Microsoft Small Basic

Wprowadzenie do programowania

Rozdział 1

Wstęp

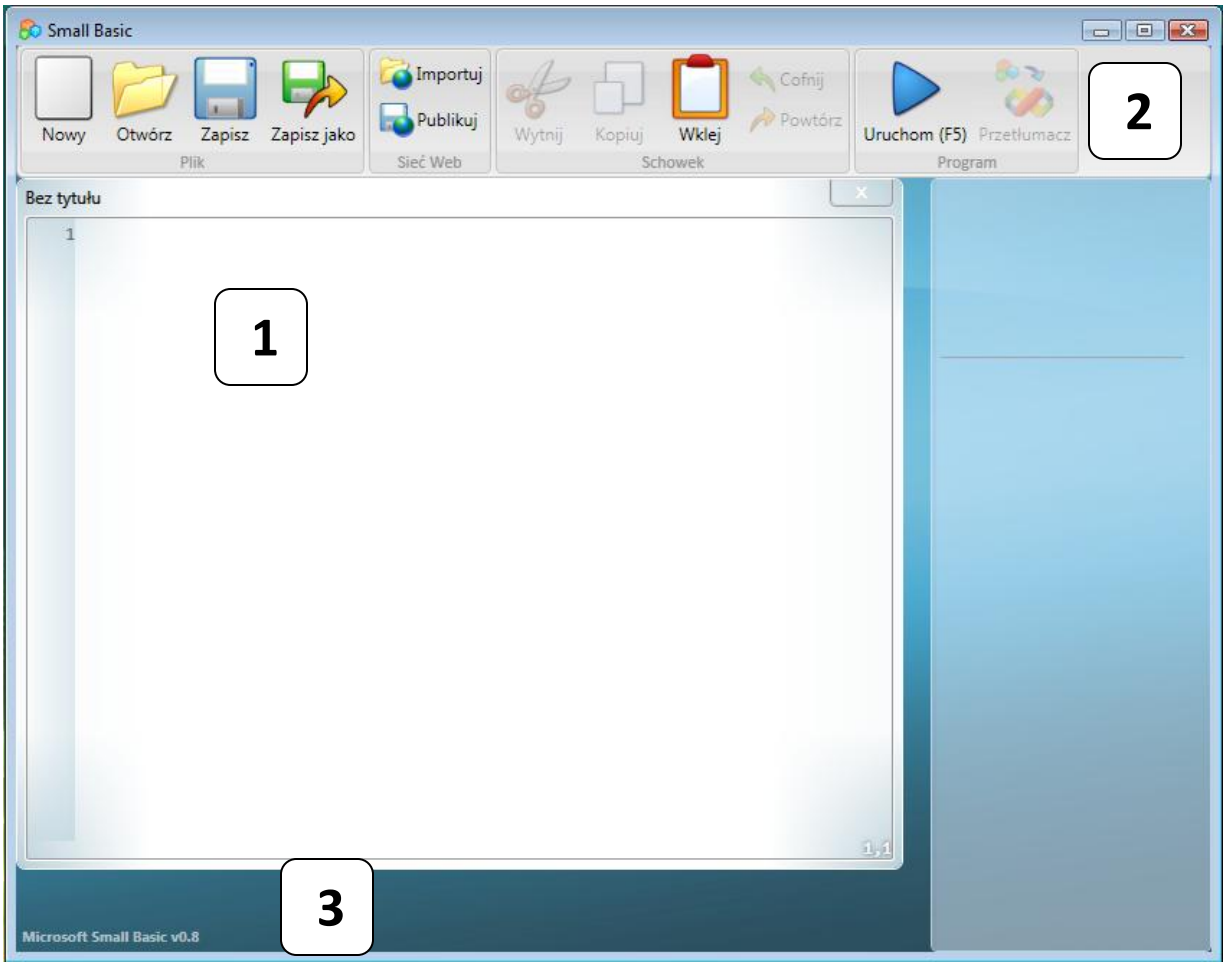
Small Basic i programowanie

Programowanie to proces, w którym dzięki językowi programowania tworzy się program komputerowy. Tak jak z ludźmi rozmawia się w różnych językach (od polskiego, poprzez angielski aż po chiński), tak i do komputerów trzeba się zwracać w pewnej ustalonej formie, zwanej językiem programowania. Na samym początku historii komputerów, liczba tych języków była stosunkowo niewielka a ich nauczenie się nie było trudne, jednak w miarę rozwoju informatyki, języki stawały się coraz bardziej złożone. W końcowym efekcie, nowoczesne języki programowania stały się tak trudne, że ich poznanie przez osobę zaczynającą dopiero przygodę z komputerami staje się dużym wyzwaniem. Można wręcz stwierdzić, że w obecnych czasach, programowanie jest tak trudne, że odstrasza osoby nie planujące rozwoju w zawodzie programisty.

Small Basic jest językiem programowania, który został zaprojektowany i stworzony po to, żeby pisanie programów było bardzo proste, dostępne i dające dużo radości, nawet dla początkujących. Założeniem języka jest zlikwidowanie bariery, która powoduje, że programowanie powszechnie postrzegane jest jako coś trudnego i dostępnego tylko dla wtajemniczonych.

Środowisko

Przed wgłębieniem się w sam język, warto poznać środowisko Small Basic. Po instalacji i pierwszym uruchomieniu, otwiera się okno widoczne na ilustracji:



Rysunek 1 – Środowisko Small Basic

Jest to środowisko Small Basic, w którym tworzy się i uruchamia program. Środowisko to, składa się z kilku istotnych części:

Edytor, oznaczony na rysunku numerem 1, jest miejscem, w którym wpisuje się program. Po otwarciu już zapisanego programu, właśnie w tym miejscu można go przejrzeć i edytować. Poza tym, można oczywyście program z edytora zapisać na dysku tak, aby móc do niego wrócić później.

Środowisko umożliwia również pracę z więcej niż jednym programem na raz. Każdy z programów ma wtedy własne okno edytora, a to z okien, w którym w danej chwili pracuje programista nazywane jest oknem aktywnym.

Pasek narzędzi, oznaczony na rysunku numerem 2, jest używany do wydawania poleceń aktywnemu oknu edytora lub całemu środowisku Small Basic. Zastosowanie poszczególnych poleceń przedstawione zostanie w dalszej części dokumentu.

Powierzchnia, oznaczona numerem 3, jest miejscem, w którym otwierają się wszystkie okna edytora.

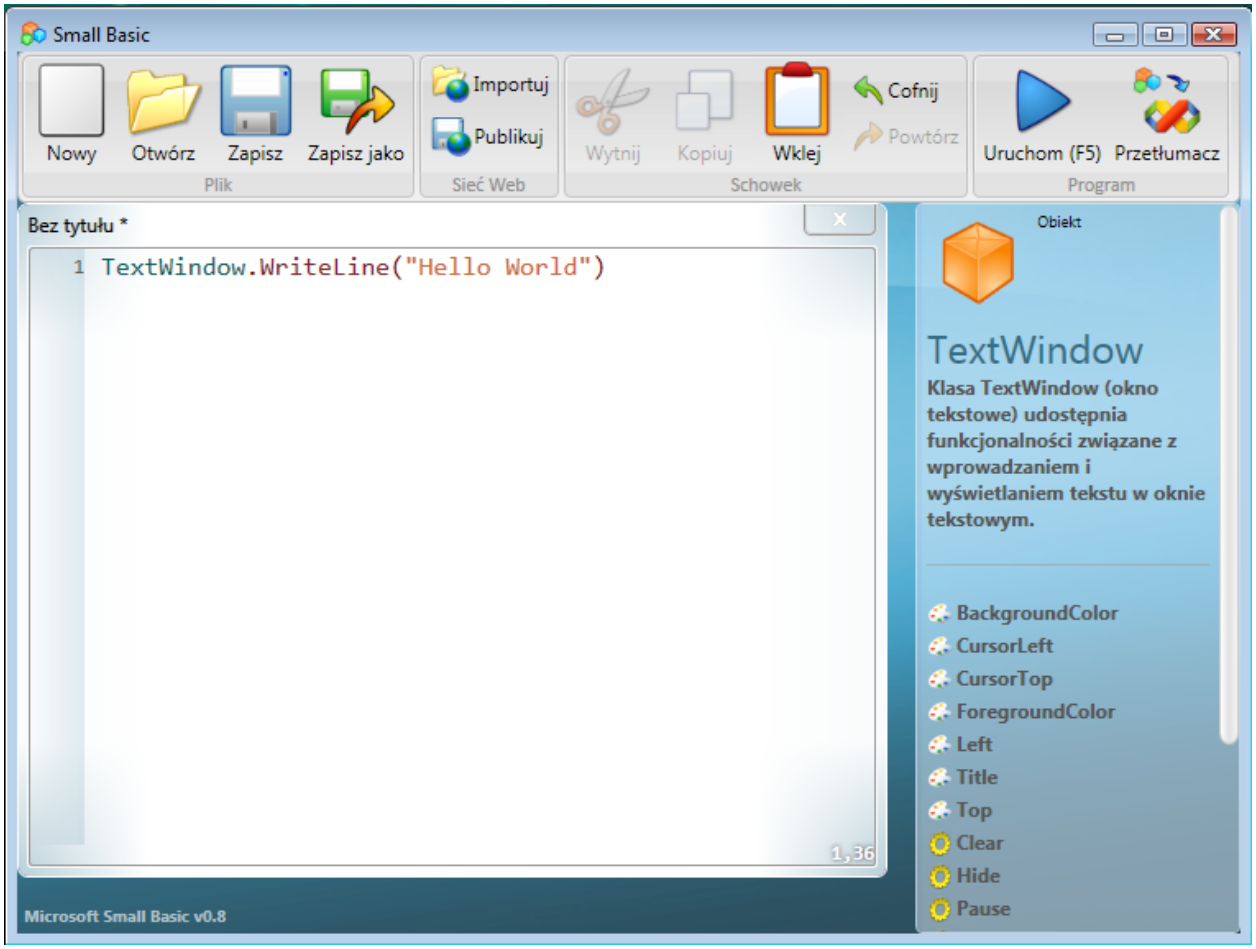
Pierwszy program

Teraz, znając już w ogólnym zarysie środowisko, można zacząć programować. Jak wspomniano wcześniej, program pisze się w edytorze. Można więc wprowadzić następującą linię:

Tekst Hello World oznacza "witaj świecie". Nie jest to może bardzo sensowne, ale niemal odkąd istnieje programowanie – właśnie ten tekst każdy programista chce umieścić w swoim pierwszym programie.

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

To jest pierwszy program w języku Small Basic. Jeżeli polecenie zostało wprowadzone poprawnie, to widok na ekranie powinien być mniej więcej taki:



Rysunek 2 – Pierwszy program

Teraz, gdy program jest wpisany, można go uruchomić, żeby zobaczyć co się stanie. W środowisku Small Basic, program uruchamia się przez użycie przycisku "Uruchom" na pasku narzędzi albo przez naciśnięcie

F5 na klawiaturze. Jeżeli wszystko jest wpisane poprawnie, efektem działania programu będzie okno takie, jak na ilustracji poniżej.

Rysunek 3 – Wynik działania programu

Gratulacje! Pierwszy program w języku Small Basic został właśnie napisany i uruchomiony. Program jest oczywiście bardzo prosty, ale tak czy inaczej – to pierwszy krok w kierunku prawdziwego programowania. Teraz, aby tworzyć bardziej zaawansowane programy, trzeba tylko zrozumieć, co tak naprawdę się stało – jakie polecenia zostały wydane komputerowi i skąd komputer wiedział co z nimi zrobić. W kolejnym rozdziale, pierwszy program zostanie przeanalizowany, więc jego zrozumienie nie powinno być trudne.

Podczas wprowadzania programu, można zauważyć wyskakujące okna z podpowiedziami, takie jak na rysunku 4. Mechanizm ten nazywany jest "intellisense" i pomaga programiście pisać program szybciej. Używając klawiszy góra/dół, można prosto przechodzić pomiędzy wyświetlonymi podpowiedziami i naciskając Enter – wstawić całą podpowiedź do programu.

Rysunek 4 - Intellisense

Zapisywanie programu

Jeżeli programista zechce zakończyć pracę ze środowiskiem Small Basic, ale myśli o tym, aby kiedyś wrócić do swoich programów – powinien je zapisać. Tak naprawdę, warto zapisywać program nie tylko zamykając środowisko, ale również co jakiś czas w trakcie pracy, ponieważ dzięki temu, nawet w przypadku awarii czy wyłączenia zasilania, nic nie zginie. Program można zapisać używając przycisku "Zapisz" z paska narzędzi lub naciskając na klawiaturze kombinację Ctrl+S (naciskając S z przytrzymanym przyciskiem Ctrl).

Rozdział 2

Zrozumienie pierwszego programu

Czym tak naprawdę jest program komputerowy?

Mówiąc najprościej – program jest zestawem instrukcji dla komputera. Instrukcje te bardzo dokładnie "mówią" komputerowi, co powinien zrobić i komputer zawsze instrukcje te wykonuje. Oczywiście, tak jak w przypadku ludzi, komputer może wykonać instrukcje wydane w języku, który jest zrozumiały. Język taki nazywany jest właśnie językiem programowania. Istnieje ogromna ilość języków programowania a Small Basic jest po prostu jednym z nich.

W normalnej rozmowie dwóch osób używa się słów, z których składa się zdania, dzięki którym konkretna informacja przekazywana jest od mówiącego do słuchającego. Podobnie jest w językach programowania. Zestawy specjalnych (pochodzących najczęściej z języka angielskiego) słów, przekazują informację komputerowi. Program komputerowy praktycznie zawsze jest zbiorem takich zestawów, które mają pewien sens tak dla programisty, który je wpisał jak i dla komputera, który ma z nich skorzystać.

Istnieje wiele języków zrozumiałych dla komputera. Java, C++, Python, VB, C# i inne są nowoczesnymi i uniwersalnymi językami. Są one powszechnie stosowane do pisania tak prostych jak i bardzo złożonych programów.

Programy w Small Basic

Typowy program w języku Small Basic składa się z zestawu instrukcji. Każda linia w programie jest instrukcją, a każda instrukcja – pewnego rodzaju poleceniem dla komputera. Gdy komputer musi wykonać program w języku Small Basic, wczytuje program i odczytuje polecenie z pierwszej linii. Po zrozumieniu polecenia, komputer je wykonuje, a gdy wykona – odczytuje kolejną linię programu. Proces jest kontynuowany i wykonanie programu kończy się, gdy komputer dotrze do ostatniej linii.

Pierwszy program

Pierwszy program, opisany kilka stron wcześniej, wyglądał następująco:

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

Program jest bardzo prosty i składa się z jednego polecenia. Polecenie to nakazuje komputerowi wyświetlenie w oknie tekstowym linii tekstu zawierającej **Hello World**.

Można to przetłumaczyć, jako zrozumiałe dla komputera polecenie:

```
Napisz Hello World
```

Można zauważyć, że polecenia języka programowania dają się podzielić na części, podobnie jak zdania dzielą się na słowa. W pierwszym poleceniu występują trzy różne części:

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

Znak kropki, nawiasy oraz cudzysłów są specjalnymi znakami, które są konieczne, aby komputer był w stanie zrozumieć intencje programisty.

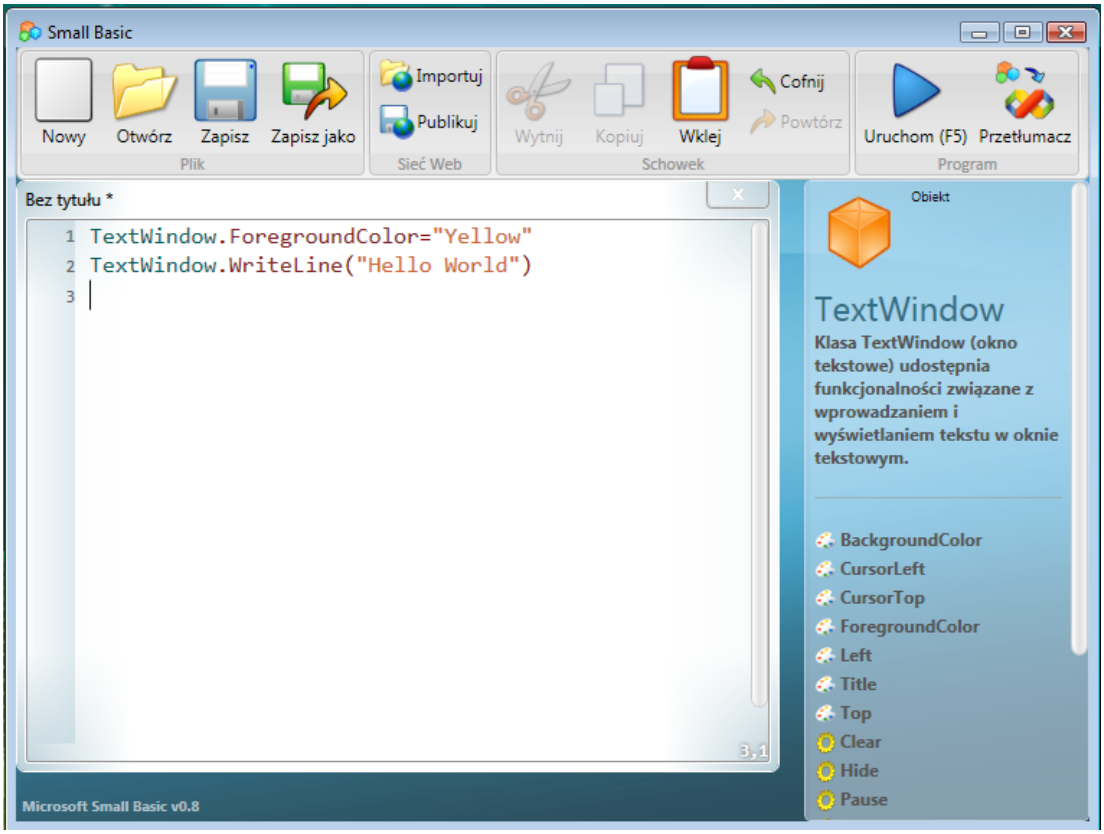
Po uruchomieniu programu pojawia się czarne okno zawierające tekst. Nazywane jest ono oknem tekstowym (text window) lub konsolą. Pojawia się w nim efekt działania programu. W programie, **TextWindow** jest *obiektem* a poza oknem tekstowym istnieje jeszcze wiele innych obiektów, których można użyć w programach. Na obiektach można wykonywać pewne operacje. Przykładowo, w pierwszym programie wykorzystano operację WriteLine (zapisz linię), dla której w nawiasie przekazano tekst, który miał zostać wyświetlony. Tekst przekazany do wyświetlenia przez WriteLine nazywany jest parametrem (argumentem) dla operacji. Istnieją operacje, które używają jednego parametru, takie które używają wielu parametrów ale również i takie, które nie używają żadnego.

