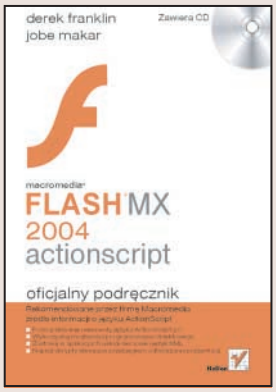
Macromedia Flash MX 2004 ActionScript. Oficjalny podręcznik



Autorzy: Derek Franklin, Jobe Makar
Tłumaczenie: Piotr Cieślak (rozdz. 1 - 7, 18), Marcin Samodulski (wstęp, rozdz. 15 - 17, 19 - 21)
ISBN: 83-7361-533-4
Tytuł oryginału: Macromedia Flash MX 2004 ActionScript: Training from the Source
Format: B5, stron: 728

Polecane przez firmę Macromedia źródło informacji o języku ActionScript

- Poznaj składnię i elementy języka ActionScript 2.0
- Wykorzystaj możliwości programowania obiektowego
- Zastosuj w aplikacjach usługi sieciowe i język XML
- Napisz skrypty sterujące przebiegiem odtwarzania prezentacji

Macromedia Flash MX 2004 i Macromedia Flash MX 2004 Professional to aplikacje służące do tworzenia publikacji multimedialnych opartych na grafice wektorowej. Zaimplementowany w nich rozbudowany obiektowy język programowania ActionScript daje projektantom nieograniczoną swobodę twórczą. Za jego pomocą można stworzyć proste skrypty nawigacyjne, rozbudowane mechanizmy obsługi danych, narzędzia weryfikacji danych wprowadzanych przez użytkowników, narzędzia ładowania plików z dysków oraz elementy interfejsu użytkownika.

„Macromedia Flash MX 2004 ActionScript. Oficjalny podręcznik" to zbiór ćwiczeń przygotowanych przy współpracy z firmą Macromedia — producentem Flasha. Dzięki przykładom przedstawianym w kolejnych lekcjach poznasz możliwości języka ActionScript 2.0. Dowiesz się wszystkiego o jego elementach i składni, nauczysz się korzystać z danych zewnętrznych i kontrolować elementy prezentacji tworzonej we Flashu.

- Składnia języka ActionScript
- Zdarzenia i ich obsługa
- Uzyskiwanie dostępu do elementów prezentacji
- Klasy obiektów
- Definiowanie i wykorzystywanie funkcji
- Struktury sterujące — pętle i wyrażenia warunkowe
- Obsługa, pobieranie, wysyłanie i walidacja danych
- Korzystanie z języka XML
- Obsługa plików multimedialnych
- Testowanie skryptów i usuwanie z nich błędów

Poznaj możliwości języka ActionScript 2.0 w praktyce, korzystając z podręcznika polecanego przez firmę Macromedia.

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

9 Automatyzacja skryptów za pomocą pętli

Każdy z nas musi czasem wykonywać czynności, które wymagają wielokrotnego powtarzania przynajmniej jednego zadania. Jeśli np. wysyłamy 100 zaproszeń na ślub, powtarzającymi się czynnościami są składanie papieru, sklejanie kopert i naklejanie znaczków. Trzeba je wykonać 100 razy. W języku ActionScript zbiory powtarzających się akcji nazywamy *pętlami*. Pozwalają one na uruchomienie danych instrukcji dowolną ilość razy, co oznacza, że zamiast ponownie je przepisywać, wystarczy napisać je raz i umieścić w pętli. W tej lekcji zobaczysz, jak używa się trzech typów pętli dostępnych w języku ActionScript.

Rozwijana lista oraz siatka o wymiarach 2 na 2, wykorzystane w tej aplikacji, zostaną utworzone za pomocą pętli.

Czego się nauczysz?

W czasie tej lekcji:

- Odkryjesz, jak użyteczne są pętle,
- Poznasz typy pętli,
- Ustawisz warunki wykonania pętli,
- Utworzysz pętle zagnieżdżone,
- Użyjesz wyjątków.

