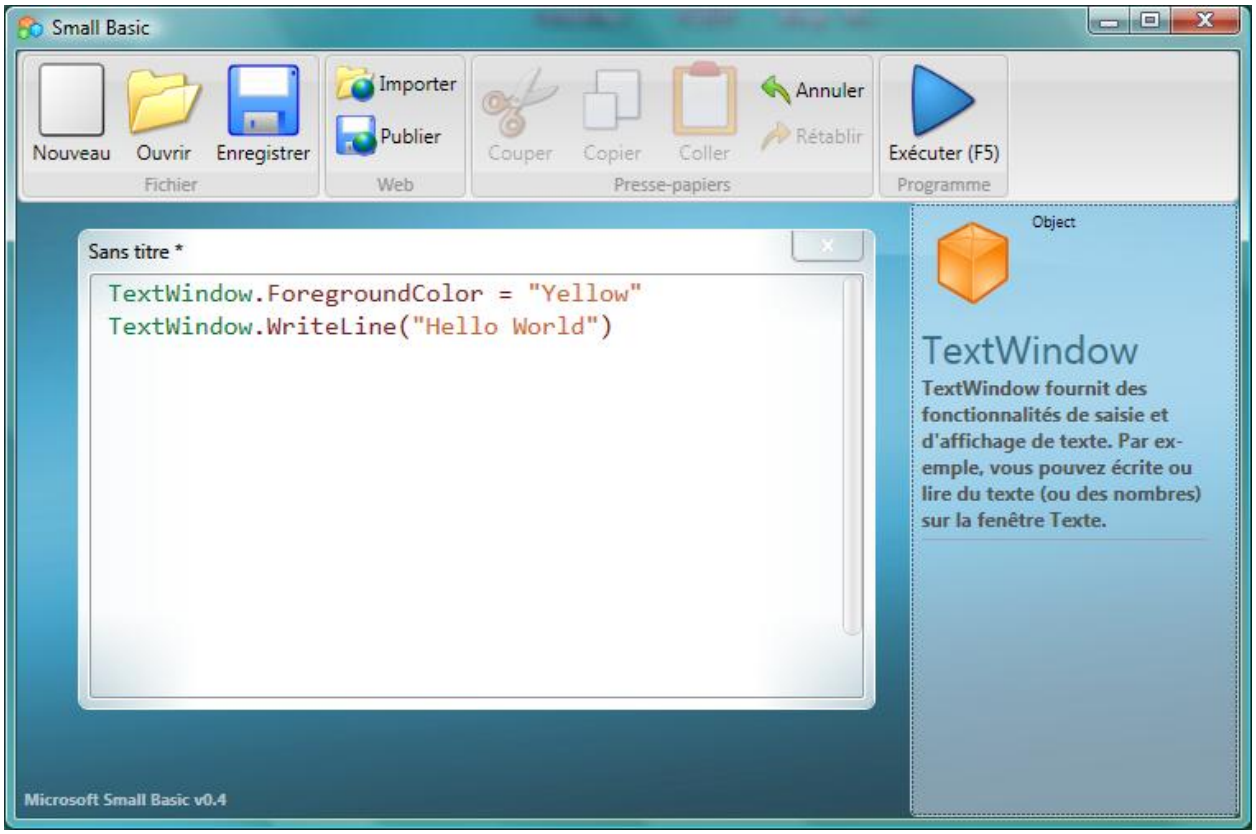

Figure 5 – Ajouter des couleurs

Quand vous exécuterez le programme ci-dessus, vous pourrez remarquer qu'il écrit le même texte "Hello World" dans la Fenêtre de texte, mais cette fois-ci le texte est écrit en jaune au lieu du gris du programme précédent.

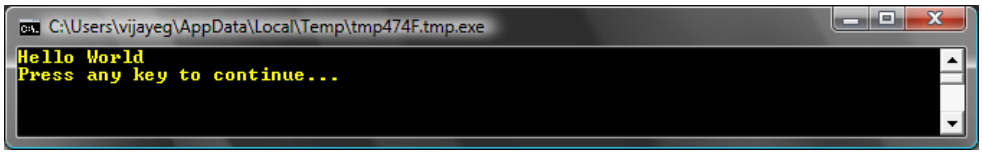

Figure 6 - Hello World en jaune

Remarquez la nouvelle commande que nous avons ajoutée à notre programme de départ. Elle utilise un nouveau mot, *ForegroundColor* (couleur du premier plan) que nous avons défini à la valeur "Yellow"

(jaune). Cela signifie que nous avons assigné “Yellow” au paramètre *ForegroundColor*. Maintenant, la différence entre *ForegroundColor* et la commande *WriteLine* est que *ForegroundColor* ne prend pas d’argument et ne nécessite pas de parenthèses, alors qu’il est suivi d’un signe *égal* et d’un mot. On définit *ForegroundColor* comme une *propriété* de la console (*Text Window*). Voici une liste des valeurs pour le paramètre *ForegroundColor*. Essayez de remplacer “Yellow” avec l’une de ces valeurs et observez le résultat – n’oubliez-pas les guillemets, ils font partie de la ponctuation nécessaire. De plus vous devez impérativement utiliser les noms des couleurs en anglais dans votre programme.

```
Mot clé                 -> Signification
Black                   -> Noir
Blue                    -> Bleu
Cyan                    -> Cyan
Gray                    -> Gris
Green                   -> Vert
Magenta                 -> Magenta
Red                     -> Rouge
White                   -> Blanc
Yellow                  -> Jaune
DarkBlue                -> Bleu foncé
DarkCyan                -> Cyan foncé
DarkGray                -> Gris foncé
DarkGreen               -> Vert foncé
DarkMagenta             -> Magenta foncé
DarkRed                 -> Rouge foncé
DarkYellow              -> Jaune foncé
```

Chapitre 3

Introduction aux variables

Utilisation de variables dans notre programme

Ne serait-il pas bien que notre programme puisse maintenant afficher “Hello” avec le nom de l’utilisateur plutôt que d’afficher le général “Hello World” ? Pour ce faire, nous devons demander à l’utilisateur de saisir son nom, le stocker quelque part, puis afficher “Bonjour” avec le nom de l’utilisateur. Voyons comment nous pouvons réaliser cela :

```
TextWindow.Write("Entrez votre nom : ")
nom = TextWindow.Read()
TextWindow.WriteLine("Bonjour " + nom)
```

Lorsque vous tapez et exécutez ce programme, vous devriez voir apparaître la fenêtre suivante.

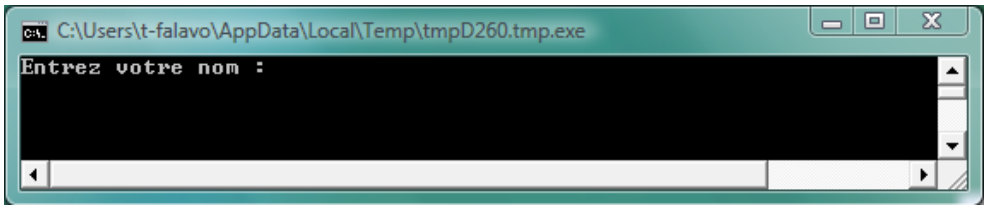

Figure 7 – Demande le nom d’utilisateur

Vous pouvez ensuite taper votre nom, puis appuyer sur la touche ENTRÉE. Voici ce que vous pouvez obtenir.

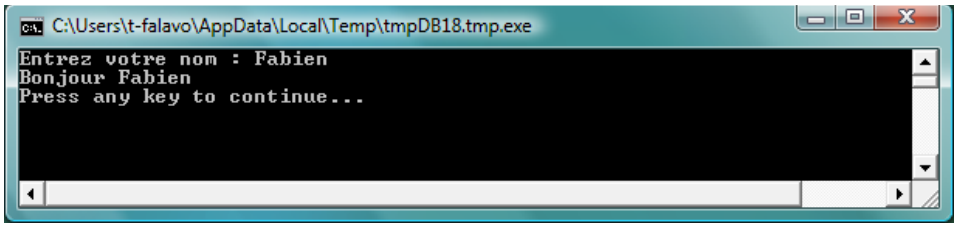

Figure 8 – Le message Bonjour s'affiche

Si vous lancer le programme une seconde fois, la même question vous sera posée. Vous pouvez saisir un nom différent et l'ordinateur vous répondra Bonjour avec ce nouveau nom.

Analyse du programme

Dans le programme que vous venez de lancer, la ligne suivante doit avoir retenu votre attention :

```
nom = TextWindow.Read()
```

Read() est similaire à *WriteLine()* mais n'affiche rien. Il s'agit d'une opération qui demande à l'ordinateur d'attendre que l'utilisateur saisisse quelque chose et appuie sur la touche ENTRÉE. Les informations qui ont été saisies par l'utilisateur sont ensuite mémorisées, votre programme peut donc, par la suite, y accéder. Le point intéressant est que, quelque soit ce que l'utilisateur a saisi, l'ensemble des données sont stockées dans une variable nommée **nom**. Une *variable* est une zone mémoire où vous pouvez stocker temporairement des valeurs et les utiliser plus tard. Dans la ligne ci-dessus, la variable **nom** a été utilisée pour stocker le nom de l'utilisateur.

La ligne suivante est aussi intéressante:

```
TextWindow.WriteLine("Bonjour " + nom)
```

C'est ici où nous utilisons la valeur stockée dans notre variable **nom**. Nous prenons la valeur de cette variable et l'attachons à `Bonjour` pour l'écrire dans la console TextWindow.

Une fois qu'une valeur est attribuée à la variable, vous pouvez la réutiliser autant que vous le souhaitez. Par exemple, vous pouvez écrire le code suivant :

> Write, tout comme WriteLine, est une autre opération de ConsoleWindow. *Write* vous permet d'écrire quelque chose dans la *ConsoleWindow*, mais permet d'ajouter du texte sur la même ligne que le texte actuel.

```
TextWindow.Write("Entrez votre nom : ")
nom = TextWindow.Read()
```

```
TextWindow.Write("Bonjour " + nom + ".  ")
TextWindow.WriteLine("Comment allez-vous " + nom + "?")
```

Voici le résultat d'exécution du code précédent :

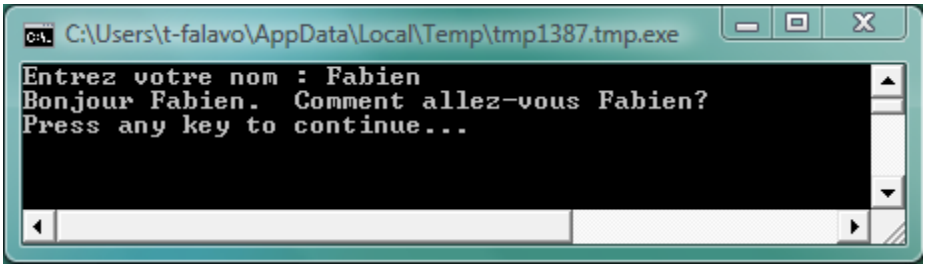

Figure 9 - Réutilisation d'une variable

Jouer avec les nombres

Nous venons de voir comment vous pouvez utiliser des variables pour stocker le nom de l'utilisateur. Au travers des exemples suivants, nous allons voir comment nous pouvons stocker et manipuler des nombres avec les variables. Commençons par un programme très simple:

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

Quand vous lancez ce programme, vous obtenez l'affichage suivant :

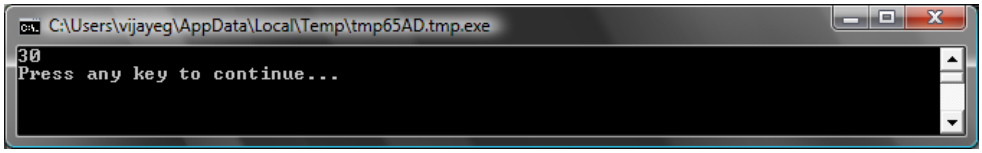

Figure 10 - Addition de deux nombres

Dans la première ligne du programme, vous affectez à la variable **number1** la valeur 10. Dans la seconde ligne, vous affectez la valeur 20 à la variable **number2**. Dans la troisième ligne, vous additionnez **number1** et **number2**, puis attribuez le résultat à la variable **number3**. Dans ce cas, **number3** se verra attribuer la valeur 30. C'est bien ce qui est affiché dans la console

On peut noter que les nombres ne sont pas entre guillemets. ***Pour les nombres, les guillemets ne sont pas nécessaires****. Vous n'avez besoin des guillemets uniquement lorsque vous utilisez du texte.*

TextWindow.

Maintenant, modifions le programme et voyons le résultat :

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

Le programme ci-dessus va multiplier **number1** par **number2** et stocker le résultat dans **number3**. Vous pouvez voir le résultat ci-dessous :

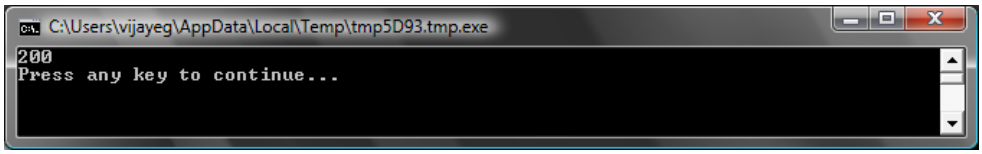

Figure 11 - Multiplication de deux nombres

De façon similaire, vous pouvez soustraire ou diviser des nombres. Voici la soustraction:

```
number3 = number1 - number2
```

Le symbole pour la division est '/'. Le programme ressemblera à ceci :

```
number3 = number1 / number2
```

Et le résultat pour la division est :

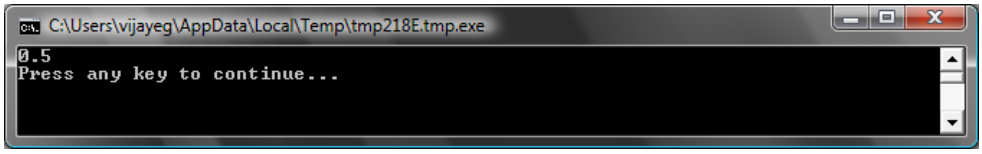

Figure 12 – Division de deux nombres

Un simple convertisseur de température

Pour le programme suivant, nous utiliserons la formule $℃ = \frac{5(°F-32)}{9}$ dans le but de convertir les degrés Fahrenheit en degrés Celsius.