Znaki interpunkcyjne takie jak cudzysłów, nawiasy, czy kropki są bardzo istotną częścią programów. Zmiana ich pozycji lub ilości może znacząco zmienić sposób, w jaki computer "zrozumie" program.

Drugi program

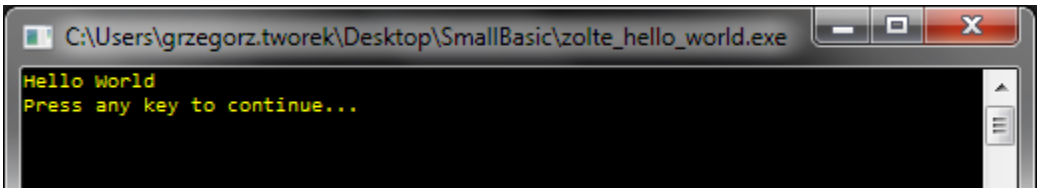
Teraz, gdy pierwszy program stał się bardziej zrozumiały, można go nieco urozmaicić, dodając mu trochę kolorów.

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



Rysunek 5 – Dodanie kolorów

Gdy uruchomi się powyższy program, można zauważyć, że w oknie tekstowym wyświetlany jest ten sam tekst "Hello World" ale tym razem jest on żółty zamiast jasnoszary jak poprzednio.



Rysunek 6 - Hello World na żółto

Warto zwrócić uwagę, jaka instrukcja została dodana do programu. Używa ona nowego słowa *ForegroundColor*, do którego, po znaku równości dodano słowo "*Yellow*". Polecenie takie, dla programisty oznacza *przypisanie* wartości "*Yellow*" do właściwości *ForegroundColor* obiektu *TextWindow*. Warto tutaj zwrócić uwagę na różnicę pomiędzy właściwością a opisaną wcześniej operacją. Właściwość nie ma żadnych nawiasów, natomiast można do niej przypisać pewną wartość. Oznacza to mniej więcej "niech ForegroundColor będzie równe Yellow". Jeżeli programista chciałby użyć koloru innego niż żółty, to ma do dyspozycji wiele innych wartości do przypisania zamiast Yellow. W "Dodatku B" na końcu podręcznika znajdują się nazwy wszystkich kolorów oraz przykłady tego, jak będą one wyglądać na ekranie. Należy przy tym pamiętać, żeby używając nazwy koloru użyć również znaków cudzysłowu – są one istotnym elementem języka programowania

Rozdział 3

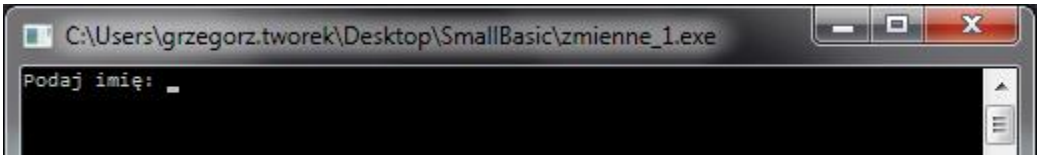
Wprowadzenie do zmiennych

Użycie zmiennych w programie

W poprzednim rozdziale powstał program piszący na ekranie "Hello World". Byłoby miło, gdyby program ten zamiast każdemu użytkownikowi wyświetlać ten sam komunikat, umiał zapytać o imię i wyświetlić coś, co będzie lepiej dostosowane do konkretnej osoby. Żeby to zrobić, program musi najpierw zapytać użytkownika o imię, zapisać je sobie gdzieś a potem wyświetlić "Witaj" a po nim – zapamiętane imię. Można to zrobić w następujący sposób:

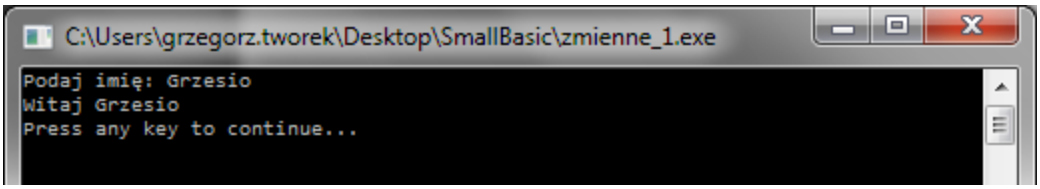
```
TextWindow.Write("Podaj imię: ")
name = TextWindow.Read()
TextWindow.WriteLine("Witaj " + name)
```

Gdy program zostanie uruchomiony, wyświetli okno z pytaniem:



Rysunek 7 – Pytanie o imię

Po wpisaniu przez użytkownika imienia i naciśnięciu Enter, program wyświetli:



Rysunek 8 – Spersonalizowane powitanie

Jeżeli program zostanie uruchomiony ponownie, po raz kolejny pojawi się to samo pytanie. Można oczywiście wpisać inne imię i program je zapamięta a później użyje w wyświetlanym powitaniu.

Analiza programu

W uruchomionym przed chwilą programie, jedna linia powinna przykuć uwagę programisty:

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

Read() (ang. czytaj) wygląda podobnie do *WriteLine()*, ale nie ma parametrów. Oczywiście jest to operacja, która mówi komputerowi, że ma odczytać to, co użytkownik wpisze i zatwierdzi klawiszem Enter. Gdy użytkownik naciśnie Enter, komputer pobiera to, co zostało wpisane i przekazuje programowi. Najciekawsze w tym miejscu jest to, że program zapamiętuje zwróconą informację w *zmiennej* o nazwie name. Zmienna powinna być tu rozumiana jako specjalne, nazwane miejsce, w którym można przechować jakąś wartość tak, żeby móc jej użyć w przyszłości.

> Write(), *podobnie jak* WriteLine() *jest operacją obiektu TextWindow. Write() pozwala na napisanie czegoś na ekranie w taki sposób, że kolejny wyświetlony tekst pojawi się w tej samej linii.*

Kolejna linia również jest bardzo ciekawa:

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

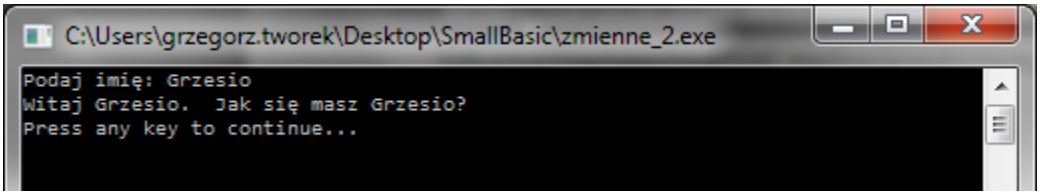
W linii tej, widać faktyczne użycie wartości przechowywanej w zmiennej o nazwie name. Powyższe polecenie pobiera wartość zmiennej, dodaje ją do słowa "Witaj" i wyświetla na ekranie, w oknie TextWindow.

Gdy zmienna ma już wartość, można ją wykorzystywać w programie wielokrotnie. Na przykład, możliwe jest następujące zastosowanie:

```
TextWindow.Write("Podaj imię: ")
name = TextWindow.Read()
```

```
TextWindow.Write("Witaj " + name + ".  ")
TextWindow.WriteLine("Jak się masz " + name + "?")
```

Po uruchomieniu takiego programu, na ekranie pokaże się następujący efekt jego działania:



Rysunek 9 – Wielokrotne użycie zmiennej

Reguły nazywania zmiennych

Zmienne w programie mają swoje nazwy i to właśnie po nazwach programista może je rozróżnić. Aby efektywnie pracować ze zmiennymi, warto stosować się do kilku prostych zaleceń dotyczących nadawania im nazw:

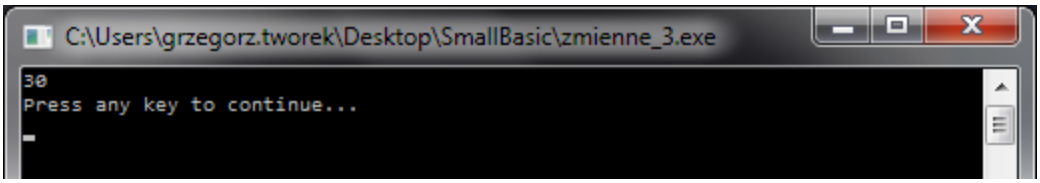
1. Nazwa powinna zaczynać się od litery,
2. Nazwa nie powinna być taka sama jak któreś ze słów kluczowych Small Basic (if, for, then, itp.),
3. Nazwa może zawierać dowolną kombinację liter, cyfr i znaków podkreślenia,
4. Wskazane jest używanie nazw, które coś znaczą. Ponieważ nazwy zmiennych mogą być dowolnie długie, warto stosować takie, które pokazują, co programista miał na myśli.
5. Często stosuje się nazwy pochodzące z języka angielskiego. Można tak robić, jednak w nazwach polskich również nie ma nic złego.

Zabawa z liczbami

W ostatnim przykładzie pokazano, jak można wykorzystać zmienną do przechowania imienia użytkownika. W kolejnych przykładach będzie można zobaczyć, jak zmienne mogą się przydać w manipulowaniu liczbami. Na początek można zacząć od bardzo prostego programu:

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

Po jego uruchomieniu, na ekranie pokaże się wynik:



Rysunek 10 – Dodawanie dwóch liczb

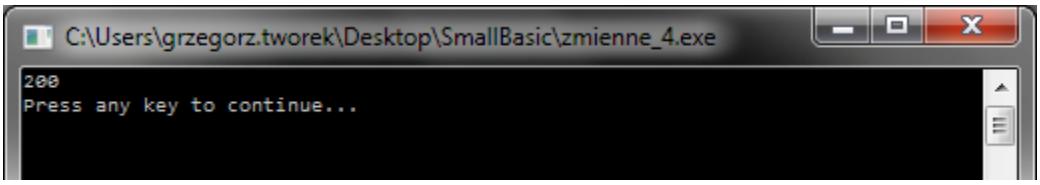
W pierwszej linii, do zmiennej number1 przypisywana jest wartość 10. W linii drugiej, do zmiennej number2 przypisywana jest wartość 20. W trzeciej linii użyta jest zmienna number3, do której przypisywana jest suma number1 i number2. Oczywiście ma ona wartość 30. Wartość ta w ostatniej linii wyświetlana jest w oknie TextWindow

Warto zwrócić uwagę, że liczby nie są ujmowane w znaki cudzysłowu. Cudzysłów jest używany tylko podczas pracy z tekstem.

Można teraz lekko zmodyfikować ostatni program, aby zobaczyć jak zmieni się jego działanie:

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

Program pomnoży przez siebie number1 i number 2, po czym zapisze wynik w zmiennej number3. Wynik jest widoczny poniższym ekranie:



Rysunek 11 – Mnożenie dwóch liczb

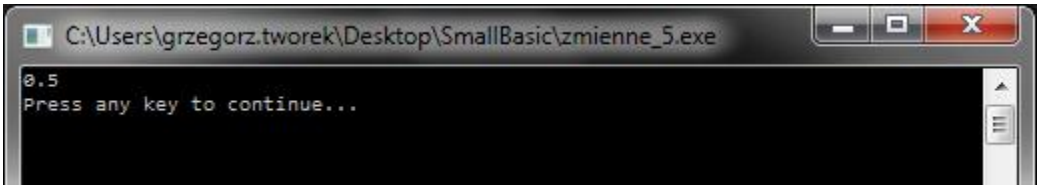
W analogiczny sposób, wartości zmiennych można od siebie odejmować:

```
number3 = number1 - number2
```

Jeżeli potrzebne byłoby dzielenie, należałoby użyć znaku "/". Najważniejsza linia programu wyglądałby wtedy jak na poniższym przykładzie:

```
number3 = number1 / number2
```

Rezultatem jego działania będzie:



Rysunek 12 – Dzielenie dwóch liczb

Prosty konwerter temperatur

Kolejny program będzie używał wzoru służącego do przeliczania stopni Fahrenheita (używanych powszechnie na przykład w USA) na stopnie Celsjusza: $℃ = \frac{5(°F-32)}{9}$.

Najpierw, program musi pobrać od użytkownika temperaturę w stopniach Fahrenheita i zapisać ją w zmiennej. Najlepiej wykorzystać do tego celu specjalną operację służącą do pobierania wartości liczbowych: **TextWindow.ReadNumber**.

```
TextWindow.Write("Podaj temperaturę w stopniach Fahrenheita: ")
fahr = TextWindow.ReadNumber()
```

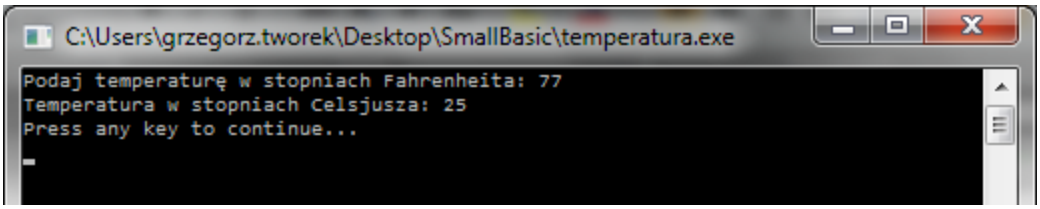
Mając już tę wartość, należy ją przeliczyć:

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

Ponieważ w programie nie da się zapisywać zwykłych ułamków, trzeba przy pomocy nawiasów wskazać, że działanie **fahr – 32** ma zostać wykonane jako pierwsze. Po wyliczeniu temperatury w stopniach Celsjusza wystarczy wyświetlić ją na ekranie. Całość powinna wyglądać mniej więcej tak:

```
TextWindow.Write("Podaj temperaturę w stopniach Fahrenheita: ")
fahr = TextWindow.ReadNumber()
celsius = 5 * (fahr - 32) / 9
TextWindow.WriteLine("Temperatura w stopniach Celsjusza: " + celsius)
```

Rezultat działania programu wygląda następująco:



Rysunek 13 – Konwerter temperatur

Rozdział 4

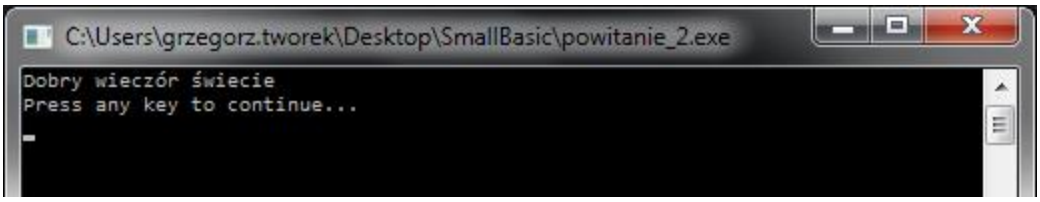
Wykonanie warunkowe

Pierwszy napisany program wyświetlał ogólne powitanie "Witaj świecie". Byłoby miło, gdyby zamiast angielskiego słowa "witaj" pojawiło się "dzień dobry" jeżeli program jest uruchomiony rano lub "dobry wieczór", jeżeli ktoś uruchomi go wieczorem. Żeby to było możliwe, program musi sprawdzić aktualną godzinę i w zależności od niej zachować się w różny sposób.

```
If (Clock.Hour < 18) Then
  TextWindow.WriteLine("Dzień dobry świecie")
EndIf
If (Clock.Hour >= 18) Then
  TextWindow.WriteLine("Dobry wieczór świecie")
EndIf
```

Uruchamiając taki program, w zależności od pory dnia na ekranie będzie można zobaczyć:

Rysunek 14 – Dzień dobry świecie



Rysunek 15 – Dobry wieczór świecie

Warto przyjrzeć się temu programowi. W pierwszych trzech liniach widać, że program sprawdza czy Clock.Hour jest mniejsze niż 18 i jeżeli tak – wyświetla wariant z "dzień dobry". Słowa **If**, **Then** oraz **EndIf** są tak zwanymi słowami kluczowymi rozumianymi przez komputer, gdy program jest wykonywany. Po słowie **If** musi wystąpić warunek i w tym przypadku jest to (**Clock.Hour < 12)**. Nawiasy dookoła warunku są niezbędne, aby komputer poprawnie odczytał intencje programisty. Po warunku, pojawia się słowo **Then**, a po nim operacje, które powinny zostać wykonane, jeżeli warunek jest spełniony. Całość kończy się słowem **Endif**, które wskazuje koniec fragmentu wykonywanego warunkowo.