Przybliżony czas

Ta lekcja zajmie w przybliżeniu 45 minut.

Materiały do lekcji

Pliki startowe:

Lekcja09\Assets\pictureShow1.fla

Lekcja09\Assets\phoneNumberSearch1.fla

Gotowe projekty:

pictureShow3.fla

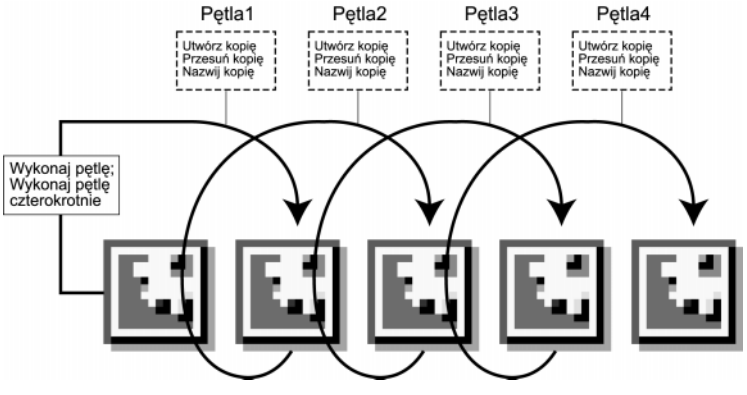
phoneNumberSearch2.fla

Dlaczego pętle są użyteczne?

Pętle pozwalają programowi Flash na powtarzanie akcji (lub ich zbioru), co oznacza, że używając tylko kilku wierszy kodu w języku ActionScript, możesz uruchomić daną instrukcję nawet kilka tysięcy razy. W tym języku pętle wykorzystuje się do zadań, które bez nich byłyby trudne lub niemożliwe do wykonania np.do:

- automatycznego tworzenia rozwijanych list,
- sprawdzania danych,
- wyszukiwania tekstu,
- powielania klipów filmowych,
- kopiowania zawartości jednej tabeli do innej,
- wykrywania w grach zderzeń obiektów.

Pętli można użyć do automatyzacji dowolnej ilości zadań, takich jak tworzenie instancji klipów filmowych. Przypuśćmy, że projekt wymaga równomiernego rozmieszczenia 100 instancji tego samego klipu. Wystarczy wówczas przeciągnąć jedną z nich z biblioteki, a następnie napisać cztero lub pięciowierszowy kod powodujący jej stukrotne powielenie i rozmieszczenie kopii w równomiernych odstępach. Jest to znacznie łatwiejsze niż stukrotne przeciąganie instancji z biblioteki i ręczne pozycjonowanie ich na scenie.



Pętle mogą być dostosowywane do wymagań programu. Przypuśćmy, że napisałeś instrukcję powodującą automatyczne tworzenie menu zawierającego 15 przycisków. Nieznacznie ją modyfikując, możesz dodawać i usuwać jego elementy. Wykonanie tego ręcznie wymaga dodania lub usunięcia zaznaczonego elementu, a następnie przesunięcia wszystkich pozostałych w górę lub w dół, a niekiedy nawet modyfikacji istniejącego kodu.

W miarę postępów w przyswajaniu wiedzy zawartej w tej lekcji (oraz w całej reszcie książki) docenisz rolę pętli w skryptach.

Typy pętli

Język ActionScript zawiera trzy typy pętli, z których wszystkie wykonują akcję (lub ich zbiór) wówczas, gdy dany warunek jest prawdziwy.

Pętla while

Składnia tego popularnego typu pętli jest następująca:

```
while (someNumber < 10) {
  // wykonaj te akcje
}
```

Wyrażenie `someNumber < 10` jest warunkiem, który określa liczbę *iteracji* (powtórzeń) pętli. Podczas każdej z nich wykonane zostają zawarte w niej akcje. Sposoby określania warunku (i wychodzenia z pętli) zostały opisane w podrozdziale „Pisanie i rozumienie warunków pętli”.