Tout d'abord, récupérons la température en Fahrenheit de l'utilisateur et stockons-la dans une variable. Il y a une opération particulière qui nous permet de ne récupérer que des chiffres. Cette opération est **TextWindow.ReadNumber**.

```
TextWindow.Write("Entrez une température en Fahrenheit : ")
fahr = TextWindow.ReadNumber()
```

Une fois que nous avons la température en Fahrenheit stockée dans une variable, nous pouvons la convertir en degrés Celsius de la façon suivante :

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

Les parenthèses ordonnent à l'ordinateur de calculer la soustraction **`farh - 32`** en premier, puis d'effectuer le reste de l'opération. Maintenant, tout ce qu'il nous reste à faire est d'afficher le résultat à l'écran de l'utilisateur. En assemblant le tout, nous obtenons ce programme :

```
TextWindow.Write("Entrez une température en Fahrenheit : ")
fahr = TextWindow.ReadNumber()
celsius = 5 * (fahr - 32) / 9
TextWindow.WriteLine("La température en Celsius est " + celsius)
```

Et le résultat de ce programme sera:

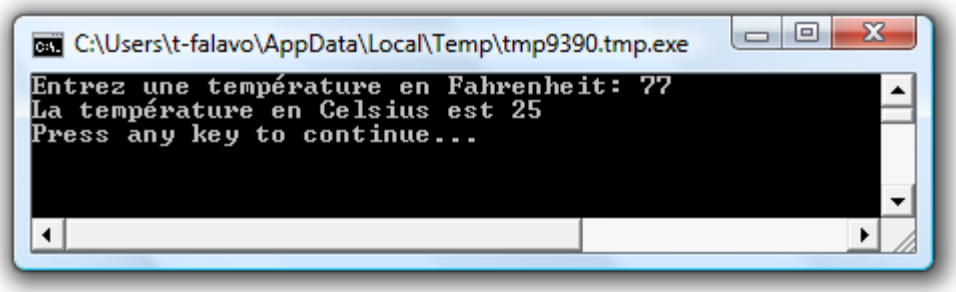

Figure 13 – Conversion de température

Chapitre 4

Conditions et branchements

Revenons à notre premier programme, ne serait-il pas cool que, plutôt que d'afficher le général *Hello World*, que nous puissions afficher *Good Morning World*, ou *Good Evening World* en fonction de l'heure? Pour notre prochain programme, nous allons afficher *Good Morning World* si nous sommes avant midi et *Good Evening World* dans le cas contraire.

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

En fonction de l'heure à laquelle vous lancerez le programme, vous verrez l'un des textes suivants:

Figure 14 - Good Morning World

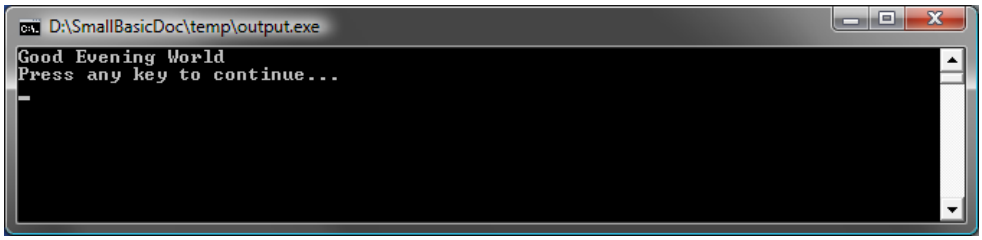

Figure 15 - Good Evening World

Analysons maintenant les trois premières lignes de notre programme. Vous l'avez déjà compris, la première ligne demande à l'ordinateur d'afficher "Good Morning World" si Clock.Hour est inférieur à 12. Les mots clés **If**, **Then** et **EndIf** sont des mots particuliers qui sont compris par l'ordinateur quand le programme est lancé. Le mot **If** est toujours suivi d'une condition, qui est, dans ce cas, **(Clock.Hour < 12)**. Souvenez-vous que les parenthèses sont nécessaires pour que l'ordinateur comprenne vos intentions. La condition est suivie de **Then** ainsi que de l'opération réelle à exécuter. L'opération est ensuite suivie de **EndIf**. Cela indique à l'ordinateur que l'exécution conditionnelle est terminée.

En Small Basic, vous pouvez utiliser l'objet Clock *pour accéder à la date et heure actuelle. Il vous fournis également un ensemble de propriétés qui vous permettent d'obtenir le jour, le mois, l'année, l'heure, les minutes, et les secondes séparément.*

Entre le mot **Then** et **EndIf**, il peut y avoir plus d'une opération et l'ordinateur les exécutera toutes si la condition est valide. Par exemple, vous pouvez écrire quelque chose de semblable à ceci:

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

Else

Dans le programme au début de ce chapitre, vous avez sans doute remarqué que la seconde condition était redondante. La valeur de **Clock.Hour** peut être inférieure à 12 ou non. Nous n'avons pas vraiment besoin de vérifier la seconde condition. Dans ce cas précis, nous pouvons réduire les deux déclarations **If..Then..Endif** à un seul en utilisant un nouveau mot clé, **Else**.

Si nous devions réécrire ce programme en utilisant **Else**, voici ce à quoi il ressemblerait :

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

```
EndIf
```

Ce programme va faire exactement la même chose que l'autre, ce qui nous emmène à une leçon très importante en programmation informatique:

> *En programmation, il y a généralement plusieurs façons de faire la même chose. Parfois, une façon de faire a plus de sens qu'une autre : le choix est laissé au programmeur. Plus vous écrirez des programmes, et plus vous deviendrez expérimenté, plus il sera facile pour vous d'identifier ces différentes techniques, leurs avantages, et inconvénients.*

Indentation

Dans tous les exemples précédents, vous pouvez voir comment les instructions entre les *If, Else* et *EndIf* sont indentées. L'indentation n'est pas obligatoire. L'ordinateur comprendra très bien le programme sans elle. Toutefois, elles nous aident à voir et à comprendre la structure du programme plus facilement. En conséquence, il est généralement considéré comme une bonne pratique d'indenter le code entre chaque bloc.

Pair ou Impair

Maintenant que nous avons les instructions **If..Then..Else..EndIf** dans notre boîte à outils, nous allons écrire un programme qui, en fonction d'un nombre donné, nous dira s'il est pair ou impair.

```
TextWindow.Write("Entrez un nombre : ")
num = TextWindow.ReadNumber()
remainder = Math.Remainder(num, 2)
If (remainder = 0) Then
  TextWindow.WriteLine("Le nombre est Pair")
Else
  TextWindow.WriteLine("Le nombre est Impair")
EndIf
```

Lorsque vous exécuterez ce programme, vous verrez quelque chose de semblable à ceci:

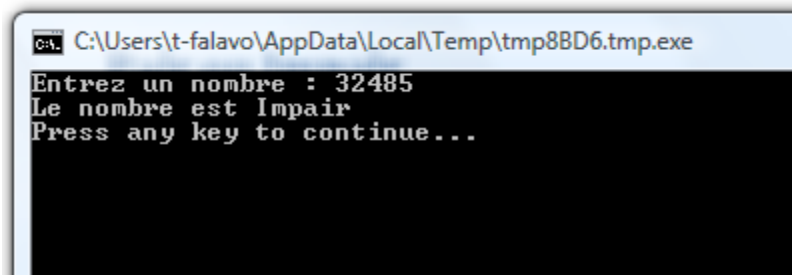

Figure 16 - Pair ou Impair

Dans ce programme, nous avons introduit une nouvelle opération qui peut vous être utile, **Math.Remainder**. Comme vous l'avez peut-être déjà compris, **Math.Remainder** va diviser le premier nombre par le second et retourne le reste.

Branchements

Rappelez-vous, dans le deuxième chapitre, vous avez appris que l'ordinateur traite un programme une déclaration à la fois, dans l'ordre de haut en bas. Toutefois, il y a une instruction spéciale qui peut ordonner à l'ordinateur de sauter d'une opération à une autre. Jetons un œil au prochain programme.

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

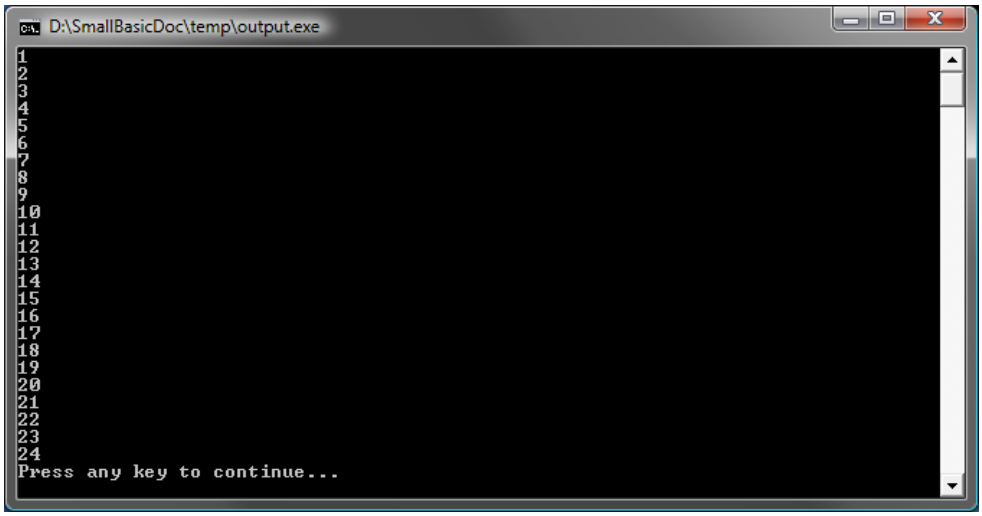

Figure 17 - Utiliser Goto

Dans le programme ci-dessus, nous avons assigné la valeur 1 à la variable **i**. Et puis nous avons ajouté une nouvelle opération qui se termine par le double point (:)

```
start:
```

C'est ce qu'on appelle une *étiquette* (ou *label* en anglais). Les étiquettes sont des marqueurs que l'ordinateur peut comprendre. Vous pouvez nommer l'étiquette comme vous voulez et vous pouvez ajouter autant d'étiquettes que vous souhaitez dans votre programme tant qu'elles portent toutes un nom différent.

Une autre opération intéressante est celle-ci :

```
i = i + 1
```

Cela indique à l'ordinateur d'ajouter 1 à la variable **i** et d'assigner le résultat à cette même variable **i**. Donc, si la valeur de i était de 1 avant cette opération, elle sera de 2 après que cette déclaration soit exécutée.

Et enfin, le code suivant indique à l'ordinateur que si la valeur de i est inférieure à 25, il doit commencer à exécuter les déclarations à partir de l'étiquette **start**.

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

Exécution sans fin

En utilisant l'instruction **Goto**, vous pouvez répéter des instructions un nombre indéterminé de fois. Par exemple, vous pouvez prendre le programme Pair ou Impair et le modifier comme ci-dessous. Vous pouvez stopper le programme en cliquant sur le bouton de fermeture (X) dans le coin à droite de la fenêtre.

```
begin:
TextWindow.Write("Entrez un nombre : ")
num = TextWindow.ReadNumber()
remainder = Math.Remainder(num, 2)
If (remainder = 0) Then
  TextWindow.WriteLine("Le nombre est Pair")
Else
  TextWindow.WriteLine("Le nombre est Impair")
EndIf
Goto begin
```

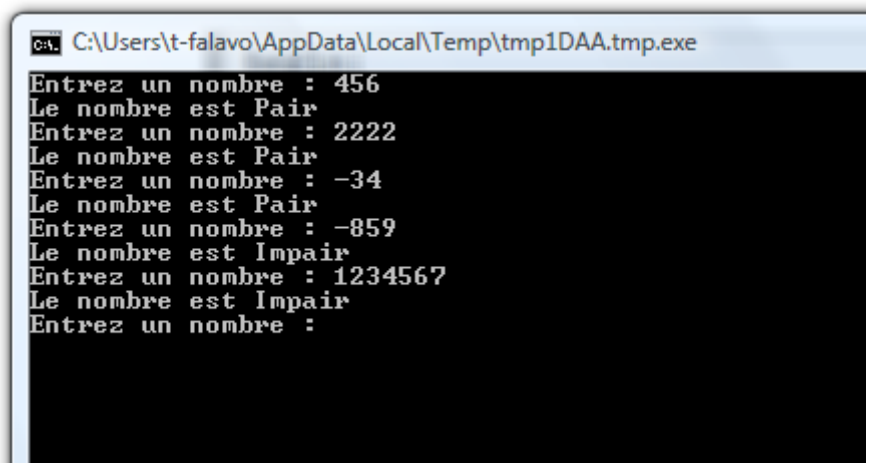

Figure 18 – Le programme Pair ou Impair tournant indéfiniment

Chapitre 5

Les boucles

Boucle For

Reprenons le programme écrit dans le chapitre précédent.

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

Ce programme affiche des nombres de 1 à 24 selon l'ordre numérique. Le fait d'incrémenter une variable est très courant en programmation et les langages de programmation fournissent généralement des méthodes simplifiées pour cela. Le programme ci-dessous est équivalent au programme précédent :

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

On obtient alors:

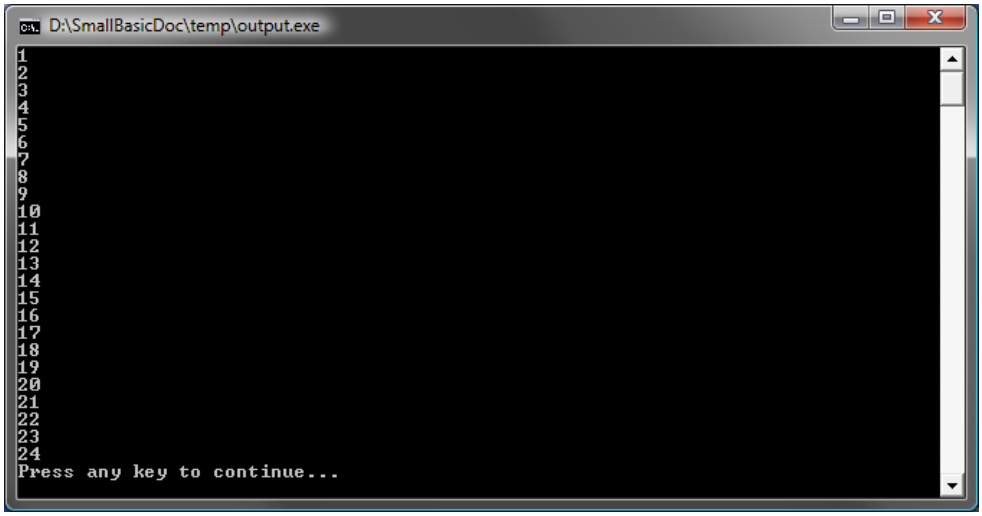

Image 19 – Utilisation d'une boucle FOR

On remarque que l'on est passé d'un programme de 8 lignes à un programme 4 lignes en obtenant le même résultat. Souvenez-vous que, plus tôt, nous disions qu'il y avait généralement plusieurs façons de faire la même chose : ceci en est un bon exemple.