W języku Small Basic można używać obiektu Clock *(zegar) aby odczytać bieżącą datę i czas. Clock udostępnia wiele właściwości takich jak Day (dzień), Month (miesiąc), Year (rok), Hour (godzina), Minutes (minuty) i Seconds (sekundy).*

Jeżeli warunek podany po słowie **If** jest spełniony, to wykonają się wszystkie instrukcje zawarte pomiędzy **Then** a **EndIf**. Na przykład, możnaby napisać taki program:

```
If (Clock.Hour < 10) Then
  TextWindow.Write("Dzień dobry. ")
  TextWindow.WriteLine("Jak smakowało śniadanie?")
EndIf
```

Else

W programie umieszczonym na początku rozdziału daje się zauważyć pewna niepotrzebna nadmiarowość (zwana przez programistów redundancją). Program najpierw sprawdza czy Clock.Hour jest mniejsze od 18 a potem, czy jest większe lub równe. Oczywiste jest, że skoro nie jest mniejsze, to musi być większe lub równe, więc sprawdzanie dwa razy tego samego warunku jest tak naprawdę zbędne. W przypadku języka Small Basic, można prosto zrezygnować z drugiego sprawdzenia, dodając do znanej już składni **If..Then..EndIf** słowo kluczowe **Else**.

Program napisany z użyciem tego słowa może wyglądać tak::

```
If (Clock.Hour < 18) Then
  TextWindow.WriteLine("Dzień dobry świecie")
Else
```

```
  TextWindow.WriteLine("Dobry wieczór świecie")
EndIf
```

Ten program zachowa się tak samo jak wersja z dwoma sprawdzeniami bieżącej godziny, co prowadzi do bardzo istotnego wniosku:

> *W programowaniu zazwyczaj istnieje wiele sposobów rozwiązania tego samego zadania. Wybór należy do programisty. Czasem jedna droga jest bardziej sensowna niż inne a poznanie, która jest właściwa zwykle przychodzi wraz z doświadczeniem.*

Wcięcia

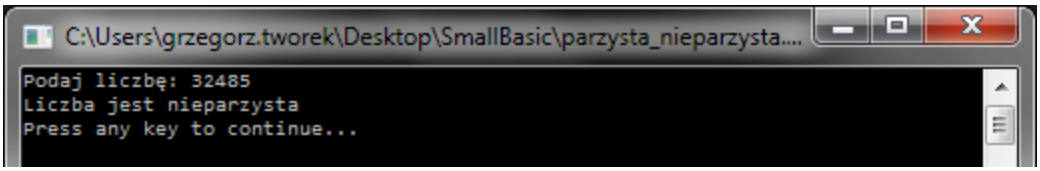
We wszystkich przykładach wykorzystujących *If, Else* oraz *EndIf* można zauważyć, że linie pomiędzy tymi słowami kluczowymi nie zaczynają się w pierwszej kolumnie, ale są poprzedzone spacjami (w potocznej mowie programistów – wcięte). Nie jest to niezbędne i komputer dokładnie tak samo zrozumie program bez wcięć. Przyjęło się jednak stosować wcięcia, ponieważ dzięki nim, program jest znacznie prostszy do zrozumienia przez człowieka. Uznawane jest to za tak zwaną dobrą praktykę i niemal wszyscy programiści na świecie właśnie tak piszą swoje programy. Wcięcia można zrobić automatycznie, klikając na już napisanym programie prawym przyciskiem myszy i wybierając z menu kontekstowego pozycję "Formatuj kod źródłowy".

Parzyste czy nieparzyste

Teraz, gdy użycie **If..Then..Else..EndIf** jest jasne, można napisać wiele programów, które wykorzystają te polecenia. Przykładem może być prosty program wykrywający czy liczba jest parzysta czy nieparzysta.

```
TextWindow.Write("Podaj liczbę: ")
num = TextWindow.ReadNumber()
reszta = Math.Remainder(num, 2)
If (reszta = 0) Then
  TextWindow.WriteLine("Liczba jest parzysta")
Else
  TextWindow.WriteLine("Liczba jest nieparzysta")
EndIf
```

Po uruchomieniu, na ekranie można zobaczyć:



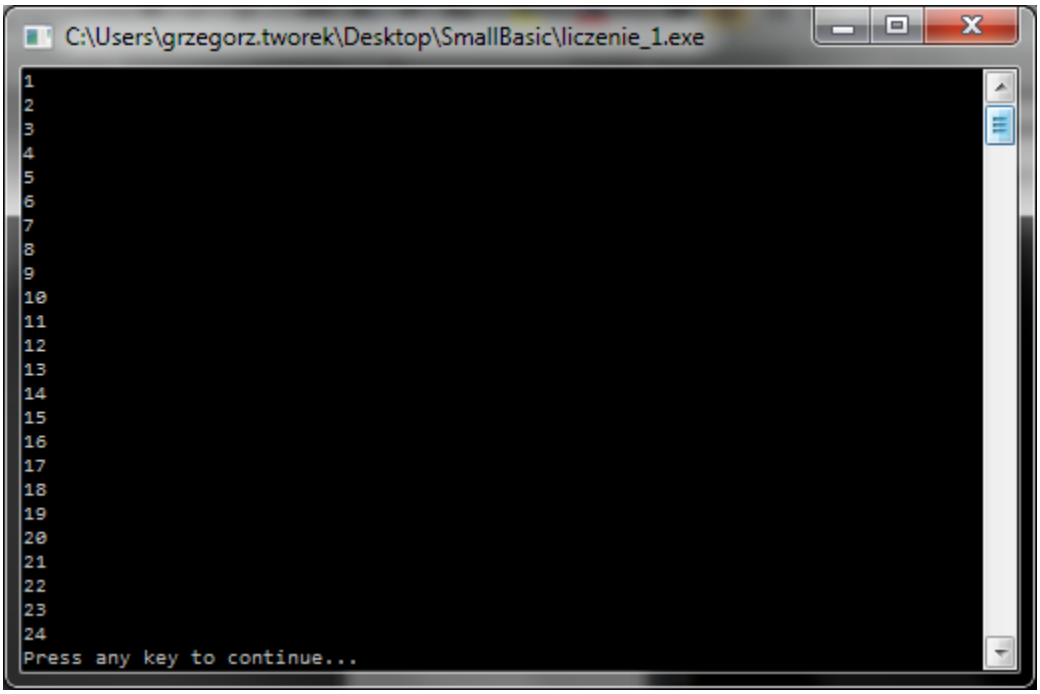
Rysunek 16 – Parzysta czy nieparzysta

W tym programie użyto operacji **Math.Remainder**. Jak nietrudno się domyślić, oblicza ona resztę z dzielenia pierwszego argumentu przez drugi

Skoki

W pierwszych programach, komputer wykonywał polecenia jedno po drugim, w takiej kolejności jaka wynikała z kolejności linii w programie. W językach programowania istnieje zwykle specjalne polecenie, pozwalające na wykonanie tak zwanego skoku do innej linii. Zobaczyć to można na przykładzie:

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



Rysunek 17 – Użycie polecenia Goto

W powyższym programie, do zmiennej i przypisano wartość 1. Następna linia zawiera słowo kończące się dwukropkiem (:)

```
start:
```

Polecenie takie nazywane jest *etykietą* (ang. Label).Jego zastosowanie można porównać do adresu lub zakładki. Mówi ono "to miejsce w programie nazywa się start". W programie może być dowolnie dużo etykiet pod warunkiem, że ich nazwy nie będą się powtarzać.

Następnym ciekawym poleceniem jest:

```
i = i + 1
```

Oznacza ono "przypisz do zmiennej i wartość tej zmiennej powiększoną o 1". Jako, że na początku programu przypisano do i wartość 1, to po wykonaniu tego polecenia, wartość ta wyniesie 2.

I w końcu:

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

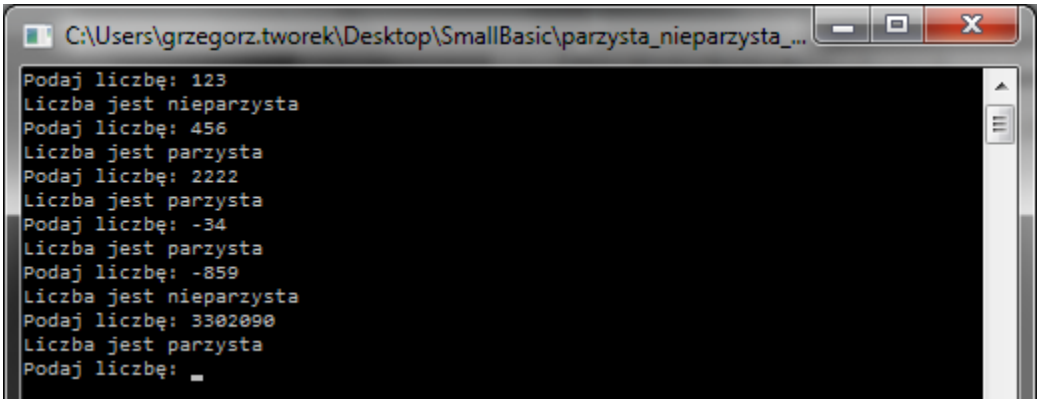
Ta część sprawdza czy i jest mniejsze od 25. Jeżeli tak, to program wykonuje skok do miejsca oznaczonego etykietą start. Czyli, po raz kolejny wyświetla i na ekranie, zwiększa je, sprawdza czy jest mniejsze od 25 itd.

Wykonanie bez końca

Używając polecenia **Goto**, można w prosty sposób sprawić, aby komputer bez końca wykonywał zestaw poleceń. Na przykład, opisany wcześniej prosty program sprawdzający czy liczba jest parzysta czy nie, można zmodyfikować tak, aby nigdy nie skończył swojego działania. Użytkownik oczywiście może zakończyć program klikając na ikonie X w prawym górnym rogu okna.

```
begin:
TextWindow.Write("Podaj liczbę: ")
num = TextWindow.ReadNumber()
reszta = Math.Remainder(num, 2)
If (reszta = 0) Then
  TextWindow.WriteLine("Liczba jest parzysta")
Else
```

```
  TextWindow.WriteLine("Liczba jest nieparzysta")
EndIf
Goto begin
```



Rysunek 18 – Parzysta czy nieparzysta bez końca

Rozdział 5

Pętle

Pętla For

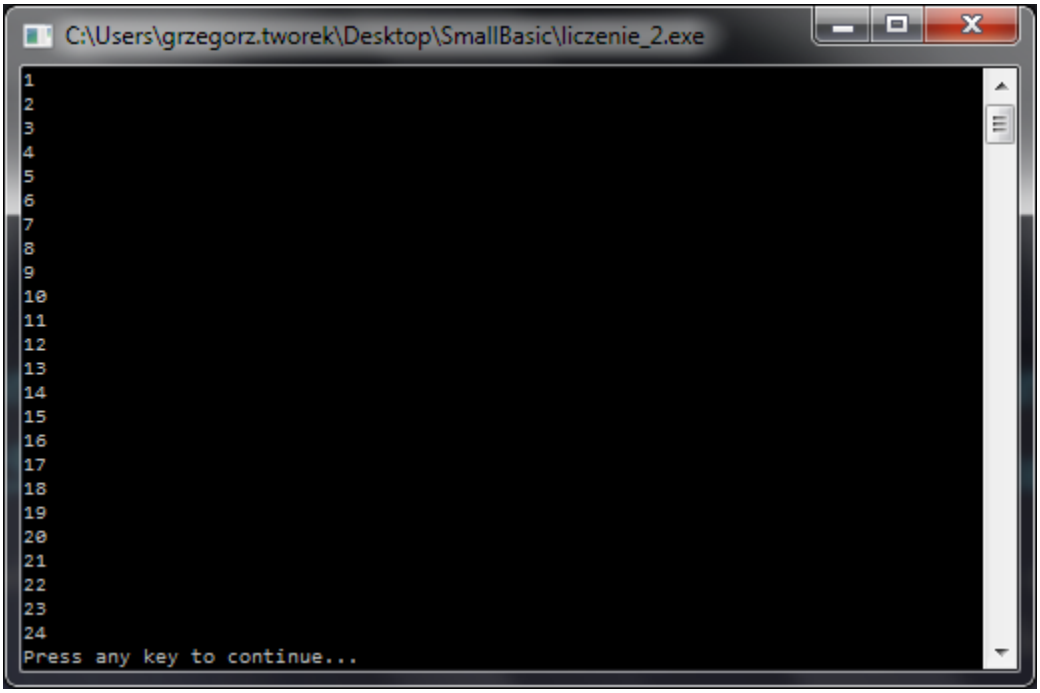
W poprzednim rozdziale przeanalizowany został prosty program liczący do 24.

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

Ponieważ w programowaniu bardzo często wykorzystuje się operacje polegające na liczeniu od 1 do zadanej liczby, wprowadzono specjalne polecenie upraszczające to zadanie. Poniższy program robi dokładnie to samo, co poprzedni:

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

Wynikiem jego działania oczywiście jest:



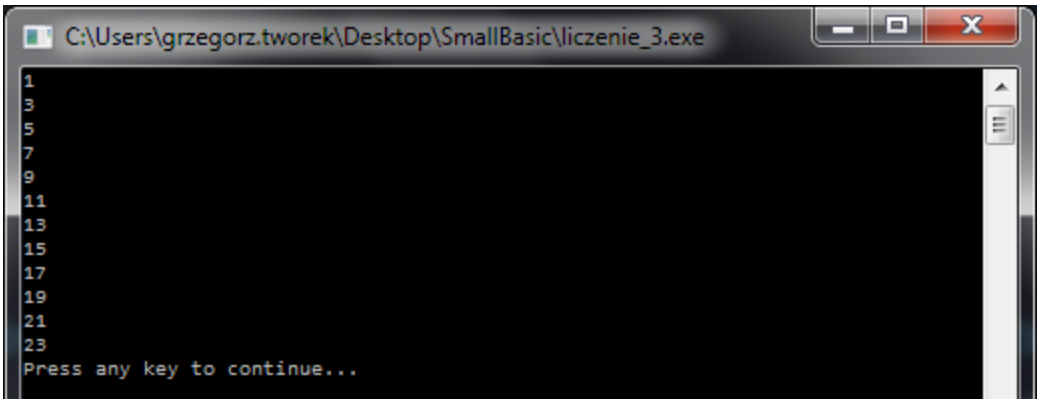
Rysunek 19 – Użycie pętli For

Warto zwrócić uwagę, że program robiąc dokładnie to samo, zajmuje 3 linie a nie 7. Jest to doskonałym przykładem, wspominanej już wcześniej zasady, że niemal każdy program można napisać na wiele sposobów.

Struktura **For..EndFor** w nazewnictwie stosowanym przez programistów nazywana jest *pętlą*. Umożliwia ona użycie zmiennej, nadanie jej wartości początkowej i pozwala komputerowi na samodzielne zwiększanie jej wartości. Za każdym razem, gdy wartość jest zwiększona, komputer ponownie wykona wszystkie polecenia zawarte pomiędzy **For** i **EndFor**.

Jak widać, wartość zmiennej zwiększana jest za każdym razem o 1. Właśnie takie zwiększenie jest najczęściej potrzebne, ale jeżeli ktoś chciałby zwiększać tę wartość na przykład o 2, również mógłby to zrobić.

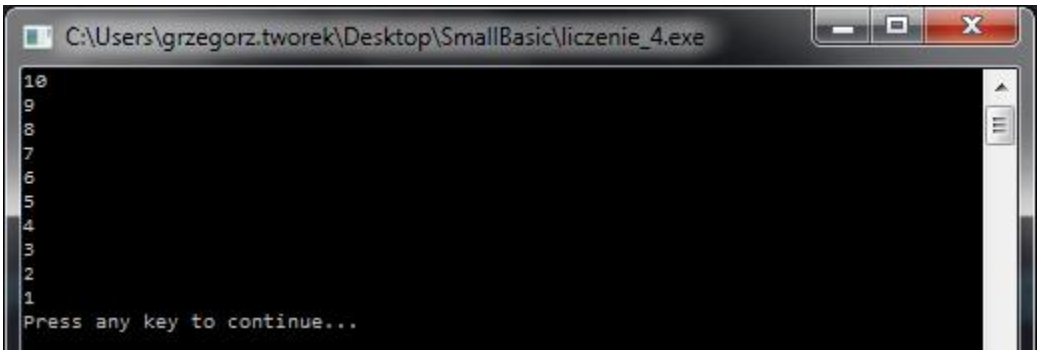
```
For i = 1 To 24 Step 2
  TextWindow.WriteLine(i)
EndFor
```



Rysunek 20 – Liczby nieparzyste w pętli For

Część **Step 2** polecenia **For** mówi właśnie o ile ma zwiększyć się wartość przy każdym przebiegu. Pominięcie słowa **Step** oznacza, że użyte ma być standardowe zwiększanie o 1. Po słowie kluczowym **Step** użyć można dowolnej liczby, włącznie z liczbami ujemnymi. Oznaczać to będzie, że komputer w każdym przebiegu będzie zmniejszał wartość zmiennej zamiast ją zwiększać. Efekt będzie taki jak na poniższym przykładzie:

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

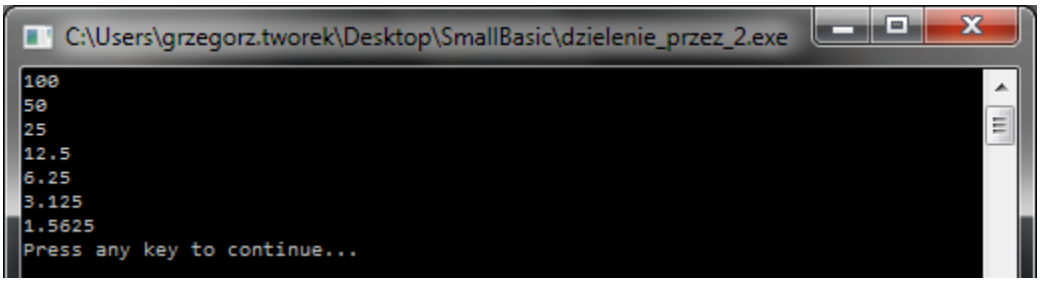


Rysunek 21 – Odliczanie od 10 do 1

Pętla While

Inną metodą zdefiniowania pętli w programie jest polecenie **While**. Jest ono szczególnie użyteczne, gdy dokładna liczba wykonań pętli nie jest z góry znana. Podczas gdy pętla For wykonuje się z góry założoną ilość razy, pętla While (ang. dopóki) wykonuje się tak długo, jak długo zadany warunek jest prawdziwy. W poniższym przykładzie, liczba dzielona jest przez 2 tak długo, jak długo wynik jest większy niż 1.