Oto przykład pętli `while`:

```
while (myClip_mc._y < 0) {
  myClip_mc._y += 3;
}
```

Powyższy skrypt przesuwa klip filmowy *myClip_mc* wzdłuż osi *y* aż do momentu, gdy jego pozycja będzie większa od 0.

Pętla for

Instrukcja `for` jest pętlą, której działanie opiera się na zwiększaniu lub pomniejszaniu wartości zmiennej. Pozwala ona na jej inicjalizację, ustalenie warunków wykonania zawartych w niej komend oraz zwiększanie lub zmniejszanie wartości zmiennej przy każdej iteracji — wszystko to w jednej instrukcji języka ActionScript. Jest ona zwykle używana do wykonywania akcji (lub ich zbioru) w zależności od wartości zmiennej — np. przy operowaniu tablicą lub wykonywaniu akcji związanych z listą klipów filmowych. Oto składnia pętli `for`:

```
for (var someNumber:Number= 0; someNumber < 10; ++someNunber) {
   // wykonaj te akcje
}
```

Trzy elementy umieszczone w nawiasie i oddzielone od siebie średnikami są używane do określenia liczby iteracji pętli. W powyższym przykładzie tworzymy zmienną `someNumber`, a następnie przyporządkowujemy jej wartość początkową `0`. Później instrukcje wewnątrz pętli są wykonywane do momentu, gdy wartość zmiennej jest mniejsza od `10`. Ostatni element umieszczony w nawiasie mówi, że przy każdej iteracji jest ona zwiększana o `1`, co ostatecznie powoduje, że zmienna `someNumber` otrzymuje wartość `10`, oznaczającą koniec pętli.

Pętla `for` jest używana głównie do powtarzania zbioru akcji określoną ilość razy. Poniżej znajduje się przykład jej zastosowania:

```
for (var i:Number=0; i<10; ++i) {
  myClip_mc.duplicateMovieClip("myClip_mc" + i, i);
}
```

Powyższy kod powoduje dziesięciokrotne powielenie instancji klipu *myClip_mc*.

Pętla for…in

W tym typie pętli jest wykorzystywana lista wszystkich właściwości obiektu. Oto jego składnia:

```
for (var i:String in someObject) {
   trace(i);
}
```

Wartość `i` jest zmienną, która przy każdej iteracji przechowuje nazwę właściwości. Może ona zostać użyta w akcjach umieszczonych wewnątrz pętli. Przyjrzyjmy się następującemu skryptowi jako przykładowi jej praktycznego zastosowania:

```
var car:Object = new Object();
car.color = "red";
car.make = "BMW";
car.doors = 2;
var result:String;
for (var i:String in car) {
  result += i + ": " + car[i] + newline;
}
```

Na początku utworzony zostaje obiekt o nazwie `car`. Kolejne trzy wiersze przyporządkowują mu właściwości (które można traktować jako zmienne znajdujące się wewnątrz obiektu) oraz odpowiadające im wartości. Następnie użyta zostaje pętla `for...in`, powodująca kolejne przyporządkowywanie wszystkich właściwości zmiennej `i`. Jej wartość zostaje użyta w akcji znajdującej się wewnątrz pętli. Pod koniec, w zmiennej `result` jest przechowywany tekst zawierający nazwy wszystkich właściwości wraz z ich wartościami.