For..EndFor est appelé, en programmation, une boucle. Cela vous permet de prendre une variable, de lui donner une valeur initiale et une valeur finale et de laisser l'ordinateur incrémenter cette variable pour vous. Chaque fois que l'ordinateur incrémente la variable, cela entraîne l'exécution du code présent entre **For** et **EndFor**.

Mais si vous vouliez que la variable soit incrémentée de 2 au lieu d'1, par exemple pour afficher tous les nombres impairs entre 1 et 24, vous pouvez utiliser une boucle pour effectuer cela.

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

Image 20 – Affichage des nombres impairs

Le **Step 2** associé au mot clé **For** indique à l'ordinateur qu'il doit incrémenter la valeur de **i** de deux unités au lieu d'une unité habituellement. En utilisant le mot clé **Step**, vous pouvez spécifier l'incrémentation que vous souhaitez. Il est possible de spécifier une valeur négative pour le mot clé **Step** afin que l'ordinateur compte à l'envers, comme dans l'exemple ci-dessous :

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

Image 21 – Compter à l'envers

Boucle While

Une boucle **While** est un autre type de boucle qui est utile lorsque l'on souhaite faire une boucle sans connaître exactement le nombre d'itérations à l'avance. Contrairement à la boucle **For** qui tourne un nombre prédéfini de fois, la boucle **While** s'exécute jusqu'à ce qu'une condition soit vérifiée. Dans l'exemple ci-dessous, on divise un nombre par 2 tant que le résultat est supérieur à 1.

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

Image 22 – Boucle de division par 2

Dans le programme ci-dessus, on assigne la valeur 100 à la variable *number* et on exécute la boucle **While** tant que la variable *number* est supérieure à 1. Au sein de la boucle, on affiche le nombre puis on le divise par deux. Comme attendu, on peut voir à l'écran que les nombres sont réduits de moitié les uns après les autres.

Il aurait été très difficile d'écrire ce programme en utilisant une boucle **For** car nous ne savons pas à l'avance le nombre de boucle à effectuer. Avec une boucle **While**, il est facile de vérifier une condition et d'indiquer à l'ordinateur de continuer la boucle ou de sortir de celle-ci.

Il est intéressant de noter que chaque boucle **While** peut être remplacée en utilisant une condition **If**..**Then**. Par exemple, le programme précédent peut être réécrit de la façon suivante.

```
number = 100
startLabel:
TextWindow.WriteLine(number)
number = number / 2
If (number > 1) Then
  Goto startLabel
EndIf
```

*En réalité, l'ordinateur réécrit chaque boucle **While** un utilisant des conditions **If…Then** avec un ou plusieurs mots clé **Goto**.*

Chapitre 6

Introduction à la programmation graphique

Jusqu'à présent, tous nos exemples ont été réalisés en utilisant **TextWindow** afin d'expliquer les bases du langage Small Basic. Toutefois, Small Basic contient un ensemble d'outils permettant la réalisation d'applications graphiques.

Présentation de GraphicsWindow

Tout comme nous avions TextWindows pour le texte et les nombres, Small Basic fournit également **GraphicsWindow** qui permet de dessiner des objets graphiques. Commençons par afficher une **GraphicsWindow**.

```
GraphicsWindow.Show()
```

Quand on lance ce programme, on remarque qu'à la place de l'écran noir de texte, on obtient une fenêtre comme celle de l'image-ci-dessous. On ne peut encore rien faire sur cette fenêtre mais c'est la fenêtre de base sur laquelle nous allons travailler durant ce chapitre. Vous pouvez fermer cette fenêtre en cliquant sur le bouton 'X' dans le coin en haut à droite.

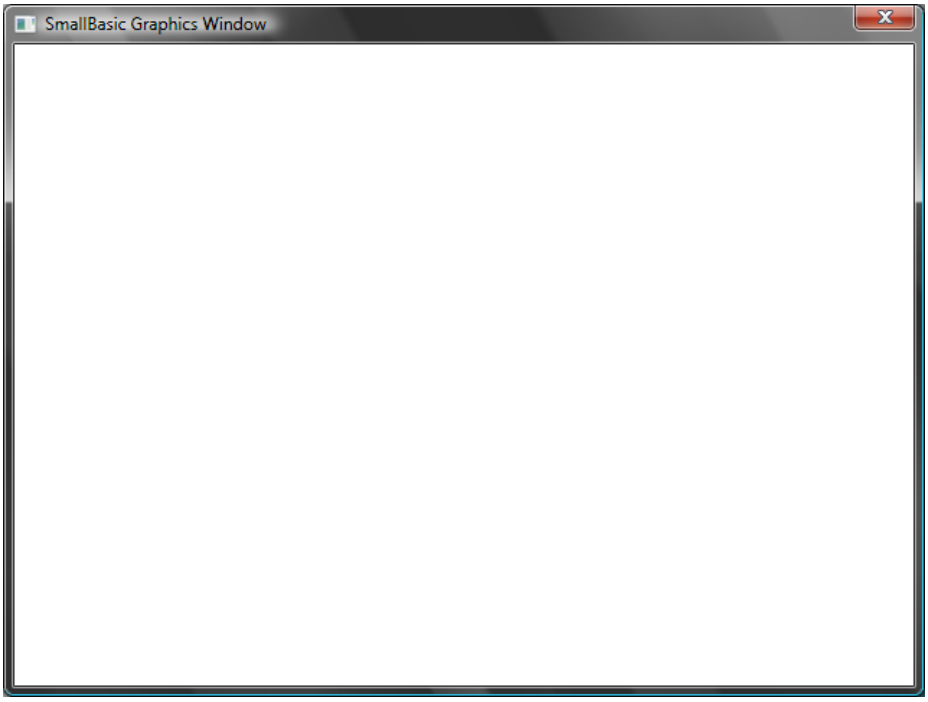

Image 23 – Une fenêtre vide

Configuration de la fenêtre graphique

GraphicsWindow vous permet de configurer l'apparence de la fenêtre selon vos désirs. Vous pouvez ainsi changer son titre, la couleur de fond ou encore sa taille. Commençons par modifier quelques propriétés de base.

```
GraphicsWindow.BackgroundColor = "SteelBlue"
GraphicsWindow.Title = "Ma fenêtre graphique"
GraphicsWindow.Width = 320
GraphicsWindow.Height = 200
GraphicsWindow.Show()
```

Voici à quoi ressemble la fenêtre modifiée par le code ci-dessus. Vous pouvez changer la couleur de fond par l'une des nombreuses valeurs listées dans l'annexe B de ce document. Jouez avec ces propriétés pour voir comment modifier l'apparence de la fenêtre.

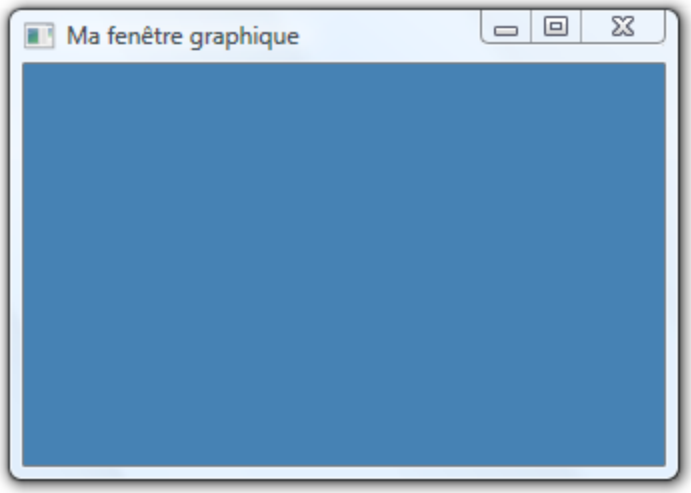

Image 24 – Une fenêtre personnalisée

Dessiner des lignes

Une fois la fenêtre prête, on peut dessiner des formes, du texte et même des images dessus. Commençons par dessiner des formes simples. Voici un programme qui dessine deux lignes sur la fenêtre.

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

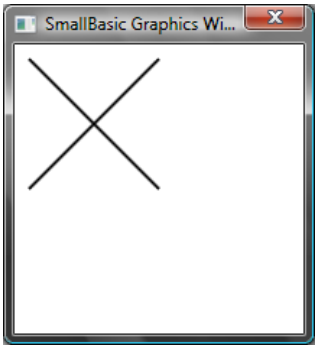

Image 25 – Une croix

Les deux premières lignes du programme définissent la taille de la fenêtre, et les deux lignes suivantes dessinent les traits de la croix. Les deux premiers nombres à droite de *DrawLine* définissent les coordonnées x et y du point de départ de chaque ligne et les deux autres nombres définissent les coordonnées x et y du point d'arrivé de chaque ligne.

Un point intéressant en informatique est que les coordonnées du coin haut gauche de la fenêtre sont (0,0). Par conséquent, dans un repère orthonormé la fenêtre est considérée comme étant dans le deuxième quadrant.

Revenons à notre programme de dessin. Tout comme nous avons modifiés des propriétés de la fenêtre, nous pouvons modifier des propriétés du trait, comme sa couleur ou son épaisseur.

Tout d'abord, modifions la couleur des lignes :

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

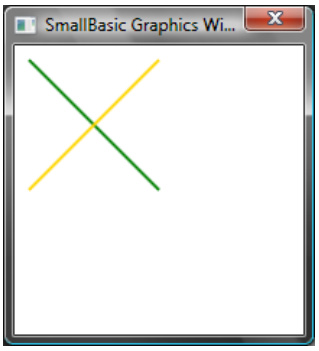

Image 26 – Changement de couleur des lignes

Maintenant, modifions la taille des traits de la croix. Dans le programme ci-dessous, nous changeons la largeur de la ligne à 10 alors que la valeur par défaut de cette propriété est 1.

Plutôt que d'utiliser des noms pour les couleurs, vous pouvez utiliser la notation des couleurs web (#RRGGBB). Par exemple, FF0000 donne du rouge, #FFFF00 du jaune et ainsi de suite.

Pour en savoir plus sur les couleurs ➔ Annexes

```
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)
```

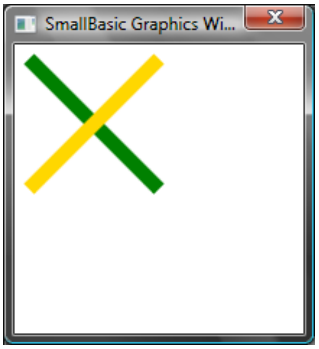

Image 27 – Une croix épaisse et colorée

PenWidth et *PenColor* modifient le crayon utilisé pour dessiner ces lignes. Cela n'affecte pas uniquement les lignes mais aussi toutes les formes dessinées avec le crayon.

En utilisant les boucles que nous avons étudiées dans le chapitre précédent, nous pouvons facilement écrire un programme qui dessine plusieurs lignes en augmentant l'épaisseur du crayon utilisé.

```
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
```

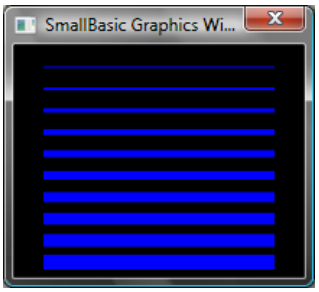

Image 28 – Épaisseurs du crayon différentes

La partie intéressante du programme est la boucle, où nous augmentons l'épaisseur du crayon (*PenWidth*) à chaque fois que la boucle est lancée, puis nous dessinons la nouvelle ligne en dessous de l'ancienne.

Dessiner des formes

Lorsque vous souhaitez dessiner des formes, il existe deux catégories de fonctions à votre disposition : les fonctions DrawXXX et les fonctions FillXXX. Les fonctions *Draw* dessinent le contour de la forme en utilisant un crayon donné (défini par *PenColor*). Les fonctions *Fill* dessinent une forme pleine grâce à un pinceau (défini par *BrushColor*).

Par exemple, dans le programme ci-dessous, deux rectangles sont dessinés. L'un en utilisant un crayon rouge pour le contour et l'autre en utilisant un pinceau vert pour le remplissage du rectangle.

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

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

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

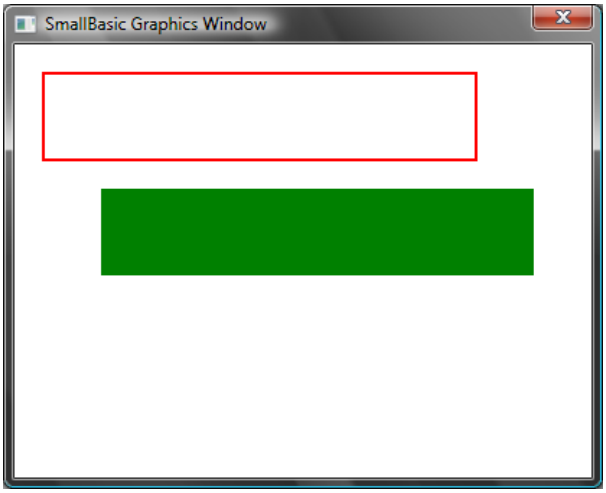

Image 29 Dessiner et colorier

Pour dessiner un rectangle, il faut 4 nombres. Les deux premiers nombres représentent les coordonnées X et Y du coin supérieur gauche du rectangle. Le troisième nombre permet de définir la largeur du rectangle, tandis que le quatrième nombre définit sa hauteur. On dessine des ellipses de la même façon comme le montre le programme ci-dessous.

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

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

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

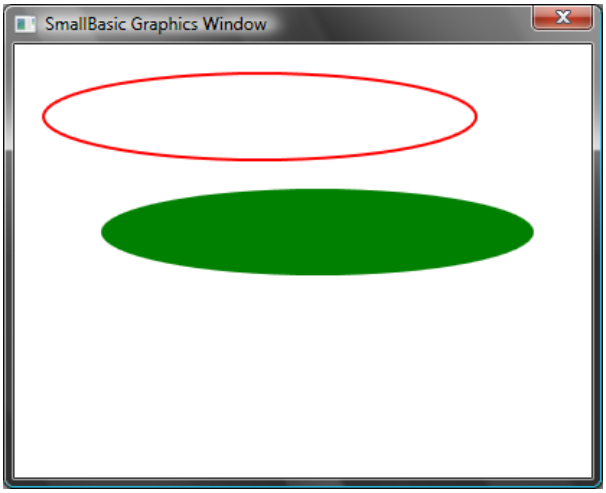

Image 30 – Dessin d'ellipses

Les Ellipses ne sont que des cas généraux des cercles. Si vous souhaitez dessiner un cercle, il suffit de spécifier la même hauteur et la même largeur.