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



Rysunek 22 – Pętla dzieląca przez 2

W powyższym programie, do zmiennej number przypisywana jest wartość 100 i program wykonuje pętlę loop tak długo, jak długo wartość zmiennej number jest większa niż 1. Wewnątrz pętli znajduje się polecenie wyświetlania aktualnej wartości na ekranie oraz dzielenie przez 2. Zgodnie z oczekiwaniem, program wyświetla na ekranie kolejne liczby, z których każda jest równa połowie poprzedniej.

Tak naprawdę, w czasie wykonywania program komputer posługuje się If…Then oraz Goto, ale napisanie program z While jest prostsze dla programisty. Program taki łatwiej jest analizować i przerabiać w przyszłości.

Napisanie tego samego z wykorzystaniem pętli For nie byłoby tak proste, ponieważ programista z góry musiałby wiedzieć po ilu dzieleniach wynik zmniejszy się poniżej 1. W przypadku pętli While jest to sprawdzane po każdym przebiegu pętli i decyzja jest podejmowana na bieżąco.

W zasadzie, każdą pętlę typu While można "przetłumaczyć" na pętlę opartą na poleceniach If...Then. Na przykład powyższy program można zapisać następująco:

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

Rozdział 6

Podstawy grafiki

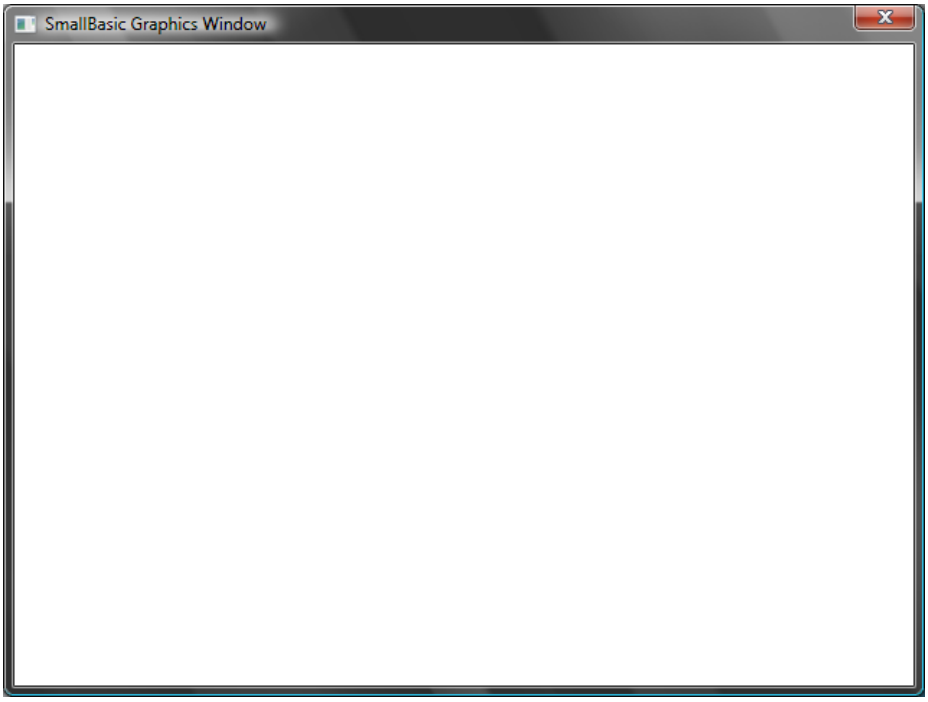
We wszystkich przedstawianych dotąd przykładach używany był obiekt TextWindow, czyli okno tekstowe. Jest to dobre dla wytłumaczenia podstaw języka, ale większość współczesnych programów pracuje raczej z grafiką niż z tekstem. Oczywiście język Small Basic pozwala również na pracę z grafiką. Niniejszy rozdział przybliża podstawy programowania grafiki, dzięki czemu możliwe będzie tworzenie bardziej rozbudowanych programów.

Obiekt GraphicsWindow

W języku Small Basic, obiekt TextWindow pozwala na pracę z tekstem czy liczbami. Podobnym obiektem jest obiekt **GraphicsWindow** (okno graficzne), który udostępnia metody pozwalające na rysowanie. Aby skorzystać z tego okna, należy je wyświetlić na ekranie.

```
GraphicsWindow.Show()
```

Po uruchomieniu takiego program można zauważyć, że zamiast dotychczasowego czarnego okna tekstowego, pojawi się okno zbliżone do tego na rysunku poniżej. Okno na razie jest puste, ale w dalszej części rozdziału pokazane zostanie jak coś w nim umieścić. Okno to można zamknąć klikając na ikonie X w prawym górnym rogu.



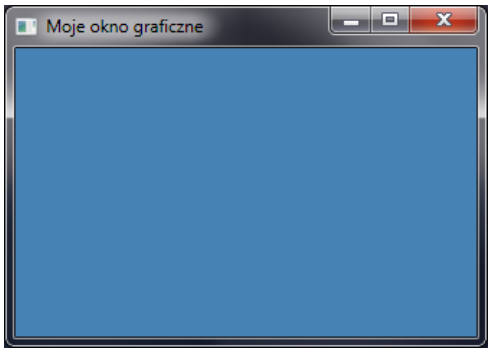
Rysunek 23 – Puste okno graficzne

Ustawianie okna graficznego

Wygląd okna graficznego może być niemal dowolnie zmieniany przez programistę. Można ustawić jego tytuł, tło czy rozmiar. Dla ćwiczenia warto nieco przerobić wyświetlone przed chwilą okno:

```
GraphicsWindow.BackgroundColor = "SteelBlue"
GraphicsWindow.Title = "Moje okno graficzne"
GraphicsWindow.Width = 320
GraphicsWindow.Height = 200
GraphicsWindow.Show()
```

Widać, że okno takie wygląda zupełnie inaczej. W "Dodatku B" na końcu podręcznika, znaleźć można listę kolorów, które można zastosować. Warto samodzielnie pobawić się właściwościami okna tak, aby w przyszłości umieć je szybko dostosować do konkretnych potrzeb.

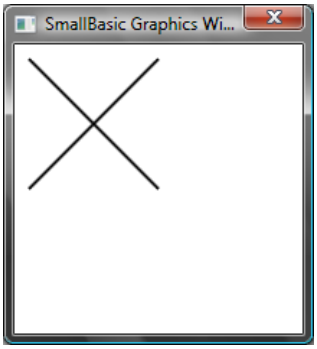


Rysunek 24 – Zmodyfikowane okno graficzne

Rysowanie linii

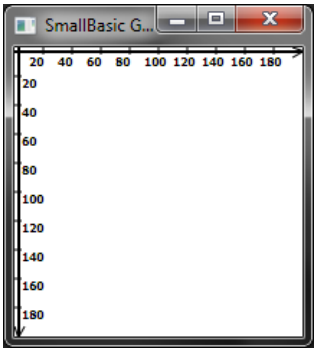
Umiejąc już wyświetlić okno graficzne, można stosunkowo prosto rysować w nim kształty, tekst czy nawet całe obrazy. Warto zacząć od prostego przykładu. Poniższy program rysuje w oknie graficznym dwie linie.

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



Rysunek 25 – Krzyżyk

Pierwsze dwie linie programu ustawiają właściwości okna graficznego a drugie dwie – rysują w nim krzyżujące się linie. Liczby będące parametrami polecenia *DrawLine* są współrzędnymi końców rysowanego odcinka. Interesujące jest, że w grafice stosowanej w komputerach, punkt (0,0) oznacza zwykle lewy górny róg okna. Oznacza to, że ekran graficzny może być traktowany jak druga ćwiartka układu współrzędnych.

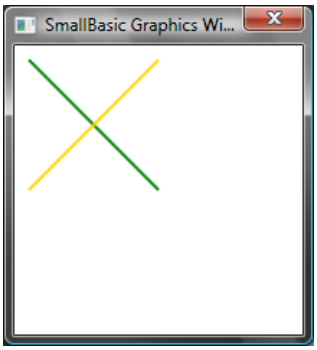


Rysunek 26 – Układ współrzędnych

Wracając do programu rysującego przecinające się linie, warto wiedzieć, że Small Basic pozwala na modyfikowanie sposobu, w jaki linie są rysowane. Do rysowania linii używany jest tak zwany pisak (Pen) i odpowiednimi poleceniami można zmienić na przykład jego kolor na zielony (green), złoty (gold) czy dowolny inny.

Zamiast używać nazw kolorów, w języku Small Basic można użyć notacji znanej ze stron internetowych (#RRGGBB). Na przykład, #FF0000 oznacza czerwony, #FFFF00 – żółty itd. Więcej szczegółów znajduje się w "Dodatku B" na końcu niniejszego podręcznika.

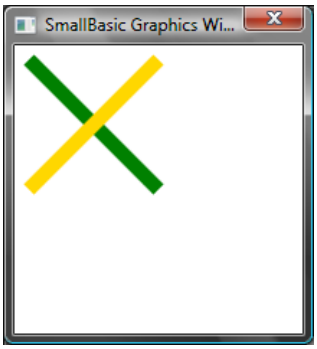
```
GraphicsWindow.Width = 200
GraphicsWindow.Height = 200
GraphicsWindow.PenColor = "Green"
GraphicsWindow.DrawLine(10, 10, 100, 100)
GraphicsWindow.PenColor = "Gold"
GraphicsWindow.DrawLine(10, 100, 100, 10)
```



Rysunek 27 – Kolorowe linie

Można również zmienić grubość pisaka a przez to grubość rysowanej linii. Domyślnie jest ona równa 1, ale dla testu można ją zmienić na przykład na 10.

```
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)
```

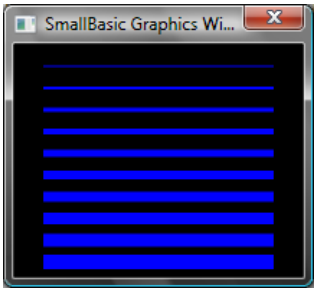


Rysunek 28 – Grube kolorowe linie

Polecenia *PenWidth* oraz PenColor zmieniają właściwości pisaka i wszystko, co będzie nim rysowane (linie, figury itp.) będzie miało określone właściwości . Dlatego, przed rysowaniem warto upewnić się, że pisak ma takie właściwości, jakie są w danym momencie potrzebne i ewentualnie ustawić je zgodnie z oczekiwaniami.

Łącząc wiedzę o właściwościach linii z umiejętnością tworzenia pętli, można łatwo napisać program rysujący kolejno linie o coraz większej grubości.

```
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
```



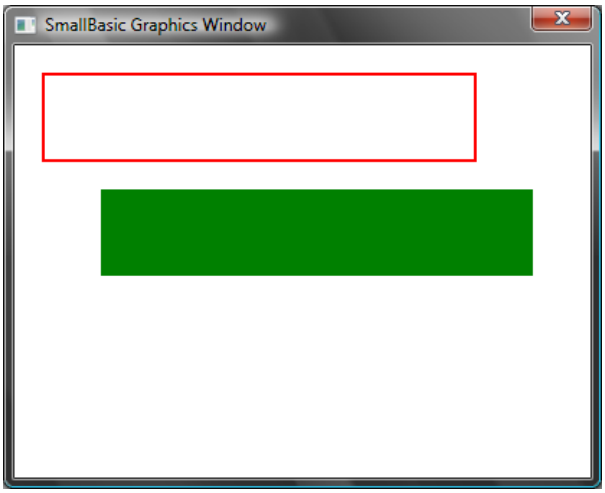
Rysunek 29 – Różne grubości pisaka

Warto przyjrzeć się pętli w programie, aby zrozumieć jak to się dzieje, że linie są coraz grubsze i rysowane jedna pod drugą.

Rysowanie i wypełnianie kształtów

W przypadku rysowania kształtów, zazwyczaj w grę wchodzą dwie operacje: Draw (rysuj) i Fill (wypełnij). Draw rysuje figurę używając pisaka (Pen) a Fill – używając pędzla (Brush). Pędzel i pisak mogą mieć w tym samym zupełnie różne właściwości. Przykładowo, poniższy program rysuje dwa prostokąty. Jeden jest pusty i wykorzystuje czerwony pisak a drugi jest wypełniony i do jego namalowania używany jest zielony pędzel..

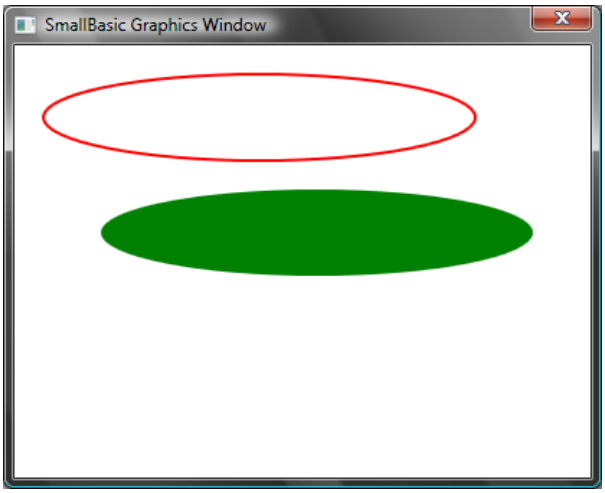
```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

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

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



Rysunek 30 – Rysowanie i wypełnianie

Warto zwrócić uwagę, że kształty te nie zostały narysowane z poszczególnych odcinków. Do ich utworzenia wykorzystano polecenia DrawRectangle i FillRectangle rysujące gotowe prostokąty. Narysowanie prostokąta wymaga czterech parametrów. Są to odpowiednio: współrzędna x lewego górnego rogu, współrzędna y lewego górnego rogu, długość podstawy (szerokość prostokąta) oraz wysokość prostokąta. Identyczny zestaw parametrów podaje się również przy rysowaniu elips:

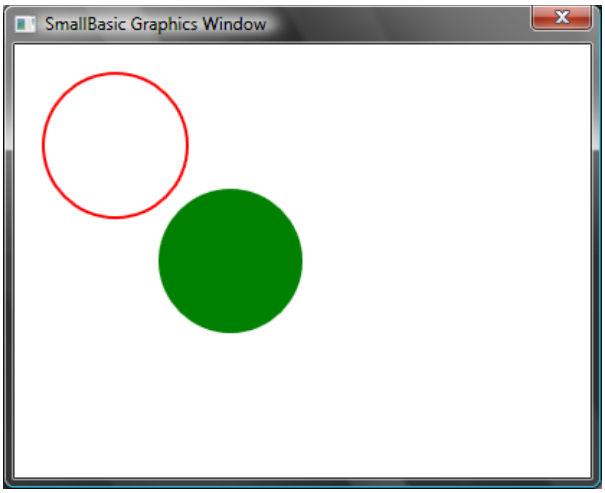
```
GraphicsWindow.Width = 400
GraphicsWindow.Height = 300

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

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



Rysunek 31 – Rysowanie i wypełnianie elips

Gdyby programista zechciał narysować okrąg lub koło, musiałby użyć elipsy o wysokości (długości osi pionowej) takiej samej jak szerokość (długość osi poziomej). Koło i okrąg są szczególnymi przypadkami elipsy tak, jak kwadrat jest szczególnym przypadkiem prostokąta.

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

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

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



Rysunek 32 – Okrąg i koło

Rozdział 7

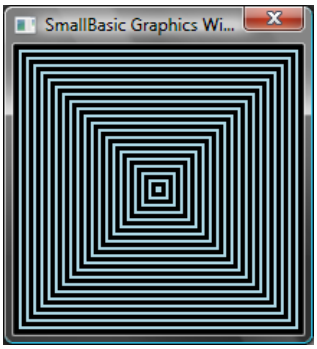
Zabawa z grafiką

W niniejszym rozdziale pokazane zostaną proste programy, które dzięki poznanym dotychczas elementom programowania w Small Basic, potrafią robić naprawdę ciekawe rzeczy.

Prostokąty

Używając pętli For, można namalować serię prostokątów o stopniowo zwiększającym się rozmiarze.

```
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
```

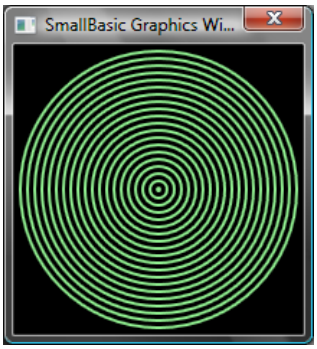


Rysunek 33 – Prostokąty

Okręgi

Analogicznie do poprzedniego programu, można utworzyć obraz składający się z okręgów..

```
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
```

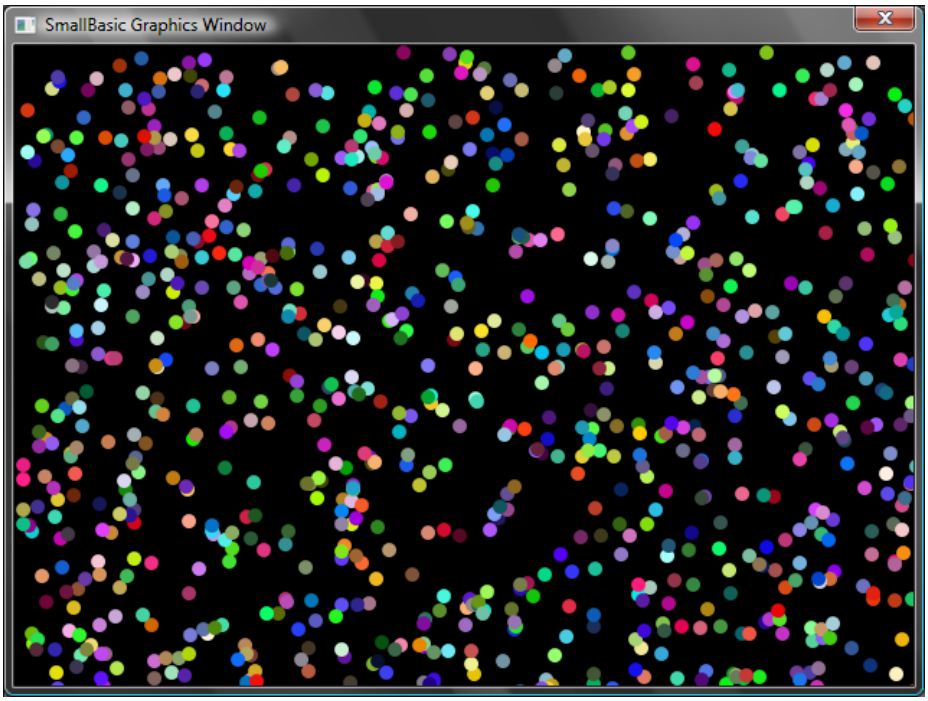


Rysunek 34 – Okręgi

Chaos

Następny program wykorzystuje metodę GraphicsWindow.*GetRandomColor* (wybierz losowy kolor) do ustawienia przypadkowego koloru dla pędzla, a następnie metodę Math.*GetRandomNumber* (wybierz losową liczbę), żeby określić przypadkowe współrzędne x i y. Następnie rysuje niewielkie koło w zadanym miejscu, wykorzystując ustawiony wcześniej kolor pędzla. Ponieważ kolory i położenie kółek są losowane, otrzymany obraz wygląda za każdym razem inaczej.

```
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
```



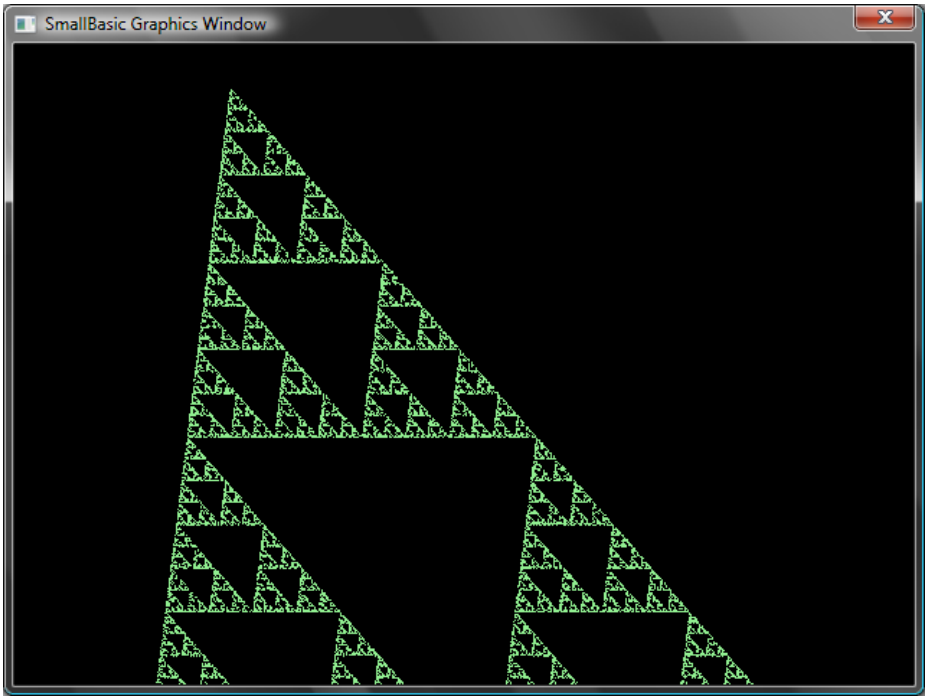
Rysunek 35 – Chaos

Fraktale

Kolejny program rysuje prosty, trójkątny fraktal zbliżony do tak zwanego Dywanu Sierpińskiego. Cechą charakterystyczną fraktali jest to, że dowolny jego fragment, po powiększeniu ma taki sam wzór jak całość. W tym przypadku, program rysuje setki trójkątów podobnych do trójkąta stanowiącego całość obrazu. Logika działania programu nie jest trywialna i w ramach samodzielnego ćwiczenia, warto w wolnej chwili poświęcić trochę uwagi, aby zrozumieć jego działanie.

```
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
```



Rysunek 36 – Trójkątny fraktal

Jeżeli ktoś chciałby zaobserwować w "zwolnionym tempie" proces budowania fraktala z punktów, musiałby wewnątrz pętli umieścić instrukcję spowalniającą Program.Delay(). Parametrem dla tej instrukcji jest liczba milisekund, (tysięcznych części sekundy), określająca na jak długo wykonanie programu powinno być wstrzymane. Zmodyfikowany program wyglądałby następująco:

```
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
```

Zwiększanie parametru podanego w instrukcji Delay() jeszcze bardziej spowolni program. Najwygodniejszą dla użytkownika prędkość można dobrać metodą prób i błędów.

W ramach modyfikacji, można również zastąpić linię:

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

liniami

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

Ta zmiana sprawi, że zamiast zielonych punktów, zaczną pojawiać się punkty w losowych kolorach.

Rozdział 8

Grafika żółwia

Logo

W latach siedemdziesiątych XX wieku, pojawił się prosty język programowania Logo. Nie był on szczególnie popularny aż do chwili, kiedy dodano do niego tak zwaną grafikę żółwia. Nazwa wzięła się stąd, że na ekranie widoczny był żółw, któremu można było wydać polecenia typu *Move Forward* (idź do przodu)*, Turn Right* (skręć w prawo)*, Turn Left* (skręć w lewo) itp. Używając żółwia, nawet niedoświadczeni programiści mogli rysować interesujące kształty na ekranie. W efekcie, język Logo stał się bardzo popularny i lubiany.

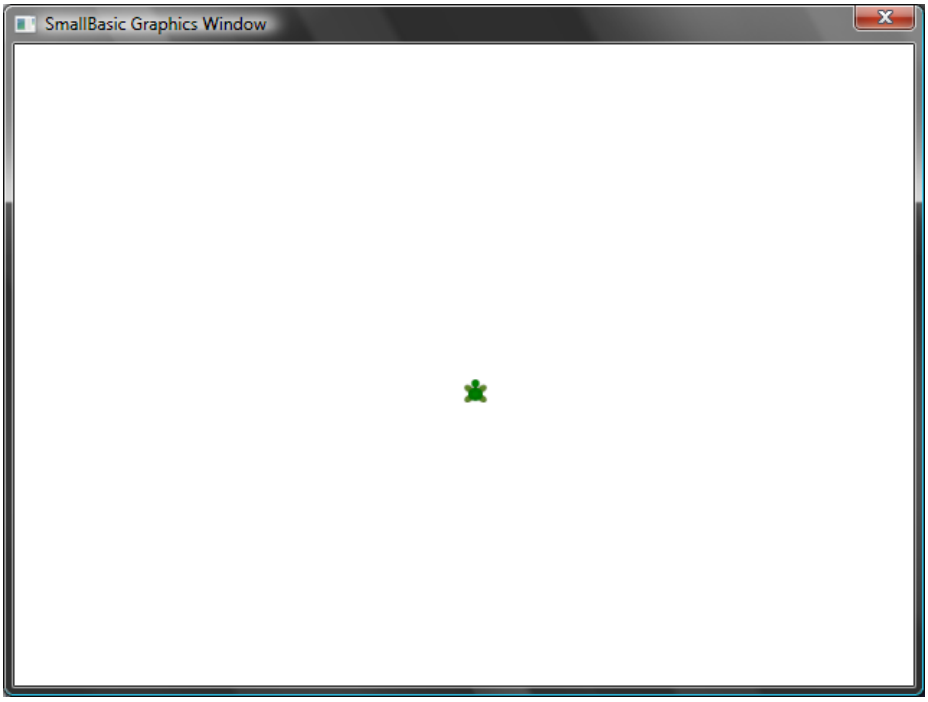
Język Small Basic wyposażony jest w obiekt **Turtle** (żółw), któremu, poprzez programy można wydawać polecenia. Niniejszy rozdział opisuje sposób, w jaki można programować zachowanie żółwia tak, aby rysował on na ekranie.

Żółw

Aby rozpocząć pracę z żółwiem, należy wyświetlić go na ekranie. Służy do tego polecenie przedstawione poniżej:

```
Turtle.Show()
```

Po uruchomieniu takiego program, pokaże się standardowe okno graficzne z żółwiem w środku. Żółw jest gotowy do pracy i można mu wydawać polecenia.



Rysunek 37 – Żółw jest widoczny

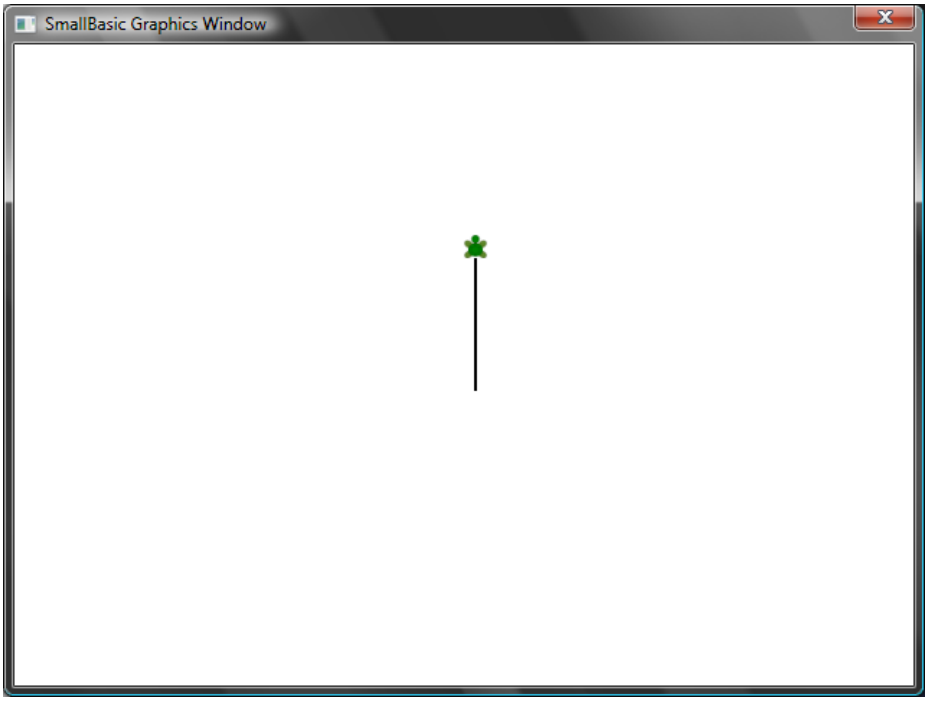
Ruch i rysowanie

Jedną z instrukcji rozumianych przez żółwia jest instrukcja **Move** (przesuń się). Parametrem dla tej operacji jest liczba, określająca jak daleko żółw ma się przesunąć. Przykładowo, jeżeli żółw ma się przesunąć o 100 punktów, polecenie będzie wyglądać następująco:

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

Po uruchomieniu tego program, można zaobserwować żółwia powoli przemieszczającego się do przodu (w górę ekranu) o 100 punktów i zostawiającego za sobą ślad. Gdy żółw skończy się przesuwać, wynikowy obraz wygląda mniej więcej jak na ilustracji:

Tak naprawdę, wywołanie metody Show() wcale nie jest niezbędne. Żółw i tak pokaże się na ekranie, gdy wyda mu się dowolne polecenie.



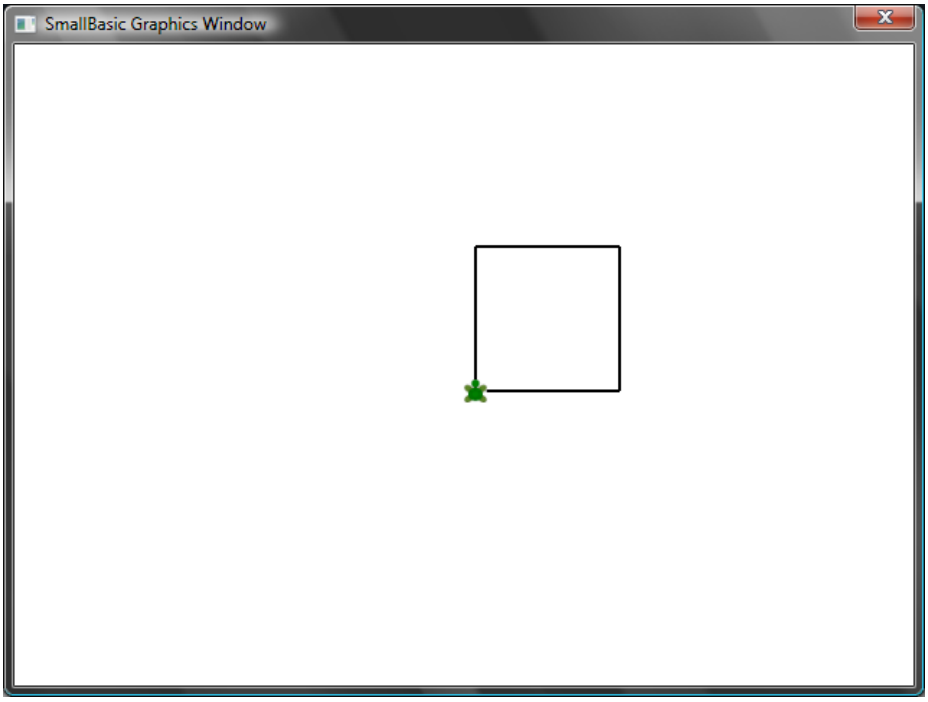
Rysunek 38 – Przesunięcie o 100 punktów

Rysowanie kwadratu

Kwadrat ma cztery boki. Dwa pionowa i dwa poziome. Aby żółw narysował kwadrat, należy kazać mu iść do przodu, obrócić się o 90 stopni w prawo, iść do przodu i tak dopóty, dopóki kwadrat nie będzie gotowy. W programie wyglądać to będzie tak.

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

Po uruchomieniu tego program, można zaobserwować jak linia po linii żółw rysuje kwadrat taki, jak na ilustracji poniżej.



Rysunek 39 – Żółw rysujący kwadrat

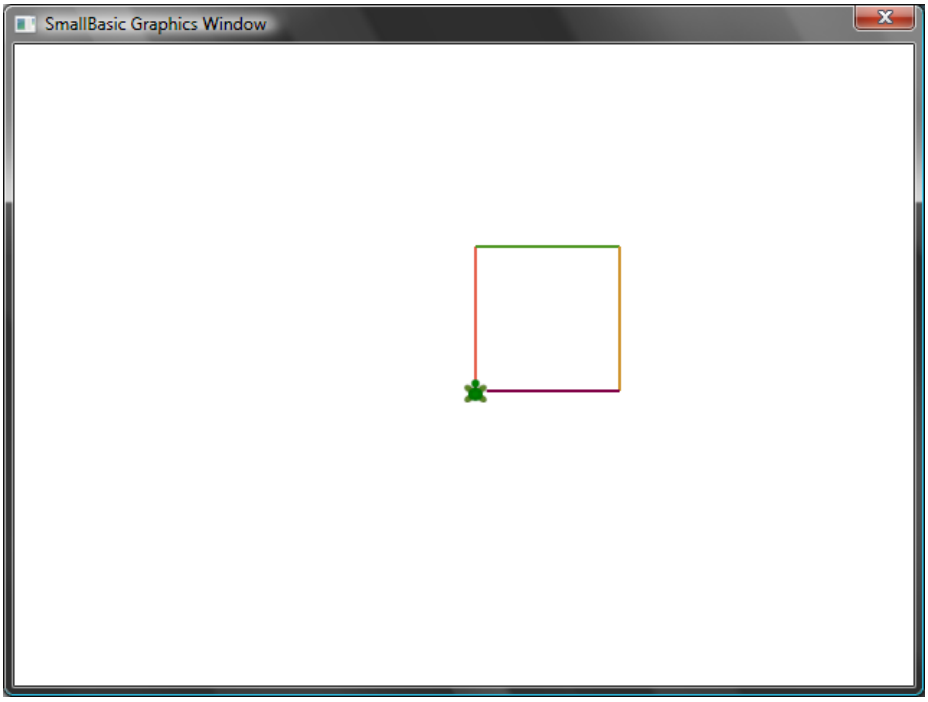
Ponieważ żółw wykonuje cztery razy te same instrukcje, warto skorzystać z opisanej we wcześniejszych rozdziałach pętli For…EndFor. Program po zmodyfikowaniu działa tak samo a jest przy tym znacznie prostszy i lepiej wygląda.

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

Zmienianie kolorów

Żółw, rysując używa tego samego mechanizmu pisaka, co opisywane we wcześniejszych rozdziałach rysowanie odcinka między dwoma punktami. W efekcie, zmiana pisaka zmieni to, co żółw zostawia na ekranie. Przykładowo, możliwe jest narysowanie kwadratu, którego każdy bok ma inny kolor.