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

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

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

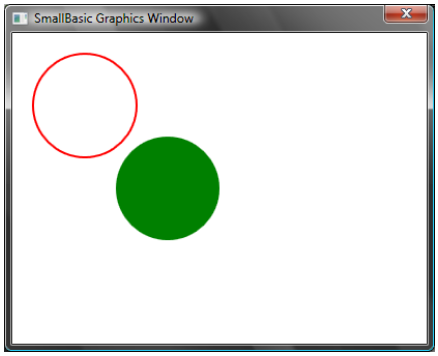

Image 31 – Cercles

Chapitre 7

Amusons-nous avec les formes

Nous allons nous amuser dans ce chapitre avec tout ce que vous avez appris jusqu'ici. Ce chapitre contient des extraits montrant comment combiner ce que vous avez appris pour créer des programmes graphiques funs.

Rectangulaires

Dans cette partie, nous dessinons plusieurs rectangles dans une boucle, en augmentant leur taille.

```
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
```

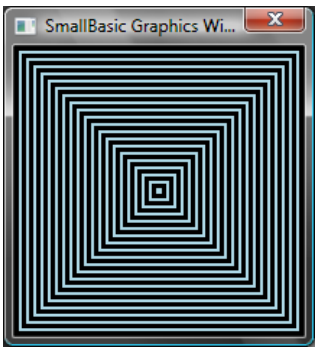

Figure 32 - Rectangalore

Circulaires

Une variante du précédent programme où il s'agit de dessiner des cercles à la place des carrés.

```
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
```

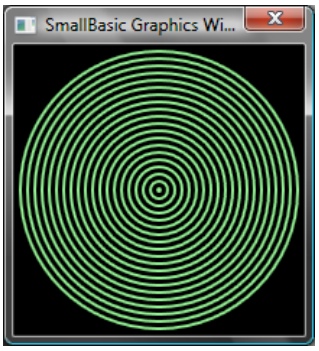

Figure 33 – Circtacular

Aléatoires

Ce programme utilise la commande GraphicsWindow.*GetRandomColor* pour fixer une couleur aléatoire au pinceau et ensuite la commande Math.*GetRandomNumber* pour attribuer les coordonnées x et y au cercle. Ces deux opérations peuvent être combinées de manières intéressantes afin de créer des programmes qui donnent des résultats différents à chaque lancement.

```
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
```

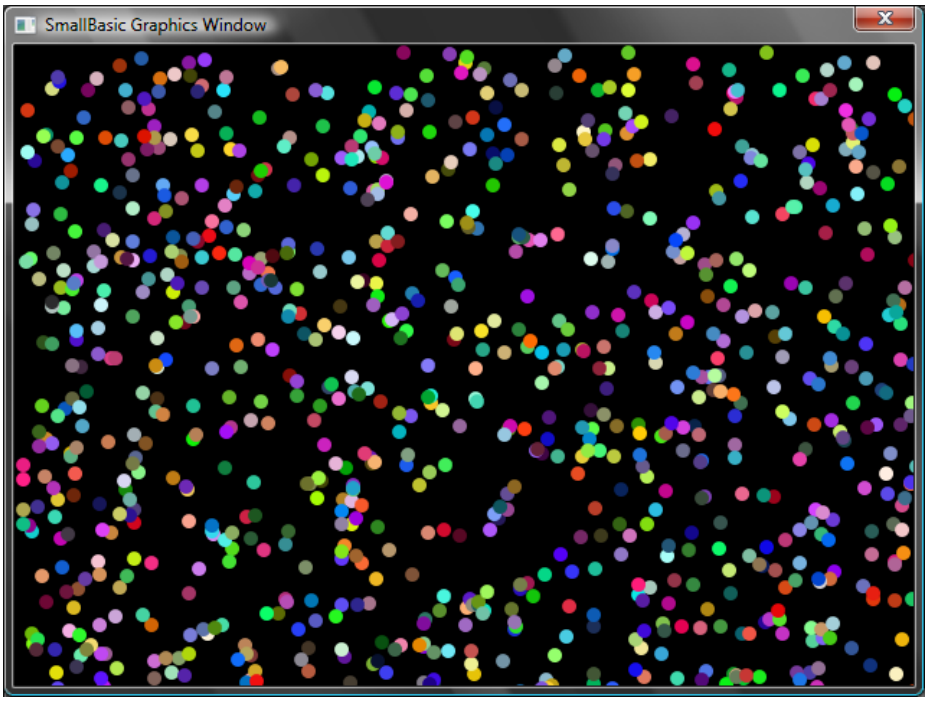

Figure 34 – Randomize

Fractales

Le programme suivant dessine un triangle en fractal en utilisant les nombres aléatoires. Une fractale est une forme géométrique qui peut être subdivisée en parties, chaque partie ressemblant trait pour trait à la forme parente. Dans notre cas, le programme dessine des centaines de triangles, chacun d'eux étant identique à son triangle parent. Et comme le programme tourne quelques secondes, vous pouvez réellement voir les triangles se former doucement à partir de simples points. La logique est quelque peu difficile à décrire et je vous laisse le soin, comme exercice, de l'approfondir.

```
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
```

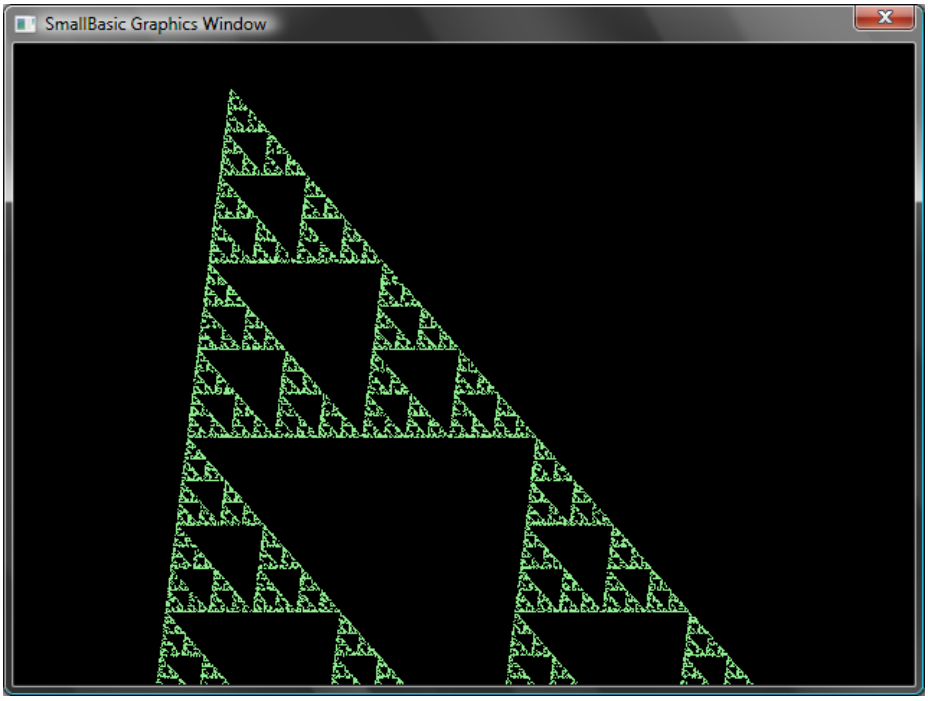

Figure 35 - Triangle Fractal

Si vous voulez vraiment voir les points se transformer pour former la fractale, vous pouvez ajouter une pause dans la boucle en utilisant la commande **Program.Delay**. Cette commande prend comme paramètre un nombre en millisecondes qui spécifie le délai d'attente. Ci-dessous le programme modifié et la ligne modifiée en gras.

```
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
```

Le fait d'augmenter le délai va ralentir le programme. Testez avec différents nombres pour voir lequel correspond le plus à ce que vous souhaitez.

Une autre modification que vous pouvez apporter à ce programme est de remplacer la ligne ci-dessous

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

par

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

Après cette modification, le programme dessine les pixels du triangle en utilisant des couleurs aléatoires.

Chapitre 8

Graphique Tortue

Logo

Dans les années 70, il existait un très simple mais très performant langage de programmation, appelé Logo qui était utilisé par quelques chercheurs. C'était avant que soit ajouté au langage ce que l'on a appelé « Turtle Graphics » et ainsi mette à disposition une « Tortue » qui était visible sur l'écran et répondait aux commandes comme « bouge », « avance », « tourne à droite », « tourne à gauche »,etc. En utilisant la tortue, les gens pouvaient dessiner des formes intéressantes sur l'écran. Ceci a rendu en conséquence le langage accessible et attractif aux personnes de tous âges, et a contribué à son succès dans les années 80.

Small basic propose la Tortue avec de nombreuses commandes qui peuvent être appelées à l'intérieur de programmes basés sur ce langage. Dans ce chapitre, nous allons utiliser la Tortue pour dessiner des éléments graphiques à l'écran.

La Tortue

Pour commencer, nous avons besoin que la tortue soit visible sur l'écran. Ceci peut être réalisé à l'aide d'une simple ligne de commande.

```
Turtle.Show()
```

Quand vous lancez ce programme, vous allez observer une fenêtre banche, comme celle que vous avez pu voir dans le chapitre précédent, sauf que celle-ci a une tortue au centre. C'est cette tortue qui va suivre nos instructions et dessiner tout ce qu'on lui demande.

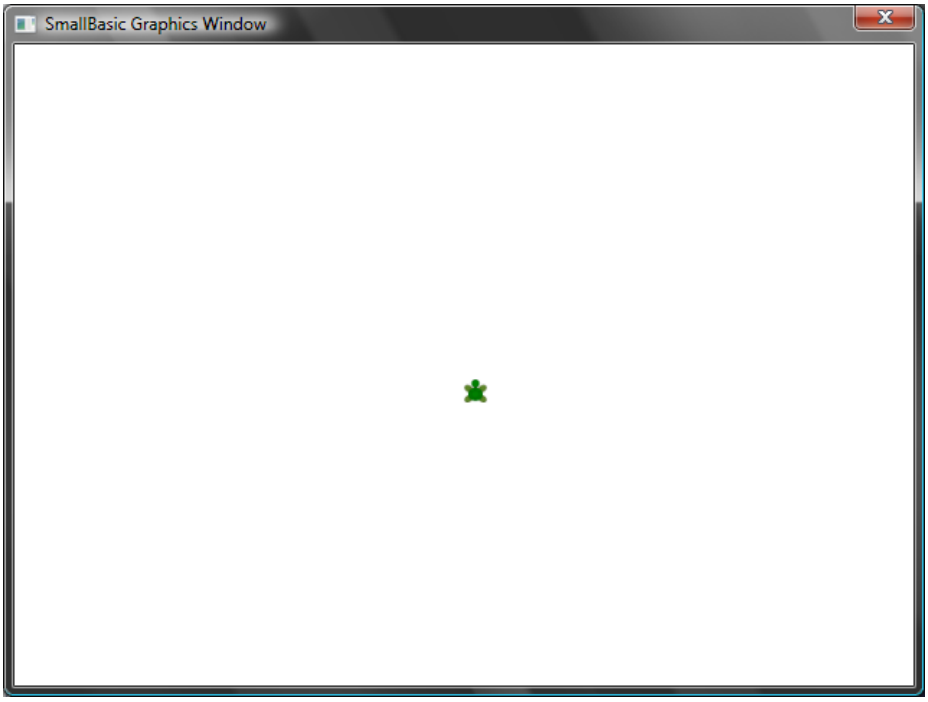

Figure 36 – La Tortue est visible

Bouger et dessiner

Une des instructions que la tortue peut comprendre est « **Move** ». Cette commande prend un nombre comme entrée. Ce nombre indique à la tortue de quelle distance elle doit se déplacer. Cela étant établi, dans l'exemple ci-dessous, nous allons demander à la tortue de bouger de 100 pixels.

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

Quand vous lancez le programme, vous pouvez voir la tortue bouger lentement de 100 pixels vers le haut. Vous pouvez aussi remarquer qu'une ligne se dessine derrière elle. Quand la Tortue a fini son mouvement, le résultat devrait ressembler à la figure ci-dessous.

Quand vous utilisez la tortue, il n'est pas nécessaire d'appeler la fonction Show(). La Tortue sera rendu visible du moment que n'importe quelle opération est exécutée.

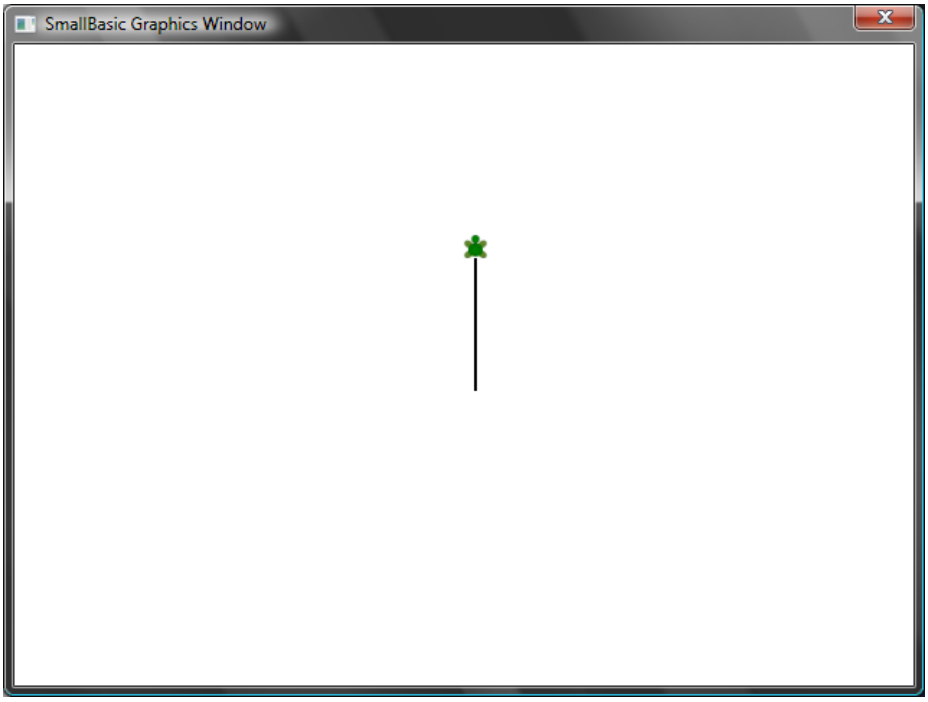

Figure 37 – Mouvement de 100 pixels

Dessiner un carré

Un carré a 4 côtés, 2 verticaux et 2 horizontaux.