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



Rysunek 40 – Zmienianie kolorów

Rysowanie złożonych obrazów

Niezależnie od poleceń **TurnRight** (skręć w prawo) i **TurnLeft** (skręć w lewo), żółw reaguje na polecenie **Turn** (skręć). Polecenie to wymaga podania parametru, który określa kąt, o jaki ma obrócić się żółw. Wykorzystując polecenie Turn, można rysować dowolny wielokąt a nie tylko figury, posiadające wyłącznie kąty proste. Przykładowy program pokazuje jak narysować sześciokąt:

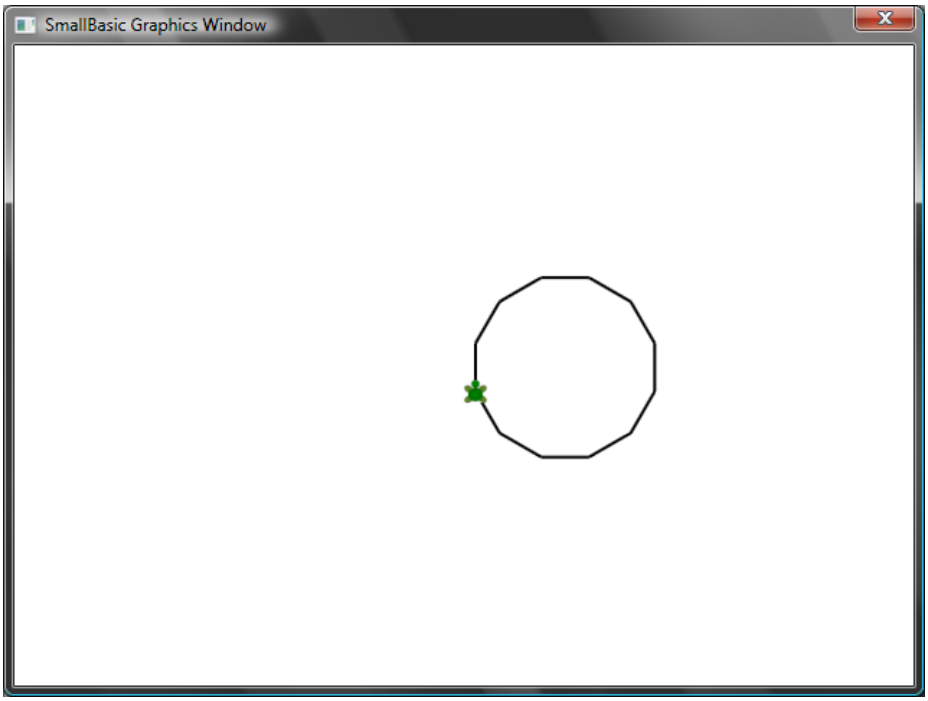
```
For i = 1 To 6
  Turtle.Move(100)
  Turtle.Turn(60)
EndFor
```

Warto uruchomić ten program, aby zobaczyć jak to się dzieje, że powstaje sześciokąt. Trzeba pamiętać, że kąt, o jaki należy obrócić żółwia nie jest tym samym, co kąt pomiędzy bokami wielokąta. Kąty "widziane" są z perspektywy żółwia i jeżeli kąt między bokami wynosi α, to żółwia należy obrócić o kąt 180-α. Dla wielokątów foremnych, kąt o jaki należy obrócić żółwia wynosi zawsze 360/n, gdzie n jest ilością boków. Uzbrojony w tę wiedzę programista może napisać prosty program rysujący dowolny wielokąt foremny

```
sides = 12

length = 400 / sides
```

```
angle = 360 / sides

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

W powyższym programie wystarczy zmienić wartość zmiennej sides (boki), aby narysować dowolny wielokąt foremny. Wszystkie pozostałe wartości potrzebne żółwiowi są automatycznie wyliczane przez program. Mając w zmiennej sides wartość 4, program narysuje kwadrat. Gdy wartość jest odpowiednio duża (na przykład 50), otrzymany na ekranie wielokąt praktycznie przypomina okrąg.



Rysunek 41 – Rysowanie dwunastokąta foremnego

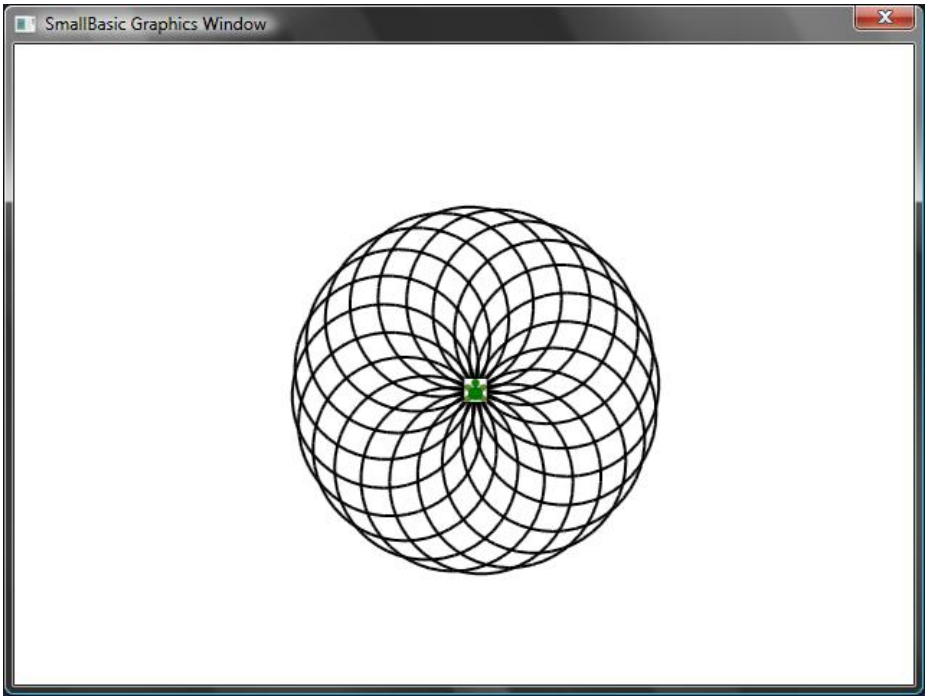
Używając tego algorytmu, można narysować okrąg, następnie obrócić żółwia o niewielki kąt, narysować ponownie okrąg itd. Okręgi nie będą się pokrywać, a efekt na ekranie będzie bardzo interesujący.

```
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
```

Powyższy program zawiera dwie pętle **For..EndFor** jedna w drugiej. Pętla wewnętrzna (*i = 1 to sides*) jest zbliżona do pętli w programie rysującym wielokąty i odpowiada za rysowanie okręgów. Pętla zewnętrzna (*j = 1 to 20*) odpowiada za obracanie żółwia o 18 stopni za każdym razem, gdy skończy on rysować okrąg. Zewnętrzna pętla wykonywana jest 20 razy, więc na wynikowym rysunku pojawia się 20 okręgów tworzących rozetę.

W programie przyspieszono żółwia, przypisując wartość 9 do właściwości Speed. Można ustawić dowolną prędkość żółwia, korzystając z wartości od 1 do 10.



Rysunek 42 – Rozeta

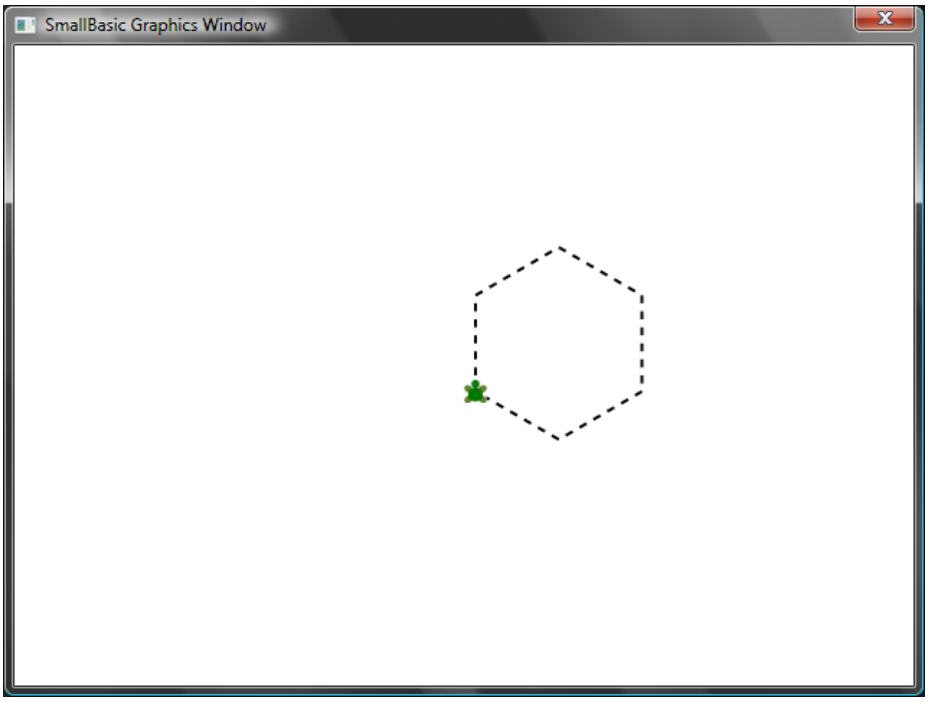
Przesuwanie żółwia

Do tej pory, żółw przemieszczając się zostawiał ślad. Nie zawsze jest to pożądane i dlatego, można wydać żółwiowi polecenie **PenUp** (podnieś pisak). Kolejne przemieszczenia żółwia nie spowodują narysowania linii na ekranie. Aby powrócić do stanu, w którym żółw rysuje, wystarczy wywołać metodę **PenDown**

(opuść pisak). Można to wykorzystać na wiele różnych sposobów, takich jak na przykład rysowanie linii przerywanych. Poniższy program rysuje wielokąt foremny stosując linię przerywaną.

```
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
```

Program ten (podobnie jak poprzedni) wykorzystuje dwie pętle. Wewnętrzna pętla rysuje linię przerywaną a zewnętrzna odpowiada za rysowanie boków wielokąta. W przykładzie jest to sześciokąt, ale przypisując inną wartość do zmiennej sides, można uzyskać dowolny inny wielokąt foremny.



Rysunek 43 – Użycie PenUp i PenDown

Rozdział 9

Procedury

Bardzo często, programując natrafia się na sytuacje, w których kilka poleceń wykonywanych jest zawsze razem, w wielu różnych miejscach programu. Można łatwo się domyśleć, że przepisywanie ich wielokrotnie ma niewielki sens. W takich przypadkach przydają się procedury, po angielsku nazywane w Small Basic *Subroutines*.

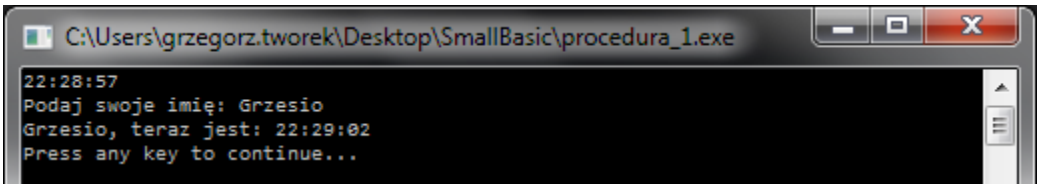
Procedura jest fragmentem programu, który ma swoje określone zadanie i jest wielokrotnie wywoływany. Aby utworzyć procedurę, należy użyć słowa kluczowego **Sub** a po nim umieścić nazwę, pod którą procedura będzie znana w programie. Koniec procedury oznaczony jest przez słowo kluczowe **EndSub**. Przykładowo, procedura o nazwie PrintTime (ang. wyświetl czas) wyświetla bieżący czas w oknie tekstowym:

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

Poniższy program używa tej procedury, wywołując ją tak, jak dotąd wywoływane były jedynie polecenia Small Basic. Procedura jest poleceniem, które użytkownik sam może sobie zdefiniować i zaprogramować.

```
PrintTime()
TextWindow.Write("Podaj swoje imię: ")
name = TextWindow.Read()
TextWindow.Write(name + ", teraz jest: ")
PrintTime()
```

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



Rysunek 44 – Wywołanie procedury PrintTime

Nawiasy po PrintTime są niezbędne, aby Small Basic mógł zrozumieć, że chodzi o wywołanie procedury zdefiniowanej w innym miejscu programu.

Zalety korzstania z procedur

Jak wspomniano powyżej, procedury powodują, że program jest krótszy i szybciej się go pisze. Jeżeli procedura raz zostanie zdefiniowana, można jej prosto używać w wielu miejscach i zrobi ona to, co określił programista.

Dodatkowo, procedury pozwalają często na uproszczenie programu, dzieląc go w pewien sposób na części odpowiedzialne za konkretne zadania. Na przykład, przy rozwiązywaniu złożonego problemu matematycznego, można podzielić go na elementy i w każdej procedurze obliczyć fragment rozwiązania. W ten sposób z jednej strony widać jak ogólnie wygląda rozwiązanie problemu, a z drugiej – w każdej chwili można wgłębić się w szczegóły.

Należy pamiętać, że procedury można wywoływać tylko z tego samego programu, w którym zostały zdefiniowane. Wywołania procedur pomiędzy programami nie są w Small Basic możliwe.

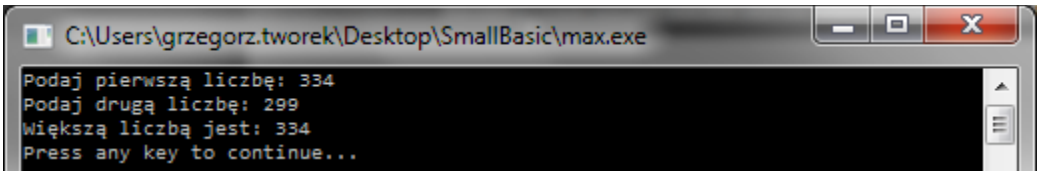
Ponadto, jeżeli procedury zostaną rozsądnie nazwane, ich zastosowanie zwiększy czytelność kodu, ponieważ łatwiej rozumie się działanie poszczególnych części programu niż każdej jego linii. Jest to pomocne zwłaszcza w sytuacji, gdy programista musi przeanalizować i zrozumieć program napisany przez kogoś innego.

Użycie zmiennych

Wewnątrz procedury można korzystać ze wszystkich zmiennych używanych w programie. Przykładowo, poniższy program pobiera od użytkownika dwie liczby i wyświetla większą z nich. Warto zwrócić uwagę, że zmienna *max* używana jest i w procedurze i poza nią.

```
TextWindow.Write("Podaj pierwszą liczbę: ")
num1 = TextWindow.ReadNumber()
TextWindow.Write("Podaj drugą liczbę: ")
num2 = TextWindow.ReadNumber()

FindMax()
TextWindow.WriteLine("Większą liczbą jest: " + max)

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

Wynik działania program może wyglądać następująco:

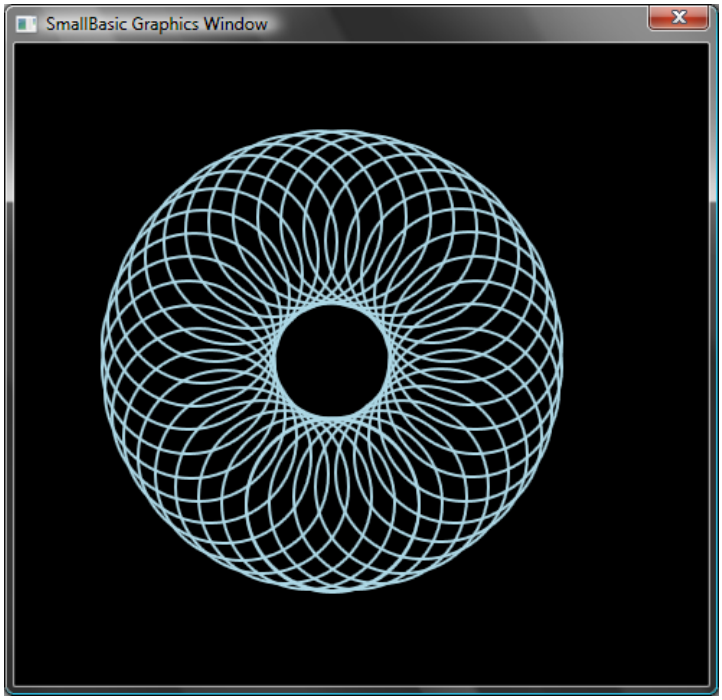


Rysunek 45 – Szukanie większej liczby

Kolejny przykładowy program pokazuje jeszcze inne użycie procedur. Tym razem, w programie wyświetlającym grafikę na ekranie, obliczane są współrzędne punktów x oraz y. Procedura **DrawCircleUsingCenter** korzysta z tych zmiennych, aby narysować okrąg. Wbudowane polecenie DrawEllipse rysuje okręgi o zadanych lewej i górnej krawędzi. Tymczasem, w przypadku procedury, można ją wywoływać, podając w zmiennych x oraz y środek okręgu. Procedura przeliczy gdzie dla danego środka wypadnie lewa i górna krawędź i narysuje odpowiedni okrąg. Można to oczywiście zrobić w programie, ale w ten sposób całość się komplikuje, a tak – jest czysta i zrozumiała.

```
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
```



Rysunek 46 – Przykład procedur w grafice

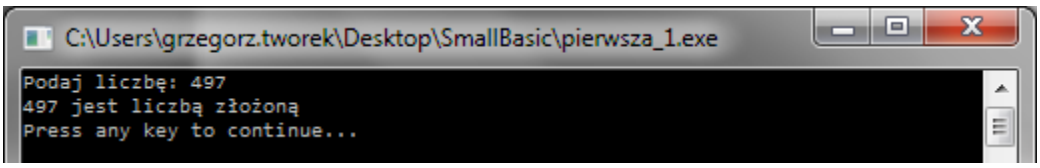
Wywołania procedur w pętlach

Bardzo często, wywołania procedur umieszczane są wewnątrz pętli. Oznacza to, że procedura będzie uruchamiana za każdym razem przy wykonaniu poleceń pętli, ale często jest tak, że przy każdym wywołaniu, jakaś zmienna będzie miała inną wartość. Przykładowo, wywołać tak można procedurę *PrimeCheck* sprawdzającą, czy dana liczba jest liczbą pierwszą czy nie. Można łatwo napisać program, który pobiera od użytkownika liczbę i sprawdza przy użyciu tej procedury czy jest ona pierwsza czy złożona.

```
TextWindow.Write("Podaj liczbę: ")
i = TextWindow.ReadNumber()
isPrime = "True"
PrimeCheck()
If (isPrime = "True") Then
  TextWindow.WriteLine(i + " jest liczbą pierwszą")
```

```
Else
  TextWindow.WriteLine(i + " jest liczbą złożoną")
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
```

Procedura PrimeCheck bierze wartość zmiennej i, po czym próbuje ją dzielić przez kolejne liczby całkowite, sprawdzając czy dzieli się ona bez reszty. Jeżeli dzieli się bez reszty, to i nie jest liczbą pierwszą. W takim przypadku procedura ustawia wartość zmiennej isPrime na "False" i kończy swoje działanie. Jeżeli liczba nie ma całkowitego podzielnika, procedura kończy swoje działanie bez ustawienia wartości zmiennej isPrime i wartość ta pozostaje ustawiona na "True".