Afin de dessiner un carré nous avons besoin que la tortue dessine une ligne, tourne à droite et dessine une autre ligne et ainsi de suite jusqu'à ce que les 4 côtés soient finis. Si nous traduisons cela en un programme, il devrait ressembler à cela.

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

Quand vous exécutez le programme, vous pouvez voir la Tortue dessiner un carré, une ligne après l'autre, et le résultat ressemble à la figure ci-dessous.

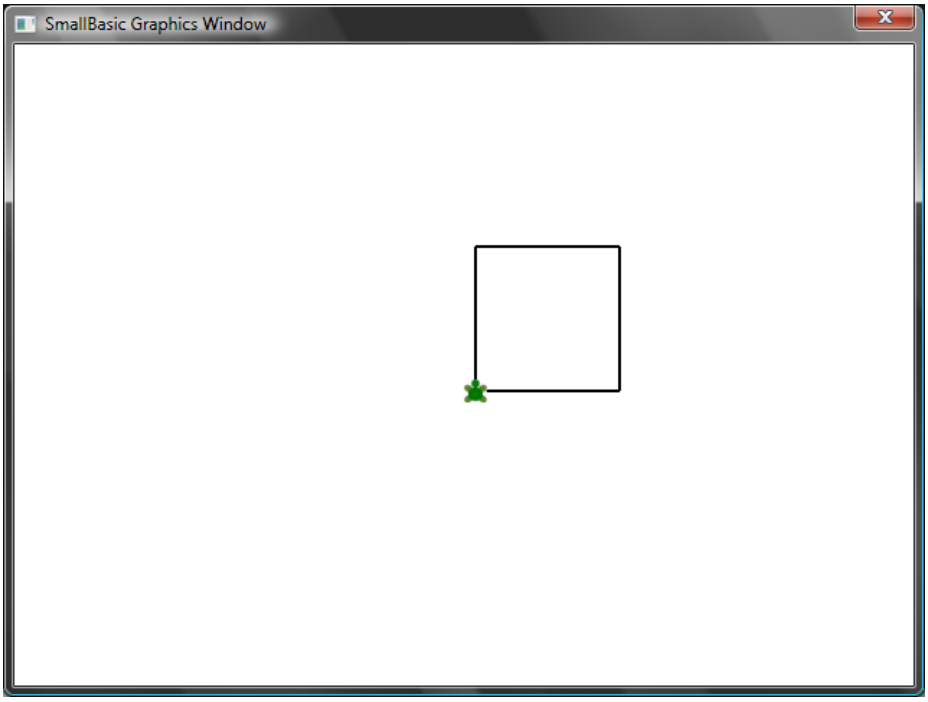

Figure 38 – La tortue dessine un carré

Il est intéressant de noter que nous sommes en train d'utiliser les deux mêmes instructions encore et encore - 4 fois précisément. Et nous avons déjà appris que de telles commandes répétitives peuvent être exécutées en utilisant des boucles. Donc, si nous prenons le programme au-dessus et que nous le modifions en utilisant la boucle « For..EndFor », nous allons nous retrouver avec un programme beaucoup plus simple.

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

Modifier les couleurs

La Tortue dessine sur exactement le même composant graphique (GraphicsWindow) que nous avons vu dans le chapitre précédent. Cela signifie que toutes les opérations apprises dans le chapitre précédent sont toujours valables. Par exemple, le programme suivant va dessiner un carré avec chacun des cotés d'une couleur différente.

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

```
EndFor
```

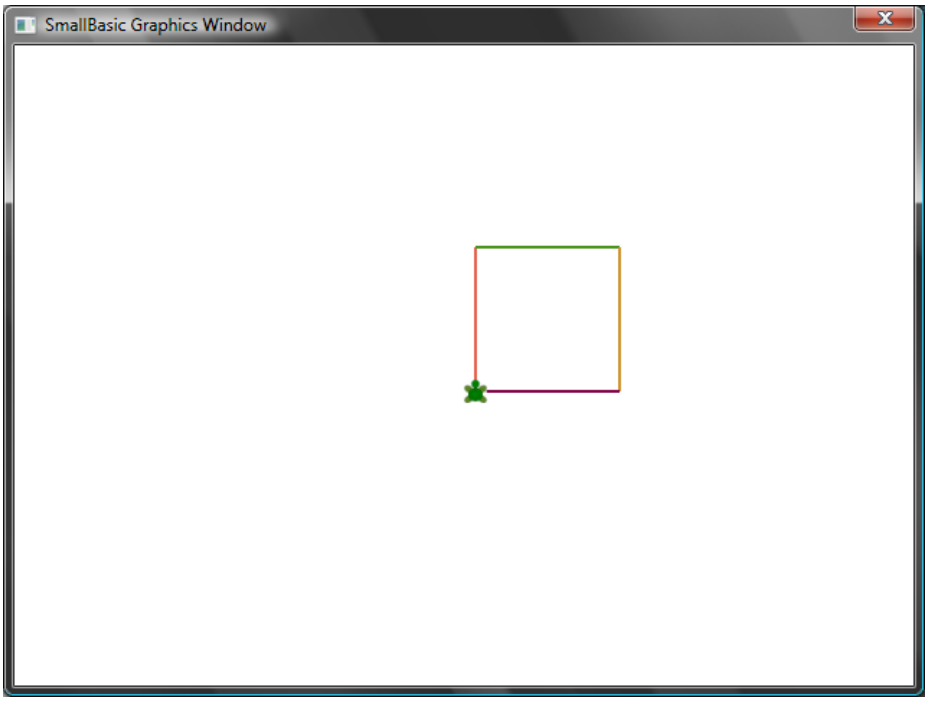

Figure 39 – modification des couleurs

Dessiner des formes plus complexes

La tortue, en plus des commandes TurnRight et TurnLeft, peut en interpréter une autre qui est Turn. Celle-ci prend une entrée qui spécifie l'angle de rotation. En utilisant cette commande, il est possible de dessiner n'importe quel polygone-à-n-côtés. Le programme suivant dessine un hexagone (un polygone à six côtés).

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

Essayez ce programme pour voir s'il dessine vraiment un hexagone. Remarquez que puisque l'angle entre les cotés est de 60 degrés, nous utilisons **Turn(60).** Pour un tel polygone, dont les côtés sont égaux, l'angle entre les côtés peut être facilement obtenu en divisant 360 par le nombre de côtés. Armé de ces informations et en utilisant des variables, nous pouvons écrire un joli petit programme générique qui sera capable de dessiner n'importe quel polygone-à-côtés.

```
sides = 12
```

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

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

En utilisant ce programme, vous pouvez dessiner n'importe quel polygone en modifiant seulement la variable **sides** (qui veut dire « côtés »). En mettant 4, nous allons obtenir le carré avec lequel nous avons débuté. En mettant une valeur suffisamment grande, disons 50, le résultat obtenu ressemble quasiment à un cercle.

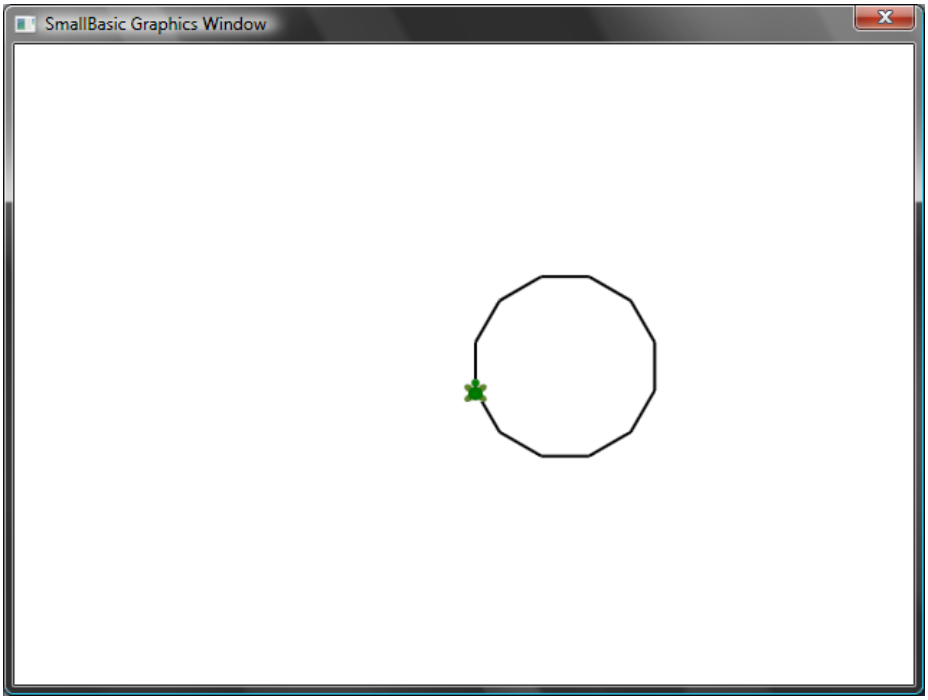

Figure 40 – Dessin d'un polygone à 12 côtés

En utilisant la technique que nous venons d'apprendre, nous pouvons faire en sorte que la Tortue dessine à chaque fois plusieurs cercles avec un petit déplacement produisant un résultat intéressant.

```
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
```

Le programme ci-dessus a deux boucles **For..EndFor**, imbriquée l'une dans l'autre. La boucle externe (i=1 jusqu'à **sides**) est similaire au programme polygone et est en charge du dessin du cercle. La boucle externe (j=1 jusqu'à 20) s'occupe de faire tourner la tortue d'un certain angle après avoir dessiné chaque cercle. Ceci commande à la Tortue de dessiner 20 cercles. Donnant comme résultat le motif suivant.

Dans le programme ci-dessus, nous avons fait en sorte que la Tortue se déplace plus vite en ajustant la vitesse à 9. Vous pouvez fixer cette propriété à n'importe quelle valeur entre 1 et 10 pour que la Tortue se déplace à la vitesse que vous souhaitez.

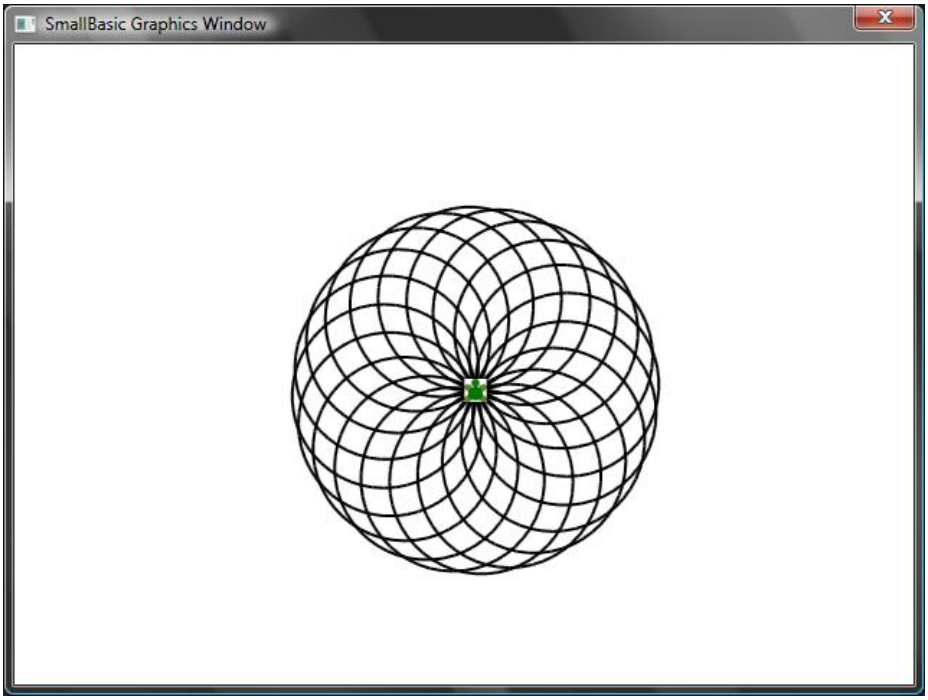

Figure 41 – mouvement de cercles

Déplacement

Vous pouvez demander à la tortue de ne pas dessiner en appelant la commande **PenUp** (signifiant « Lever le stylo »). Ceci vous permet de bouger la tortue n'importe où sur l'écran sans dessiner de ligne. En appelant la commande **PenDown** (signifiant « Baisser le stylo ») la Tortue dessinera de nouveau. Ceci

peut être utilisé pour créer des effets sympas, comme des lignes en pointillés. Le programme ci-contre utilise cela pout dessiner un polygone en pointillés.

```
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
```

De nouveau, ce programme a 2 boucles. La boucle interne dessine une seule ligne en pointillés, pendant que l'externe indique combien de lignes dessiner. Dans notre exemple, nous utilisons 6 pour la variable **sides** et donc nous obtenons un hexagone en pointillés, comme ci-dessous.

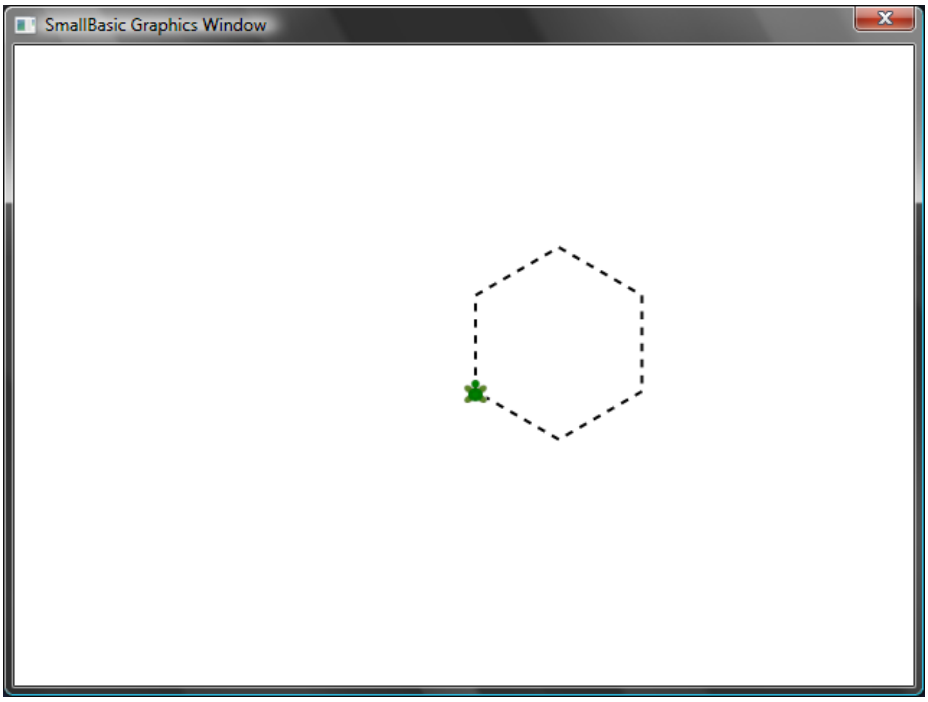

Figure 42 – Utilisation de PenUp et PenDown

Chapitre 9

Les routines

Lorsqu'on écrit un programme, on se retrouve souvent dans la situation de devoir exécuter les mêmes commandes encore et encore. Dans ces cas là, écrire plusieurs fois les mêmes lignes de code est une perte de temps. C'est là tout l'intérêt des routines.