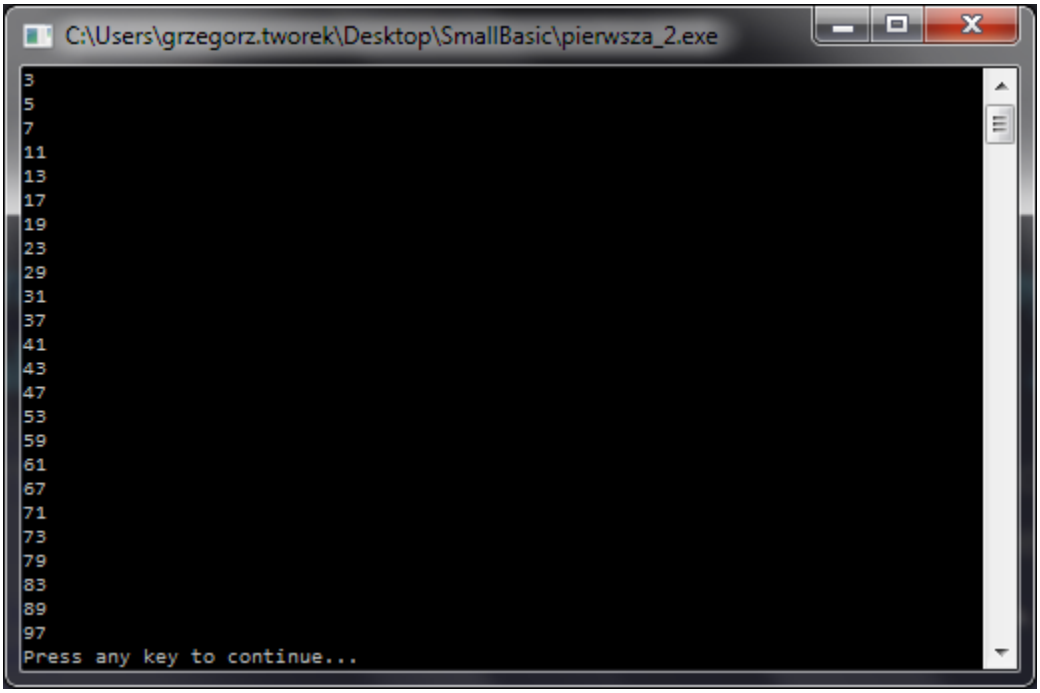
Rysunek 47 – Sprawdzanie liczb pierwszych

Teraz, mając już procedurę do sprawdzania liczb pierwszych, można przerobić program i wywołać ją dla kolejnych wartości i od 3 do 100. Zmodyfikowanie oryginalnego programu jest bardzo proste i wystarczy zamiast pobierać liczbę od użytkownika, użyć pętli For. Dzięki temu, procedura przy każdym wywołaniu otrzyma nową wartość zmiennej i do sprawdzenia.

```
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
```

Pętla For nadaje kolejne wartości zmiennej i. Za każdym razem wywoływana jest procedura sprawdzająca czy i jest liczbą pierwszą. Dzięki temu, procedura sprawdza wszystkie liczby od 3 do 100 wyświetlając na ekranie tylko liczby pierwsze. Efekt działania programu wygląda następująco:



Rysunek 48 – Liczby pierwsze

Rozdział 10

Tablice

Użycie zmiennych jest stosunkowo proste. W każdym razie powinno być proste dla kogoś, kto przebrnął przez poprzednie rozdziały podręcznika.

Warto teraz wrócić do programu, który opisany został wiele stron wcześniej:

```
TextWindow.Write("Podaj swoje imię: ")
name = TextWindow.Read()
TextWindow.WriteLine("Witaj " + name)
```

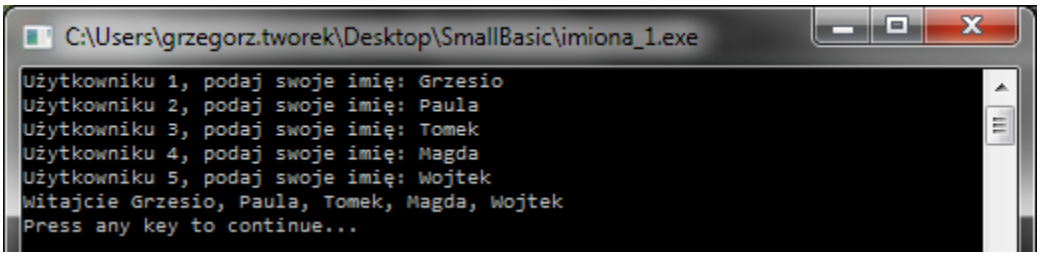
W tym programie, wprowadzone dane przechowywane są w zmiennej name. Jest ona wykorzystywana później do powitania użytkownika.

W przypadku, gdyby programista chciał napisać program witający pięciu użytkowników, mogłoby to wyglądać następująco:

```
TextWindow.Write("Użytkowniku 1, podaj swoje imię: ")
name1 = TextWindow.Read()
TextWindow.Write("Użytkowniku 2, podaj swoje imię: ")
name2 = TextWindow.Read()
TextWindow.Write("Użytkowniku 3, podaj swoje imię: ")
name3 = TextWindow.Read()
TextWindow.Write("Użytkowniku 4, podaj swoje imię: ")
name4 = TextWindow.Read()
TextWindow.Write("Użytkowniku 5, podaj swoje imię: ")
name5 = TextWindow.Read()
```

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

Po uruchomieniu, program wygląda następująco:



Rysunek 49 – Bez użycia tablic

Oczywiste jest, że powinna istnieć metoda zrobienia tego lepiej, tym bardziej, że komputery są idealne do wykonywania powtarzających się wielokrotnie czynności. Pisanie tego samego fragmentu dla każdego użytkownika wydaje się bez sensu i pętla For mogłaby się tu bardzo przydać, ale z drugiej strony – jak zapamiętać imię każdego z nich? I w tym miejscu pojawiają się tablice (ang. Arrays).

Czym jest tablica?

Tablica jest specjalnym rodzajem zmiennej, który może przechować więcej niż jedną wartość na raz. Oznacza to w praktyce, że zamiast używać **name1, name2, name3, name4** oraz **name5** aby przechować imiona pięciu użytkowników, można użyć po prostu tablicy **name** i przechować w niej pięć imion. Robi się to przy pomocy tak zwanych indeksów. W tym konkretnym przypadku, **name[1], name[2], name[3], name[4]** oraz **name[5]** mogą przechować różne wartości. Liczby od 1 do 5 są tutaj indeksami dla tablicy.

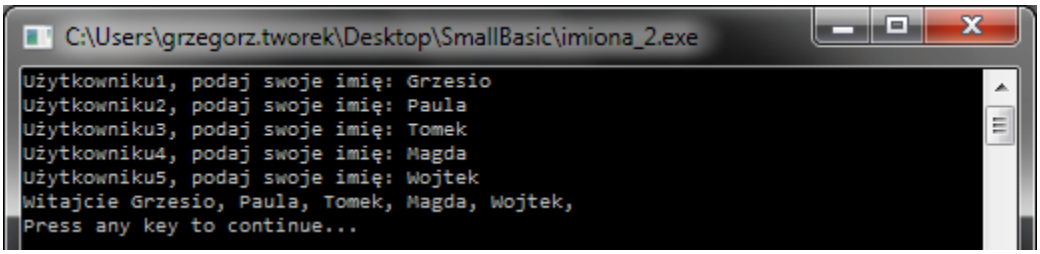
Mimo, że pozornie wydaje się, że tak naprawdę program nic nie zyskuje na tym, że zamiast name1, name2 i innych używa się **name[1]**, **name[2]**, **name[3]**, **name[4]** and **name[5]**, to jednak różnica jest znacząca. Name jest jedną zmienną. A indeksem może być inna zmienna. I to właśnie sprawia, że tablice świetnie spisują się w pętlach.

Dzięki wykorzystaniu tablic, program pytający pięciu użytkowników o imiona, wygląda następująco:

```
For i = 1 To 5
  TextWindow.Write("Użytkowniku" + i + ", podaj swoje imię: ")
  name[i] = TextWindow.Read()
EndFor
```

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

Jest to oczywiście znacznie prostsze niż wersja z pięcioma zmiennymi. Tym bardziej, że program ten można błyskawicznie przerobić tak, aby pytał o imiona dziesięciu albo i stu użytkowników.Warto zwrócić uwagę na pogrubione linie. Pierwsza z nich czyta imię użytkownika i zapisuje je jako wartość w tablicy a druga używa tej wartości do powitania. Wartość przechowywana w **name[1]** nie jest w żaden sposób naruszana przez to, co dzieje się w **name[2]** mimo, że tablica name jest cały czas tylko jedną zmienną. Jej pozycje można jednak traktować zupełnie niezależnie, jako różne wartości znane pod tą samą nazwą.



Rysunek 50 – Z użyciem tablic

Powyższy program działa tak samo jak wersja bez tablic, z jednym małym wyjątkiem: na końcu powitania wyświetla zbędny przecinek. Można to prosto poprawić, dodając warunek wewnątrz pętli For:

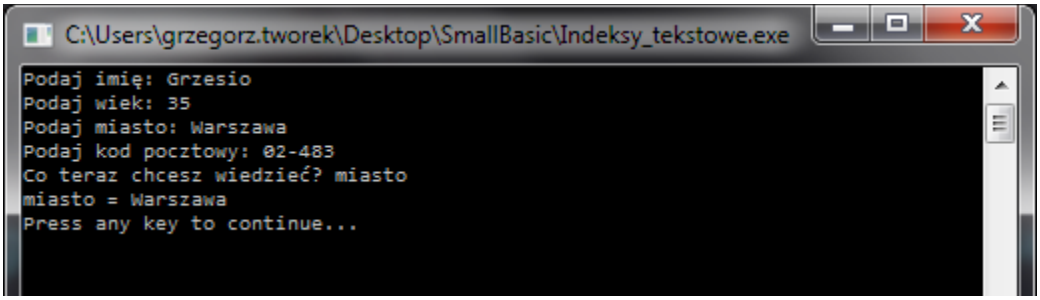
```
TextWindow.Write("Witajcie ")
For i = 1 To 5
  TextWindow.Write(name[i])
  If i < 5 Then
    TextWindow.Write(", ")
  EndIf
EndFor
TextWindow.WriteLine("")
```

Indeksowanie tablicy

W poprzednich przykładach, indeksami w tablicy były liczby. Choć jest to częste, to jednak nigdzie nie jest powiedziane, że w takim przypadku, można używać tylko liczb. Co więcej, zdarza się, że użycie

tekstów w roli indeksów jest bardzo wygodne. Poniższy program pyta użytkownika o kilka danych a na końcu wypisuje żądaną informację na ekranie.

```
TextWindow.Write("Podaj imię: ")
user["imię"] = TextWindow.Read()
TextWindow.Write("Podaj wiek: ")
user["wiek"] = TextWindow.Read()
TextWindow.Write("Podaj miasto: ")
user["miasto"] = TextWindow.Read()
TextWindow.Write("Podaj kod pocztowy: ")
user["kod"] = TextWindow.Read()

TextWindow.Write("Co teraz chcesz wiedzieć? ")
index = TextWindow.Read()
TextWindow.WriteLine(index + " = " + user[index])
```



Rysunek 51 – Użycie indeksów tekstowych

Tablice wielowymiarowe

W przypadku, gdyby programista chciał napisać program do obsługi książki telefonicznej, musiałby przechować imię i telefon wielu znajomych. W takiej sytuacji nawet użycie tablic i poznanych dotąd mechanizmów nie sprawi, że jego napisanie będzie proste. Jak w jednej tablicy przechować imiona, nazwiska i numeru dla wielu osób tak, żeby móc potem łatwo odnaleźć właściwą informację?

Zgodnie z ogólnie przyjętą w Small Basic regułą, indeksy tekstowe są niewrażliwe na wielkość liter. User["miasto"] oznacza to samo co user["Miasto"].

W takim przypadku, najwygodniej jest użyć dwóch zestawów indeksów, zwanych często wymiarami tablicy. Pierwszym indeksem jest przezwisko znajomego a drugim – słowo "name" (imię) lub "phone" (telefon), określające jaka informacja jest przechowywana w danej komórce tablicy.

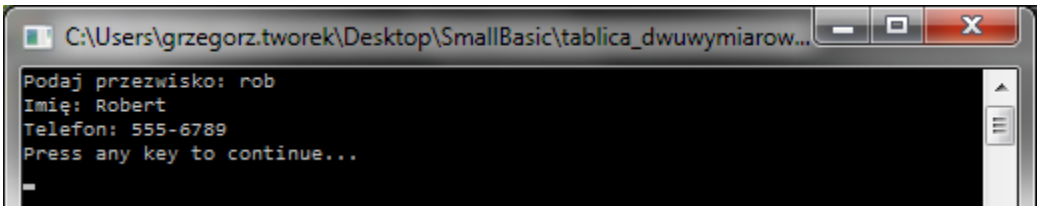
Przykładowy program mógłby wyglądać następująco:

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

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

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

Jako, że tablica **friends** ma dwa indeksy, nazywana jest tablicą dwuwymiarową.

Po wypełnieniu pól w tablicy, program może zapytać o przezwisko znajomego i wyświetlić odpowiednie dane na jego temat. Pełny program wygląda następująco:

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

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

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

TextWindow.Write("Podaj przezwisko: ")
nickname = TextWindow.Read()

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



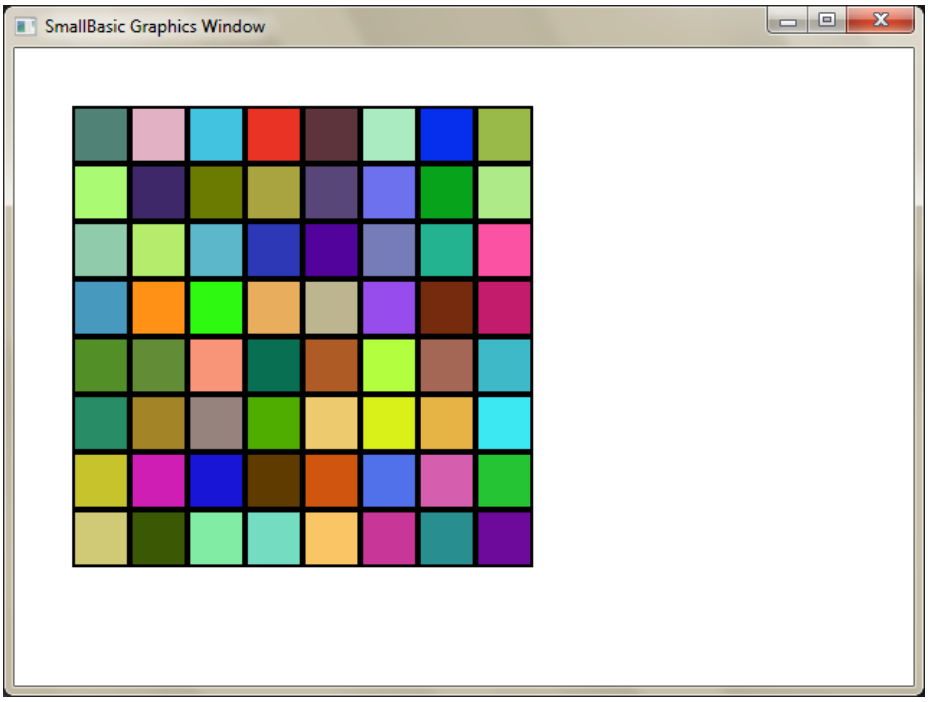
Rysunek 52 – Prosta książka telefoniczna

Użycie tablic jako reprezentacji tabel

Bardzo typowym użyciem dwuwymiarowych tablic jest programowa reprezentacja tabel lub siatek. Tabele mają wiersze i kolumny, co bardzo prosto jest przedstawić w dwuwymiarowej tablicy. Krótki program pokazujący całą ideę wygląda następująco:

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

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

Program tworzy prostokąty (AddRectangle) i układa je w siatce 8x8. Dodatkowo, trzyma ich dane w tablicy. Dzięki temu, łatwo można sięgnąć po odpowiedni prostokąt, w sytuacji, gdy jest on potrzebny w programie.