Une routine est une portion de code que l'on intègre dans un plus grand programme, qui a un but spécifique et que l'on peut appeler de partout dans le programme. Les routines sont identifiées par un nom qui se place juste après le mot clé « Sub » et qui se termine par le mot clé « EndSub ». Par exemple, la portion de code ci-dessous représente une routine de nom *PrintTime*, et qui imprime l'heure courante dans la fenêtre texte.

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

Ci-dessous un programme qui inclut la routine *PrintTime* et l'appelle à plusieurs endroits dans le code :

```
PrintTime()
TextWindow.Write("Entrez votre nom : ")
nom = TextWindow.Read()
TextWindow.Write(nom + ", il est : ")
PrintTime()

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

```
EndSub
```

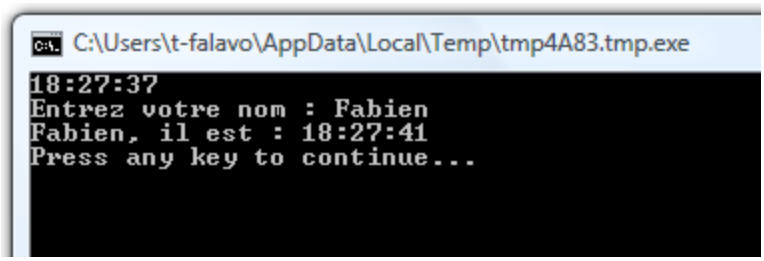

Figure 43 - Appel d'une routine simple

Vous exécutez une routine en appelant <NomDeMaRoutine>(). Comme d'habitude, les parenthèses sont nécessaires pour indiquer à l'ordinateur que l'on veut exécuter une routine.

Avantages de l'utilisation de routines

Comme nous l'avons vu ci-dessus, les routines permettent de réduire la quantité de code à taper. Une fois la routine *PrintTime* écrite, vous pouvez l'appeler de n'importe où dans votre programme et elle vous imprimera l'heure actuelle.

Rappelez-vous, vous ne pouvez appeler une routine que si elle est située dans le même programme. Vous ne pouvez pas appeler une routine située dans un autre programme.

De plus, les routines permettent de découper un problème en plusieurs plus simples. Imaginons que vous ayez une équation complexe à résoudre. Vous pourriez écrire plusieurs routines qui résolvent une partie de l'équation, puis en rassemblant les différents résultats des routines, les assembler pour reconstituer le résultat de l'équation complexe.

Les routines améliorent aussi la lisibilité du code. En d'autres mots, si vous avez des routines bien nommées pour des portions de code souvent exécuté, votre programme devient plus simple à lire et à comprendre. Ceci est très important si vous souhaitez pouvoir comprendre le programme d'un autre, ou bien si vous souhaitez que quelqu'un d'autre puisse comprendre votre programme. Il n'est même pas rare que l'on souhaite relire son code une semaine plus tard et il est alors nécessaire de l'avoir rendu accessible et intelligible.

Utilisation de variables

Dans une routine, vous pouvez utiliser et accéder à n'importe quelle variable que vous avez dans un programme. Par exemple, le programme suivant prend deux nombres en entrée et écrit le plus grand des deux. Remarquez par ailleurs que la variable « max » est utilisée hors et dans la routine.

```
TextWindow.Write("Entrez un premier nombre : ")
num1 = TextWindow.ReadNumber()
TextWindow.Write("Entrez un second nombre : ")
num2 = TextWindow.ReadNumber()

FindMax()
TextWindow.WriteLine("Le nombre maximum est : " + max)

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

Et la sortie de ce programme nous donne :

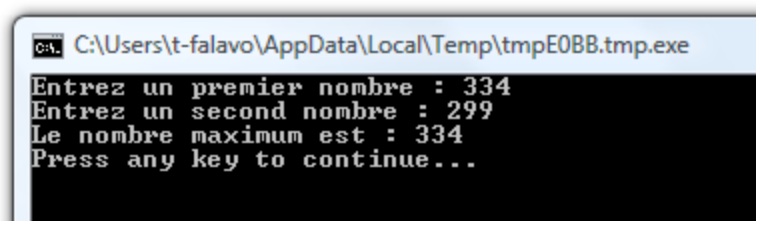

Figure 44 - Max de 2 nombres en utilisant une routine.

Regardons un autre exemple qui illustrera l'utilisation des routines. Cette fois-ci, nous utiliserons un programme graphique qui calcule plusieurs points stockés dans les variables *x* et *y*. Il appelle ensuite la routine *DrawCircleUsingCenter* (DessinerUnCercleAPartirDeSonCentre), qui se chargera de dessiner un cercle dont le centre a pour coordonnées *x* et *y*.

```
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
```

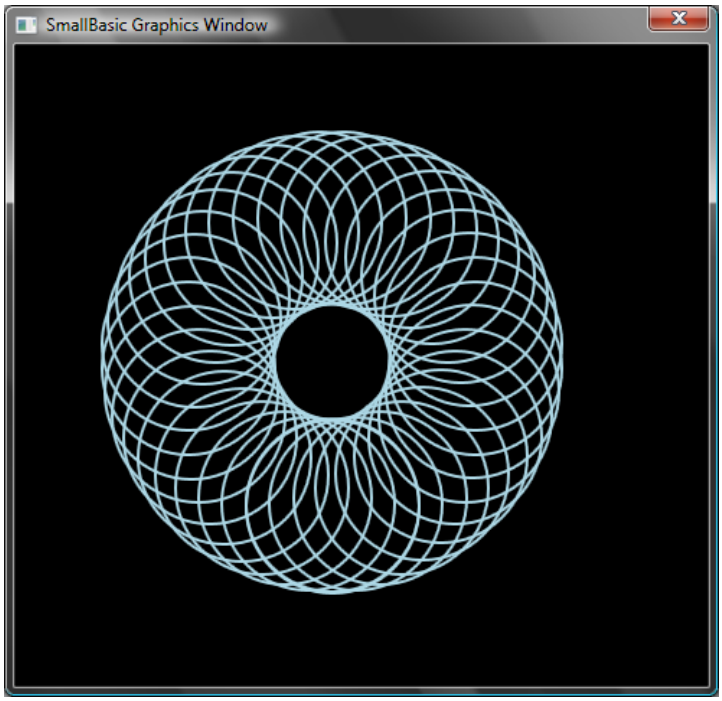

Figure 45 – Exemple graphique d'utilisation de routine

Appeler des routines dans une boucle

Parfois les routines sont appelées dans une boucle, dans laquelle elles exécutent donc la même portion de code mais avec des valeurs différentes passées en paramètre. Par exemple, si vous aviez une routine appelée *PrimeCheck* qui permet de définir si un nombre est premier ou pas, vous pourriez écrire un programme qui laisserait l'utilisateur saisir un nombre et lui indiquerait s'il a saisi un nombre premier ou pas. Voilà le programme qui illustre cette situation :

```
TextWindow.Write("Entrez un nombre : ")
i = TextWindow.ReadNumber()
estPremier = "True"
PrimeCheck()
If (estPremier = "True") Then
  TextWindow.WriteLine(i + " est un nombre premier")
```

```
Else
  TextWindow.WriteLine(i + " n'est pas un nombre premier")
EndIf

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

La routine *PrimeCheck* prend la valeur de *i* et essaie de la diviser par un nombre plus petit. Si un nombre divise *i* et qu'il n'y a pas de reste à la division, alors *i* n'est pas un nombre premier. Si c'est le cas, la routine met la valeur de *estPremier* à Faux et quitte le programme. Si le nombre est indivisible par un nombre plus petit, *estPremier* reste à Vrai.

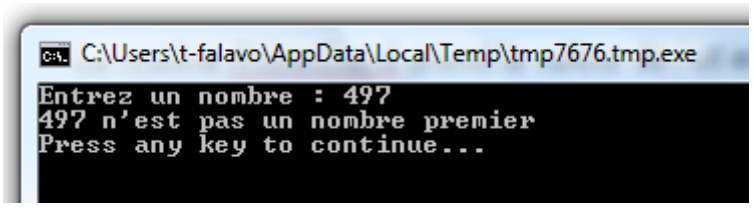

Figure 46 – Nombre premier

Maintenant que nous avons une routine qui teste si un nombre est premier ou pas, nous pouvons nous intéresser à lister tout les nombres premiers, disons de 0 à 100. Il est très facile de modifier le programme ci-dessus et de faire appel à *PrimeCheck* dans une boucle. Grâce à cette boucle, la routine sera appelée avec une valeur à chaque fois différente en paramètre. Voyons un peu ce qui est fait dans l'exemple ci-dessous :

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

Sub PrimeCheck
  For j = 2 To Math.SquareRoot(i)
```

```
    If (Math.Remainder(i, j) = 0) Then
      estPremier = "False"
      Goto EndLoop
    EndIf
  Endfor
EndLoop:
EndSub
```

Dans le programme ci-dessus, la valeur de *i* est mise à jour à chaque fois que l'on refait la boucle. Dans la boucle, un appel à la routine *PrimeCheck* est fait. La routine prend donc la valeur de *i* et calcule si elle correspond ou non à un nombre premier. Le résultat est ensuite stocké dans *estPremier* qui est accédé par la boucle hors de la routine. *i* est alors imprimé à l'écran si *estPremier* est vrai. Or comme la boucle va de 3 à 100, on obtient à l'affichage la liste des nombres premiers de 3 à 100 (Voir résultat ci-dessous).

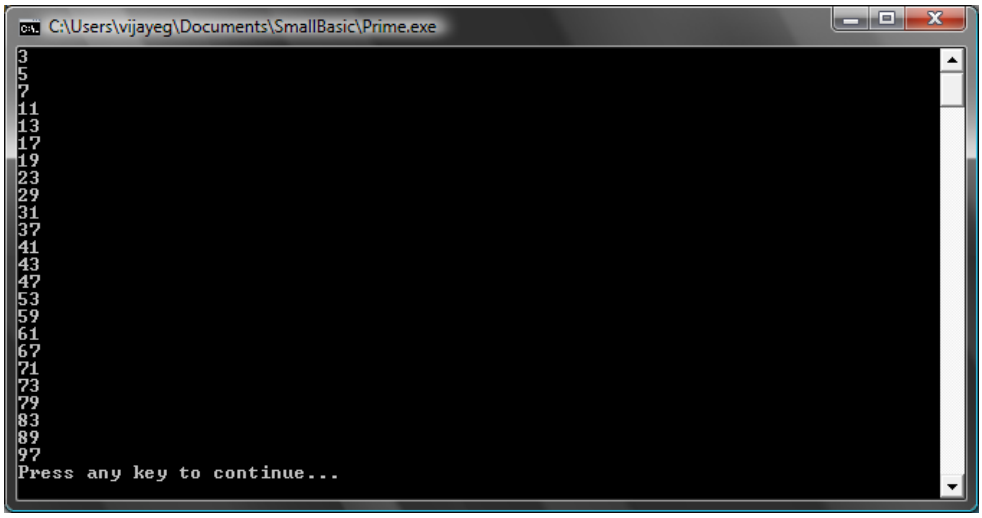

Figure 47 – Nombre Premier

Chapitre 10

Les tableaux

Vous devriez à présent être plutôt à l'aise avec l'emploi des variables, après tout vous en êtes déjà arrivé à ce chapitre et continuez de vous amuser.

Reprenons pour l'instant le premier programme d'utilisation de variables que nous avons écrit :

```
TextWindow.Write("Entrez votre nom : ")
nom = TextWindow.Read()
TextWindow.WriteLine("Bonjour " + nom)
```

Dans ce programme, nous saisissons et stockons le nom de l'utilisateur dans une variable nommée **nom**. Puis nous affichons « Bonjour » à l'utilisateur. Maintenant, imaginons qu'il y ait plusieurs utilisateurs, 5 par exemple. Comment pouvons-nous stocker tous leurs noms ? Voila une manière de le faire :

```
TextWindow.Write("Utilisateur1, entrez votre nom : ")
nom1 = TextWindow.Read()
TextWindow.Write("Utilisateur2, entrez votre nom : ")
nom2 = TextWindow.Read()
TextWindow.Write("Utilisateur3, entrez votre nom :")
nom3 = TextWindow.Read()
TextWindow.Write("Utilisateur4, entrez votre nom :")
nom4 = TextWindow.Read()
TextWindow.Write("Utilisateur5, entrez votre nom :")
nom5 = TextWindow.Read()
```

```
TextWindow.Write("Bonjour ")
TextWindow.Write(nom1 + ", ")
TextWindow.Write(nom2 + ", ")
TextWindow.Write(nom3 + ", ")
TextWindow.Write(nom4 + ", ")
TextWindow.WriteLine(nom5)
```

Voici le résultat d'exécution du code précédent :

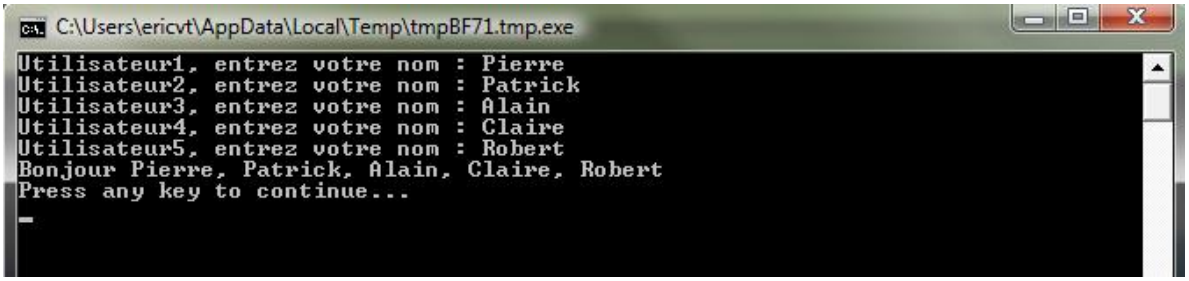

Figure 48 – Sans utilisation de tableaux

N'existe t-il pas un meilleur procédé pour écrire un programme aussi simple ? Surtout qu'en sachant qu'un ordinateur est doué pour les taches répétitives, pourquoi faudrait-il avoir à répéter le même code pour chaque utilisateur ? L'astuce dans cet exemple est de stocker et d'accéder plusieurs noms d'utilisateur en utilisant la même variable. Dans le cas échéant on pourrait utiliser la boucle **For** que nous avons apprise au chapitre précédent. Voilà comment les tableaux vont pouvoir nous aider.

Qu'est ce qu'un tableau ?

Un tableau est un type de variable spécial qui peut contenir plusieurs valeurs à la fois. Cela signifie qu'au lieu d'avoir à créer **nom1, nom2, nom3, nom4** et **nom5** afin de stocker cinq noms d'utilisateur, nous pouvons utiliser **nom** pour stocker les cinq noms. La façon dont plusieurs valeurs sont stockées s'effectue grâce à ce que l'on appelle un « index ». Par exemple **nom[1], nom[2], nom[3], nom[4]** et **nom[5]** peuvent tous stocker une variable. Les nombres 1, 2, 3, 4, 5 sont appelés indices de tableaux.

Même si **nom[1], nom[2], nom[3], nom[4]** et **nom[5]** semblent être des variables différentes, elles représentent en réalité une seule et même variable. Quel en est donc l'avantage ? La raison principale de stocker des valeurs dans un tableau est de pouvoir spécifier l'index en utilisant une autre variable, ce qui nous permet d'accéder facilement aux tableaux dans des boucles.

A présent, voyons comment mettre en pratique notre nouvelle connaissance en récrivant notre programme précédent à l'aide de tableaux.

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

```
  TextWindow.Write("Utilisateur" + i + ", entrez votre nom : ")
  nom[i] = TextWindow.Read()
EndFor

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

Ce bloque de code n'est-il pas plus facile à lire? Notez bien les deux lignes surlignées. La première stocke une valeur dans le tableau et la seconde lit la valeur à partir du tableau. La valeur que vous stockez dans **nom[1]** ne sera pas affectée par celle que vous stockez dans **nom[2]**. De ce fait vous pouvez traiter **nom[1]** et **nom[2]** comme deux variables différentes avec la même identité.

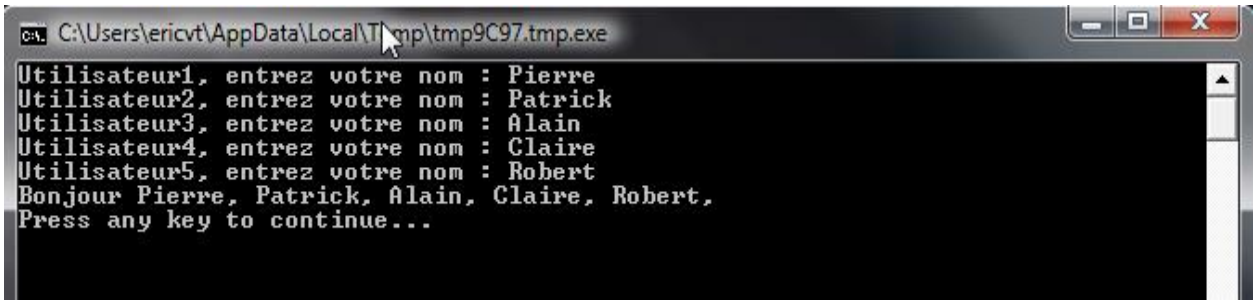

Figure 49 - Utilisation de tableaux

Le programme ci-dessus nous donne presque le même résultat que pour celui n'utilisant pas les tableaux, à l'exception de la virgule suivant le nom « Robert ». Il est possible de rectifier cela en récrivant la boucle d'affichage de la sorte :

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

Indexage de tableau

Dans notre programme précédent nous avons utilisé des nombres comme indice pour stocker et accéder aux valeurs du tableau. En fait, ces indices ne sont pas restreints à être uniquement des nombres et en pratique il est très utile de pouvoir utiliser du texte. Par exemple, dans le programme suivant, nous saisissons et stockons les éléments d'information d'un utilisateur et par la suite affichons l'information que l'utilisateur demande.

```
TextWindow.Write("Entrez votre nom : ")
utilisateur["nom"] = TextWindow.Read()
TextWindow.Write("Entrez votre age : ")
utilisateur["age"] = TextWindow.Read()
TextWindow.Write("Entrez votre ville : ")
utilisateur["ville"] = TextWindow.Read()
TextWindow.Write("Entrez votre code postal : ")
utilisateur["code"] = TextWindow.Read()

TextWindow.Write("Quelle information desirez-vous? ")
index = TextWindow.Read()
TextWindow.WriteLine(index + " = " + utilisateur[index])
```

Figure 50 – Utilisation d'index non numérique

Multiple dimension

Imaginons que nous voulions stocker le nom et le numéro de téléphone de tous nos amis et ensuite être capable de rechercher leurs numéros de téléphone quand nous le désirons. Un annuaire téléphonique, en quelque sorte… Comment pouvons-nous écrire un tel programme ?

Tout comme les variables régulières, les indices de tableaux ne respectent pas la casse.

Dans le cas suivant deux paires d'indices (aussi connues sous le nom de dimension de tableaux) sont utilisées. Admettons que nous identifions chaque ami par leur surnom. Ceci représentera le premier

index du tableau. Apres avoir utilisé le premier index pour récupérer la variable correspondant à votre ami, le deuxième des indices, **nom** et **Téléphone** nous aidera à accéder au nom et numéro de téléphone de cet ami.

La façon de stocker cette donnée se présente sous la forme suivante :

```
amis["Rob"]["Nom"] = "Robert"
amis["Rob"]["Téléphone"] = "555-6789"

amis["VJ"]["Nom"] = "Vijaye"
amis["VJ"]["Téléphone"] = "555-4567"

amis["Ash"]["Nom"] = "Ashley"
amis["Ash"]["Téléphone"] = "555-2345"
```

Du fait que nous utilisons deux indices pour le même tableau “amis”, ce tableau est qualifié de tableau à deux dimensions.

Une fois ce programme écrit, nous pouvons utiliser le surnom d’un ami et afficher les informations que nous avons stockées à son sujet. Voici donc le programme complet effectuant cette opération :

```
amis["Rob"]["Nom"] = "Robert"
amis["Rob"]["Téléphone"] = "555-6789"

amis["VJ"]["Nom"] = "Vijaye"
amis["VJ"]["Téléphone"] = "555-4567"

amis["Ash"]["Nom"] = "Ashley"
amis["Ash"]["Téléphone"] = "555-2345"

TextWindow.Write("Entrez votre surnom : ")
surnom = TextWindow.Read()

TextWindow.WriteLine("Nom : " + friends[surnom]["Nom"])
TextWindow.WriteLine("Téléphone : " + friends[surnom][" Téléphone"])
```

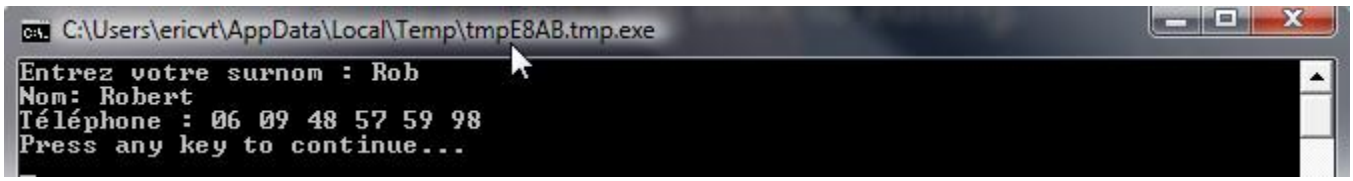

Figure 51 – Un simple annuaire téléphonique

Utilisation de tableau pour représenter des grilles informatiques

L'une des utilisations fréquentes de tableaux multidimensionnels est celle de représentation de grilles informatique ou de tables. Les grilles ont des rangées et des colonnes qui peuvent être définies par un tableau à deux dimensions. Voici un programme simple qui positionne des carrés dans une grille :

```
rangées = 8
colonnes = 8
taille = 40

For r = 1 To rangées
  For c = 1 To colonnes
    GraphicsWindow.BrushColor = GraphicsWindow.GetRandomColor()
    carrés[r][c] = Shapes.AddRectangle(taille, taille)
    Shapes.Move(boxes[r][c], c * taille, r * taille)
  EndFor
EndFor
```

Ce programme ajoute des carrés et les positionne pour former une grille de 8 par 8. De plus, il stocke aussi ces carrés dans un tableau qui permettra par la suite de les répertorier et de les réutiliser selon notre volonté.

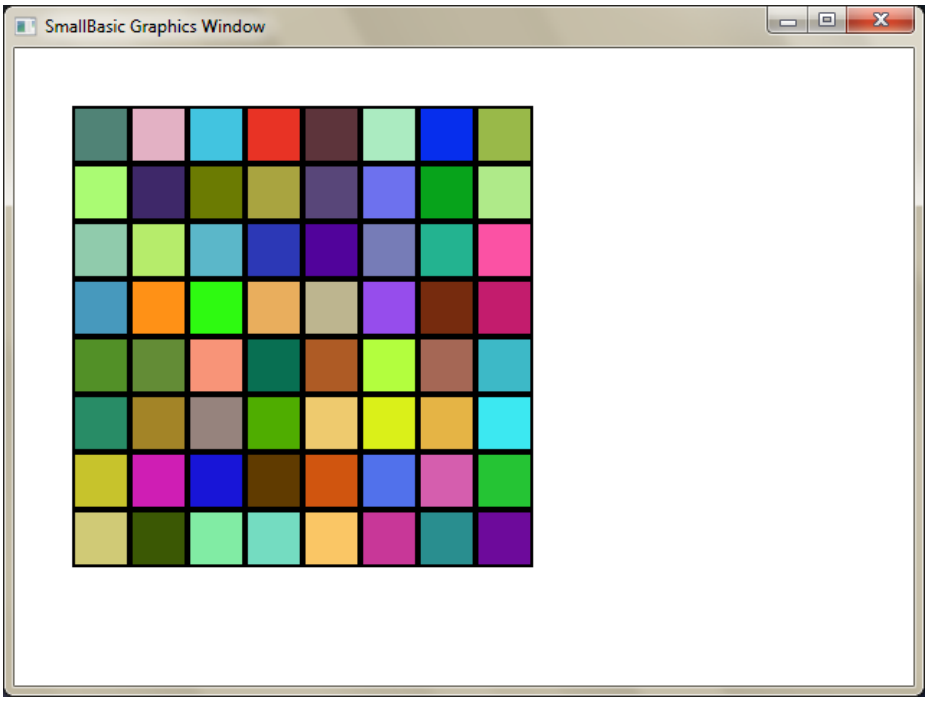

Figure 52 –Positionnement des carrés dans une grille

Par exemple, l'ajout du code suivant au programme précédent nous permettra de déplacer ces carrés dans le coin en haut à gauche.

```
For r = 1 To rangées
  For c = 1 To colonnes
    Shapes.Animate(carrés[r][c], 0, 0, 1000)
    Program.Delay(300)
  EndFor
EndFor
```

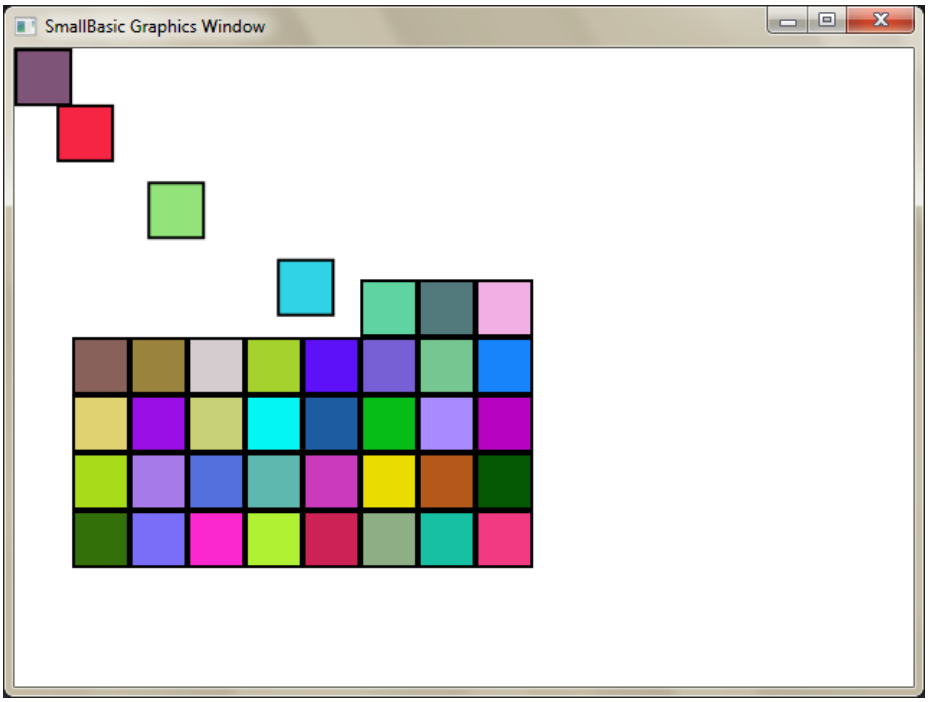

Figure 53 – Triage des carrés dans la grille

Chapitre 11

Événements et interactivité

Dans les deux premiers chapitres, nous avons introduit les objets qui possèdent des propriétés et des opérations. En plus de celles-ci, certains objets ont ce qu'on appelle des événements. Les événements sont comme des signaux appelés, par exemple, en réponse à une action de l'utilisateur, comme le mouvement de la souris, ou bien un clic. Dans un certain sens, les événements sont l'opposé des opérations. Dans le cas d'une opération, on demande à l'ordinateur de faire quelque chose ; alors que dans le cas d'un événement, c'est l'ordinateur qui vous indique que quelque chose d'intéressant vient de se passer.

En quoi sont utiles les événements ?

Les événements sont la clé de l'interactivité dans un programme. Si vous voulez donner la possibilité à l'utilisateur d'interagir avec votre programme, ce sera grâce aux événements. Imaginons que vous êtes en train de programmer un jeu de morpion. Vous voudriez sûrement commencer par proposer à l'utilisateur de choisir s'il veut les ronds ou les croix n'est-ce pas ? C'est là qu'interviennent les événements – vous recevez les données entrées par l'utilisateur grâce aux événements. Si cela vous semble dur à comprendre, nous allons prendre un exemple simple qui vous montrera ce que sont les événements et comment les utiliser.

Ci-dessous se trouve un exemple simple avec une instruction et une routine. La routine utilise l'opération *ShowMessage* sur l'objet *graphicsWindow* pour faire apparaître une boîte de dialogue à l'utilisateur.