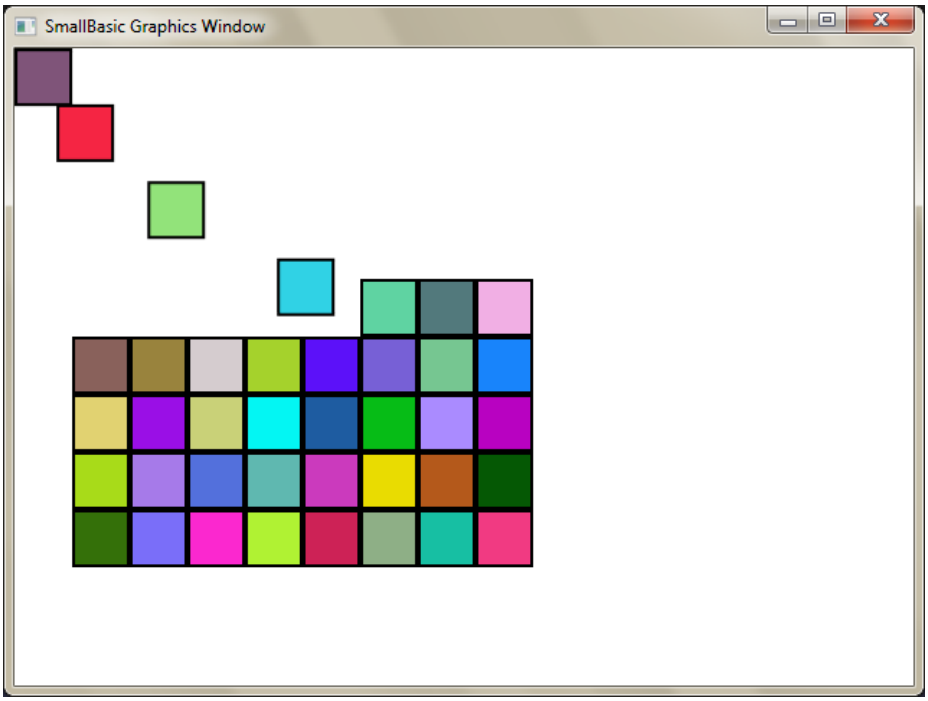


Rysunek 53 – Prostokąty na siatce

Przykładowo, dodając do poprzedniego programu kilka linii, można przemieścić prostokąty do lewego górnego rogu okna.

```
For r = 1 To rows
  For c = 1 To columns
```

```
    Shapes.Animate(boxes[r][c], 0, 0, 1000)
    Program.Delay(300)
  EndFor
EndFor
```



Rysunek 54 – Śledzenie prostokątów

Rozdział 11

Zdarzenia i interakcje

W pierwszych rozdziałach podręcznika wprowadzono pojęcie obiektu mającego właściwości i operacje (metody). Oprócz tego, obiekty mogą mieć swoje zdarzenia (ang. ***Events***). Zdarzenia są specjalnymi sygnałami, które są wysyłane na przykład, gdy użytkownik ruszy myszką lub kliknie. W pewnym sensie, zdarzenia są przeciwieństwem operacji. W przypadku operacji, to programista mówi obiektowi, że powinien coś zrobić. W przypadku zdarzenia, to obiekt powiadamia, że wydarzyło się coś wartego uwagi.

Gdzie przydają się zdarzenia?

Mechanizm zdarzeń jest kluczowy dla wszelkich rozwiązań, w których program nawiązuje interakcję z użytkownikiem. Na przykład, w przypadku gry w kółko i krzyżyk, użytkownikowi trzeba pozwolić na wykonanie ruchu. Gdy użytkownik kliknie na planszy, program dostaje przy pomocy zdarzenia powiadomienie, że to nastąpiło. Mimo, że całość brzmi na pierwszy rzut oka nieco niezrozumiale i że trochę różni się od dotychczasowego doświadczenia z językiem Small Basic, zdarzenia i sposób ich wykorzystania są proste do zrozumienia na przykładach.

Poniżej znajduje się prosty program złożony z jednego polecenia i jednej procedury. Procedura używa operacji GraphicsWindow.*ShowMessage* (pokaż komunikat) żeby wyświetlić użytkownikowi powiadomienie.

```
GraphicsWindow.MouseDown = OnMouseDown

Sub OnMouseDown
  GraphicsWindow.ShowMessage("Kliknąłeś.", "Hej!")
EndSub
```

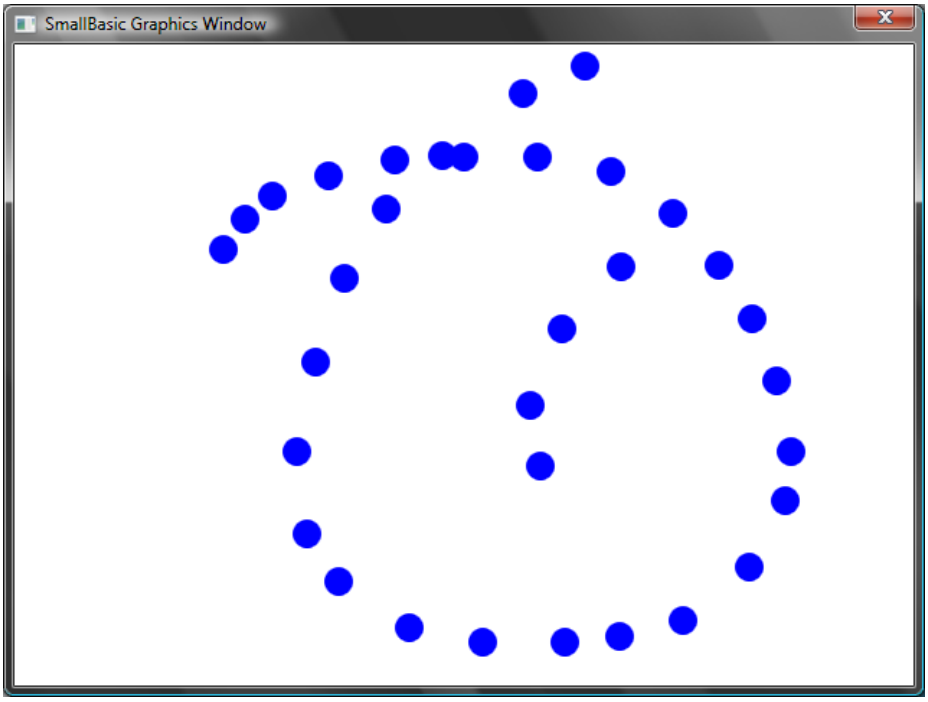
Najciekawszą częścią program jest linia, w której do obsługi zdarzenia **MouseDown** przypisywana jest procedura *OnMouseDown*. Zdarzenie MouseDown przypomina właściwość obiektu, z wyjątkiem tego, że przypisuje mu się procedurę zamiast wartości. Właśnie ta cecha wyróżnia zdarzenia. Gdy zajdzie określona sytuacja, automatycznie uruchamiana jest procedura, która została przypisana. W tym przypadku procedura *OnMouseDown* uruchamiana jest za każdym razem, gdy użytkownik kliknie w okno graficzne. Program mimo bardzo prostej konstrukcji działa zupełnie poprawnie i warto go samodzielnie wypróbować. Działanie jest zgodne z logiką – za każdym razem, gdy kliknie się w okno graficzne, pojawi się komunikat taki, jak na ilustracji.

Rysunek 55 – Reakcja na zdarzenie

Ten sposób obsługi zdarzeń ma bardzo duże możliwości i pozwala na tworzenie naprawdę zaawansowanych aplikacji. Czasem, tak skonstruowane programy nazywa się programami sterowanymi zdarzeniami.

Można oczywiście zmodyfikować procedurę *OnMouseDown* tak, aby robiła jakieś bardziej ambitne rzeczy niż wyświetlanie powiadomienia. Na przykład, tak jak w poniższym programie, można w każdym kliknięty miejscu rysować dużą niebieską kropkę.

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

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



Rysunek 56 – Obsługa zdarzenia MouseDown

Warto zwrócić uwagę, że w programie wykorzystano właściwości *MouseX* oraz *MouseY* w celu zbadania bieżącego położenia kursora myszy. Wiedząc gdzie nastąpiło kliknięcie, narysowanie odpowiedniego kółka nie jest już problemem.

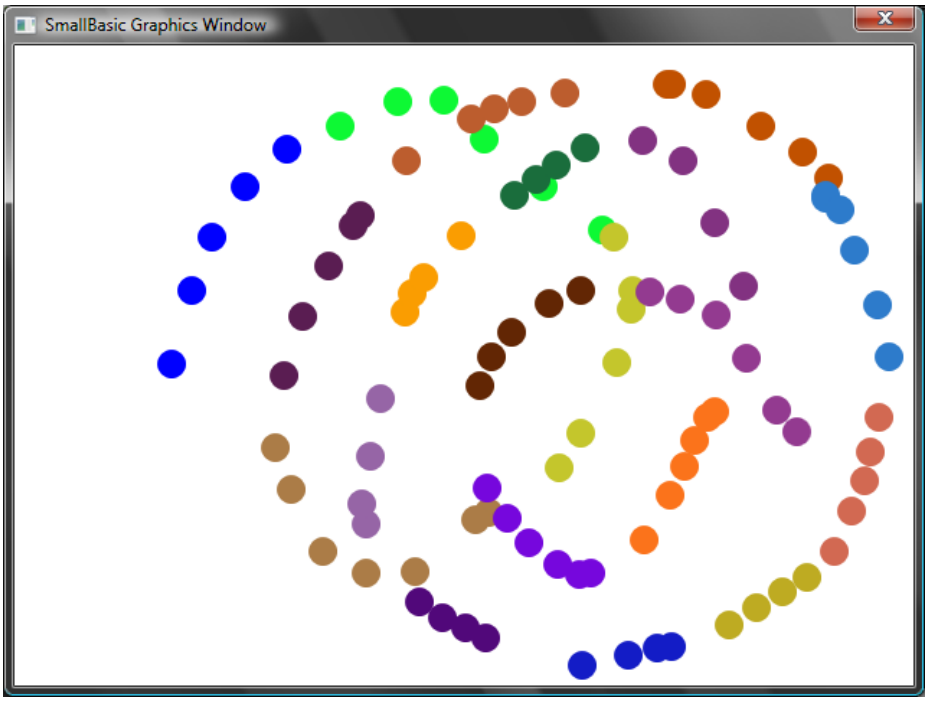
Obsługa wielu zdarzeń

Tak naprawdę, nie istnieją żadne ograniczenia dla ilości obsługiwanych zdarzeń. W szczególnym przypadku, można do ich obsługi użyć jednej procedury, ale zostanie ona wywołana tylko raz. W przypadku, gdy do zdarzenia przypisze się dwie różne procedury, liczy się wyłącznie ostatnie przypisanie.

Aby zilustrować powyższy opis, do omawianego programu można dodać obsługę naciśnięć klawiszy na klawiaturze. Naciśnięcie klawisza będzie zmieniało kolor kropek na inny a naciskanie przycisku myszy zadziała tak, jak dotąd.

```
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
```



Rysunek 57 – Obsługa wielu zdarzeń

Po uruchomieniu, klikanie w oknie zostawia niebieskie kropki. Teraz, gdy zostanie naciśnięty jakikolwiek znak na klawiaturze, kolor kropek się zmieni. Naciśnięcie klawisza wywołuje procedurę *OnKeyDown* zmieniającą kolor aktualnego pędzla na losowy. Kolejne kliknięcia myszą nadal rysują kropki, ale ich kolor jest zawsze kolorem pędzla, więc jest inny niż poprzednio używany niebieski.

Program Paint

Wyposażony w procedury i zdarzenia, programista może napisać program pozwalający użytkownikowi na rysowanie po ekranie. Napisanie takiego programu jest zaskakująco proste. Pierwszym krokiem może być napisanie programu, który zostawia na ekranie ślad tam, gdzie użytkownik przemieści kursor myszy.

```
GraphicsWindow.MouseMove = OnMouseMove

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

```
  prevX = x
  prevY = y
EndSub
```

Po uruchomieniu programu widać, że pierwsza linia zawsze zaczyna się od lewego górnego rogu, czyli punktu (0,0). Aby usunąć tę usterkę, można napisać obsługę zdarzenia *MouseDown* i zapamiętywać wartości *prevX* i *prevY* gdy zdarzenie jest wywoływane.

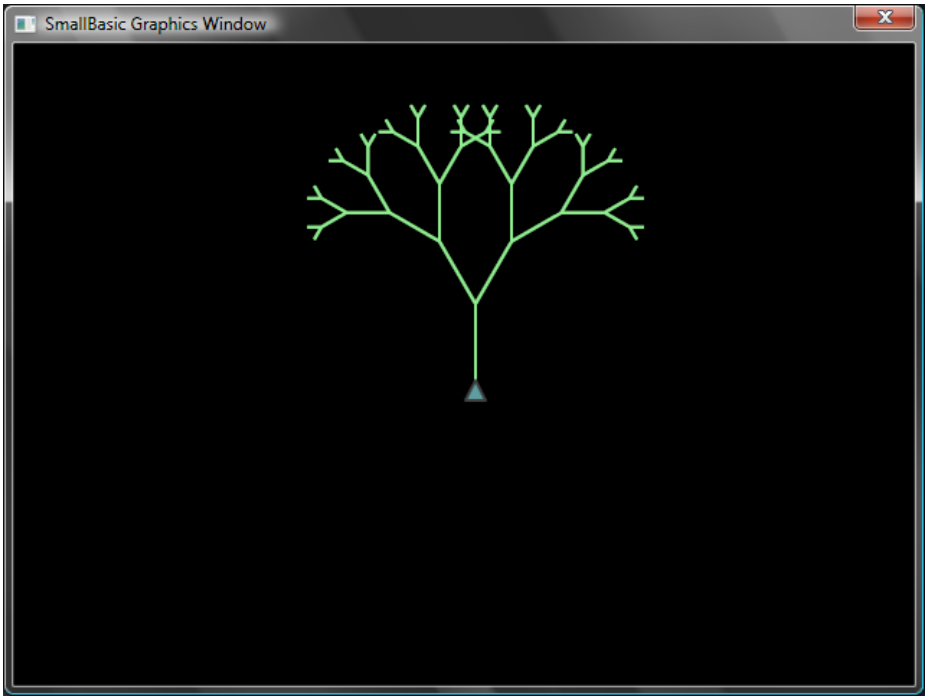
Dodatkowo, w programie chodzi o to, aby rysować tylko, gdy wciśnięty jest lewy przycisk myszy. Gdy nie jest wciśnięty, na ekranie nie powinien pojawiać się żaden ślad. Aby to osiągnąć, można użyć właściwości *IsLeftButtonDown* obiektu **Mouse**. Właściwość ta pozwala na sprawdzenie czy lewy przycisk myszy jest wciśnięty czy nie. Gdy wartość właściwości jest równa "True", program rysuje linię. W przeciwnym przypadku nie zostawia na ekranie żadnego śladu.

```
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
```

Dodatek A

Zabawne przykłady

Fraktal w grafice żółwia



Rysunek 58 – Żółw rysujący fraktal

```
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
```

Zdjęcia z serwisu Flickr

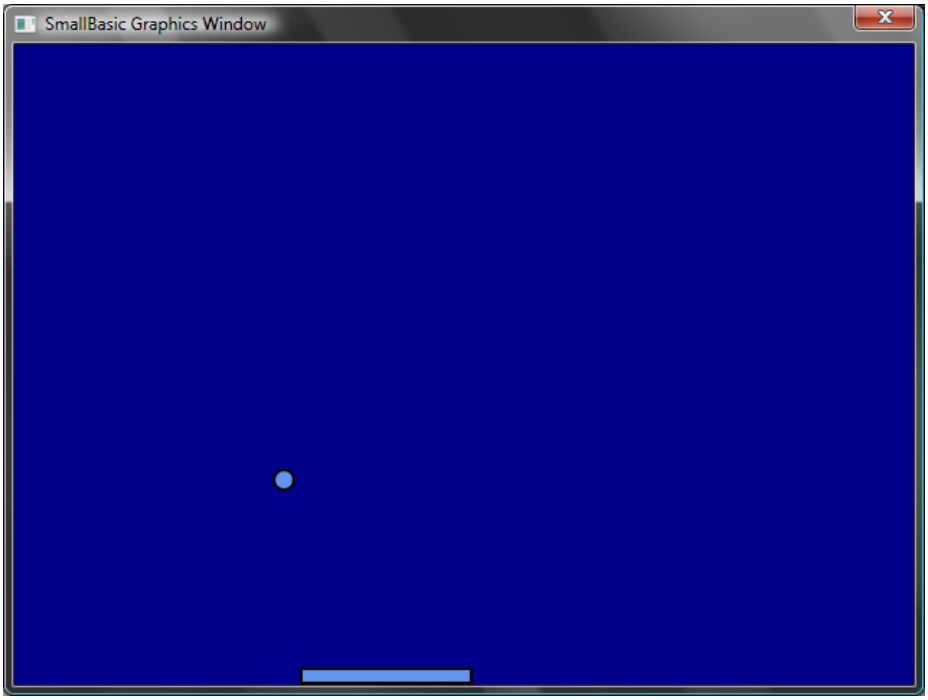
Rysunek 59 – Pobieranie zdjęcia z Flickr

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

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

Dynamiczne tapety na pulpicie

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

Odbijanie piłeczki



Rysunek 60 – Odbijanie piłeczki

```
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("Przegrałeś", "Koniec")

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

Dodatek B

Kolory

W niniejszym dodatku znajduje się lista kolorów pogrupowanych według ich odcienia.

Czerwone

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

Różowe

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

Pomarańczowe

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

Żółte

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

PaleGoldenrod	#EEE8AA
Khaki	#F0E68C
DarkKhaki	#BDB76B

Fioletowe

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

Zielone

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

Niebieskie

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

Brązowe

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

Białe

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

Szare

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