```
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
```

```
  GraphicsWindow.ShowMessage("Vous avez cliqué.", "Bonjour")
EndSub
```

La partie intéressante à noter dans le programme ci-dessus est la ligne où l'on assigne le nom de la routine à l'événement *MouseDown* de l'objet *GraphicsWindow*. Vous pouvez d'ailleurs remarquer que *MouseDown* est très similaire à une propriété – excepté qu'au lieu d'assigner une valeur, nous lui assignons la routine *OnMouseDown*. Voilà tout ce qu'il y a de spécial avec les événements – quand l'événement est appelé, la routine s'enclenche automatiquement. Dans ce cas précis, la routine *OnMouseDown* est appelée à chaque fois que l'utilisateur clique avec sa souris sur le *GraphicsWindow*. Allez-y, lancez le programme et essayez vous-même. A chaque fois que vous cliquerez sur le *GraphicsWindow* avec votre souris, vous verrez une boîte de dialogue apparaître comme sur l'image ci-dessous.

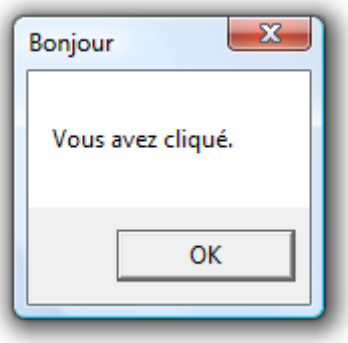

Figure 54 – Réponse d'un événement

Cette façon de manipuler les événements est très pratique et vous permet de faire des programmes créatifs et intéressants. Les programmes écrits de cette manière sont appelés « programmes orientés événements ».

Vous pouvez modifier la routine *OnMouseDown* pour qu'elle fasse apparaître autre chose qu'une boîte de dialogue. Par exemple, l'exemple ci-dessous illustre comment faire pour dessiner de gros points bleu là où l'utilisateur clique avec sa souris :

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

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

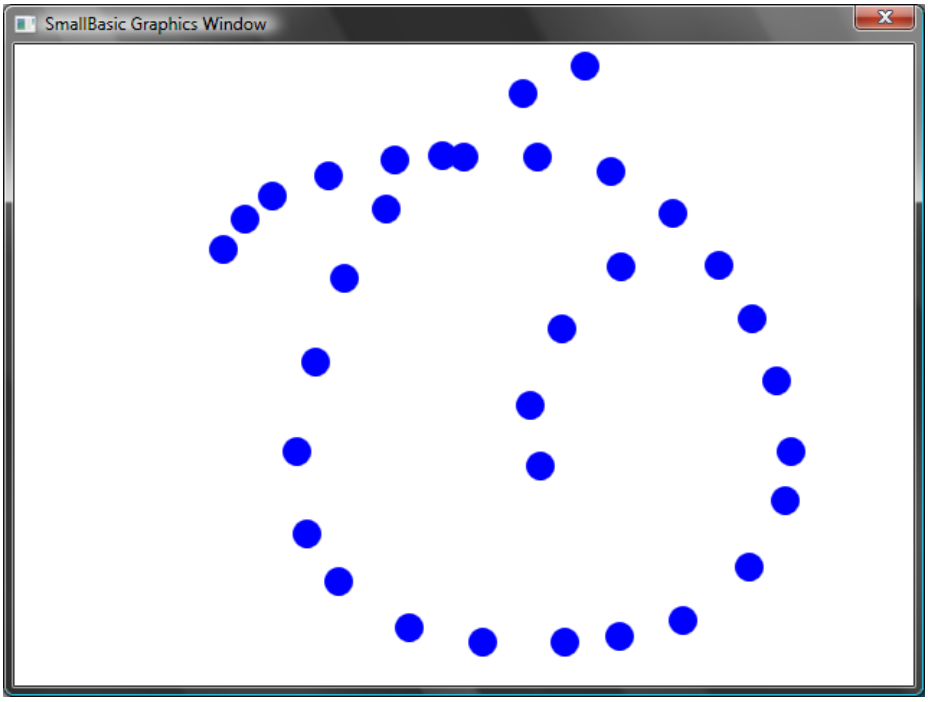

Figure 55 – Manipulation de l'événement MouseDown

Notez que dans l'exemple ci-dessus, nous utilisons *MouseX* et *MouseY* pour connaître les coordonnées de la souris. Nous les utilisons ensuite pour dessiner un cercle centré sur les coordonnées.

Manipulation de plusieurs événements

Il n'y a pas de limite au nombre d'événements que vous pouvez manipuler. Vous pouvez même avoir une routine qui manipulera plusieurs événements. Toutefois, vous ne pourrez manipuler un événement qu'une fois. Si vous essayez d'assigner deux routines au même événement, seule la seconde routine sera prise en compte.

Pour expliquer cela, prenons le dernier exemple présenté et ajoutons une nouvelle routine pour manipuler la pression des touches. Nous allons aussi créer une routine supplémentaire qui change la couleur du cercle, pour qu'à chaque fois que l'on clique avec la souris, nous ayons des cercles de différentes couleurs.

```
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
```

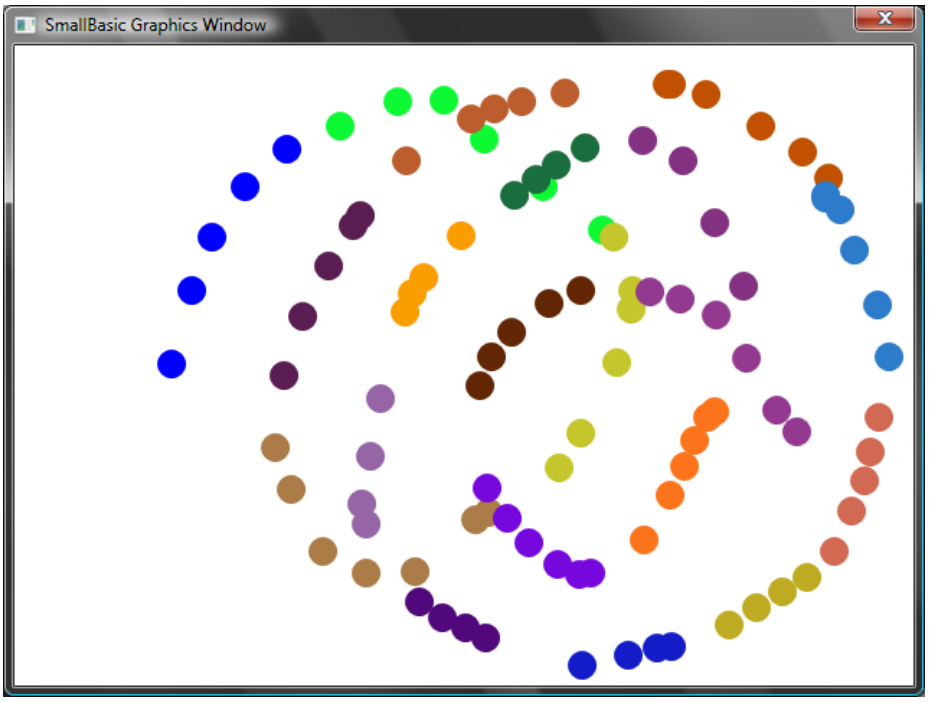

Figure 56 – Manipulation de plusieurs événements

Si vous lanciez ce programme et que vous cliquiez sur la fenêtre, vous verriez apparaître des points bleus. Maintenant, si vous appuyez sur une touche de votre clavier puis que vous cliquez à nouveau, vous aurez des points d'une autre couleur. Lorsque vous appuyez sur une touche, la routine *OnKeyDown* est exécutée, ce qui change la couleur de votre pinceau aléatoirement. Après cela si vous cliquez avec votre souris, un cercle d'une autre couleur apparaît.

Le programme Paint

Armés de nos événements et de nos routines, nous pouvons maintenant écrire un programme qui permette à l'utilisateur de dessiner sur notre fenêtre comme Paint le fait. Il est étonnamment facile d'écrire ce programme, à partir du moment où l'on arrive à découper le problème en sous-problèmes. Dans un premier temps, écrivons un programme qui permette à l'utilisateur de bouger sa souris n'importe où dans la fenêtre graphique, en laissant une trace de son passage derrière elle.

```
GraphicsWindow.MouseMove = OnMouseMove
```

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

Si vous exécutez le programme, vous remarquerez que la première ligne commence toujours du coin en haut a gauche de la fenêtre (0,0). Nous pouvons régler ce problème en manipulant l'événement *MouseDown* et en utilisant les valeurs *prevX* et *prevY* quand l'événement est appelé.

Nous souhaitons que la trace apparaisse derrière le curseur seulement lorsque le bouton gauche de la souris est cliqué. Pour avoir ce comportement, nous utiliserons la propriété *IsLeftButtonDown* sur l'objet *Mouse*. Cette propriété nous indique si c'est le bouton gauche de la souris qui est cliqué ou non. Si sa valeur est *True*, nous dessinerons la ligne, sinon nous omettrons la ligne.

```
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
```

Annexe A

Exemples sympas

La tortue fractale

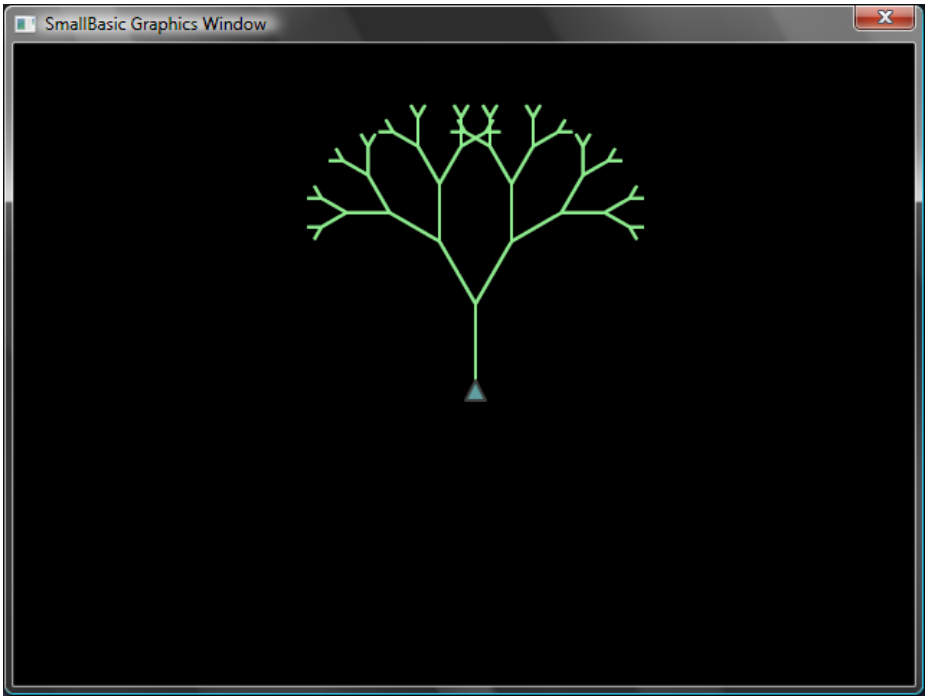

Figure 57 – La tortue dessinant un arbre

```
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
```

Récupération de photos depuis flickr

Figure 58 – L'application récupérant des photos depuis flickr

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

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

Changement dynamique du fond d'écran

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

Casse briques

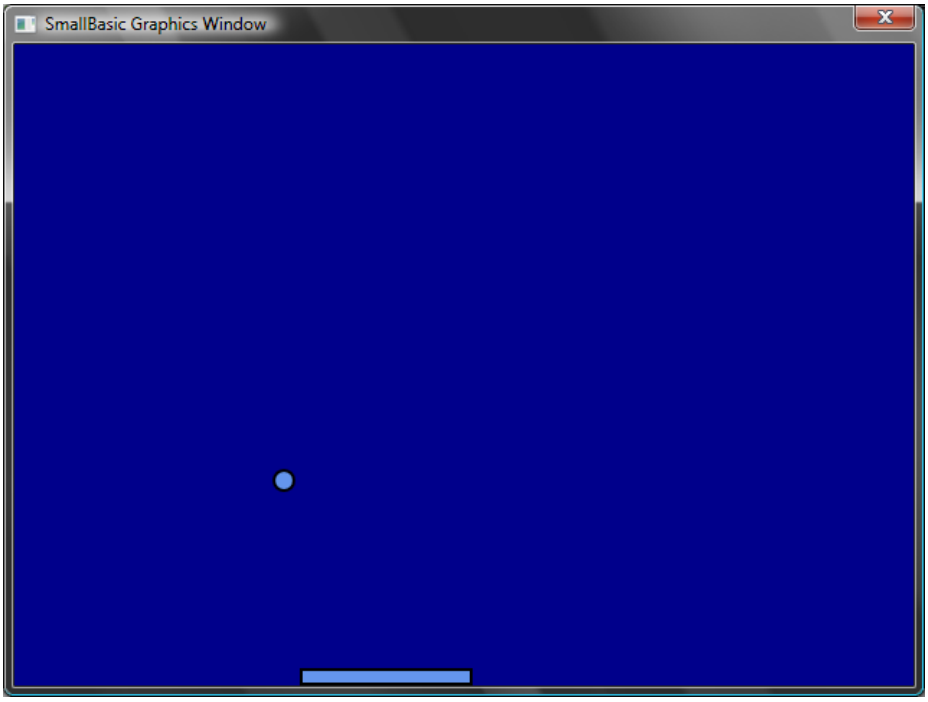

Figure 59 - Casse briques

```
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("Vous avez perdu", "Casse briques")

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

Annexe B

Couleurs

Ceci est une liste des couleurs qui sont supportées par SmallBasic. Elles sont triées et regroupées par teintes similaires. Les noms des couleurs sont des mots-clés et sont donc en anglais.

Rouge

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

Rose

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

Orange

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

Jaune

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

PeachPuff	#FFDAB9
PaleGoldenrod	#EEE8AA
Khaki	#F0E68C
DarkKhaki	#BDB76B

Violet

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

Vert

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

Bleu

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

Marron

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

Blanc

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

Gris

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