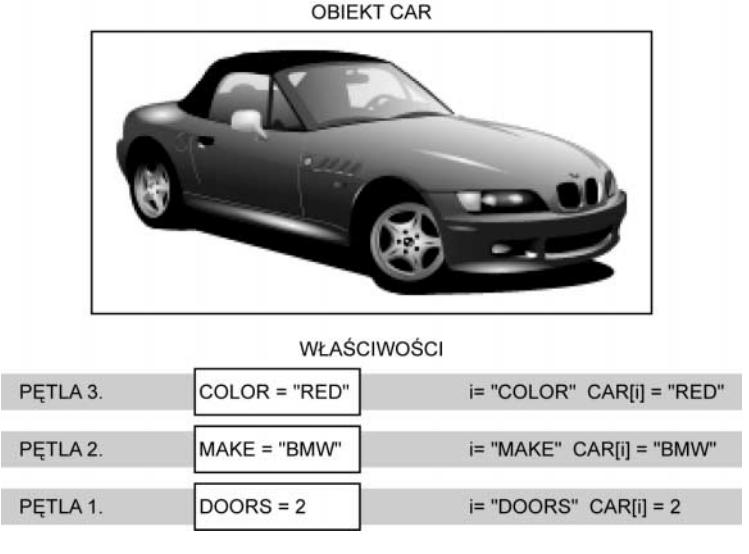
W czasie pierwszej iteracji pętli zmiennej `i` przyporządkowana zostaje wartość `doors` (ponieważ jest ona nazwą ostatniej zdefiniowanej właściwości). Wyrażenie ustalające wartość przechowywaną w zmiennej `result` wygląda następująco:

```
result = result + "doors" + ": " + 2 + newline;
```

Po pierwszym wykonaniu pętli zmienna `result` będzie zawierała ciąg znaków:

```
"doors: 2"
```

W wyrażeniu przyporządkowującym wartość zmiennej `result` zmienna `i` odnosi się do nazwy właściwości (takiej jak `doors`, `make` lub `color`). Użycie sformułowania `car[i]` (tzn. umieszczenie `i` w nawiasach kwadratowych) oznacza to samo, co napisanie `car.doors` i stanowi referencję do wartości zmiennej.



Gdy pętla zostaje zakończona, zmienna `result` będzie zawierała następujący ciąg znaków:

```
"doors: 2
make: BMW
color: red"
```

Ponieważ obiekt `car` posiada trzy właściwości, pętla `for...in` użyta w tym skrypcie wykona automatycznie jedynie trzy iteracje.

Gdy tworzysz właściwość obiektu, zostaje ona zapisana w tablicy przyporządkowującej. W przypadku zwykłej tablicy referencje do zawartych w niej obiektów mogą zostać utworzone za pomocą liczby. W tablicy przyporządkowującej referencja następuje przez nazwę. Opisana w tym podrozdziale pętla `for...in` *wykorzystuje tablicę przyporządkowującą zawierającą wszystkie referencje zawarte w określonej listwie czasowej lub obiekcie.*

Ten typ pętli może zostać użyty w wielu przypadkach, np. do odnalezienia informacji, takiej jak:

- nazwa i wartość każdej zmiennej listwy czasowej lub obiektu,
- nazwa każdego obiektu umieszczonego na listwie czasowej lub w innym obiekcie,
- nazwa i wartość każdego atrybutu dokumentu napisanego w języku XML.

Pisanie i rozumienie warunków pętli

W dalszej części tej lekcji skupimy się na pętli `loop`. Zawarte w niej akcje są wykonywane po kolei, dopóki warunek użyty do jej utworzenia pozostaje prawdziwy, np.:

```
var i:Number = 0;
while (i<10) {
  // wykonaj te akcje
}
```

Warunkiem pętli jest `i < 10`. Oznacza to, że dopóki wartość zmiennej `i` jest mniejsza od 10, wyrażenie jest prawdziwe i akcje umieszczone wewnątrz pętli zostają wykonane. Powyższej instrukcji brakuje jednak najważniejszego elementu. Nie posiada ona żadnego sposobu pozwalającego na zmianę wartości wyrażenia na fałszywą. Bez niego pętla będzie wykonywana w nieskończoność, co w rzeczywistości może spowodować zawieszenie się aplikacji. Program Flash nie może zrobić nic innego aż do momentu, gdy pętla zostanie zakończona. Aby uniknąć takiej sytuacji, wartość zmiennej `i` musi zostać zwiększona i ostatecznie osiągnąć `10`, co sprawi, że warunek pętli będzie fałszywy i zostanie ona zatrzymana.

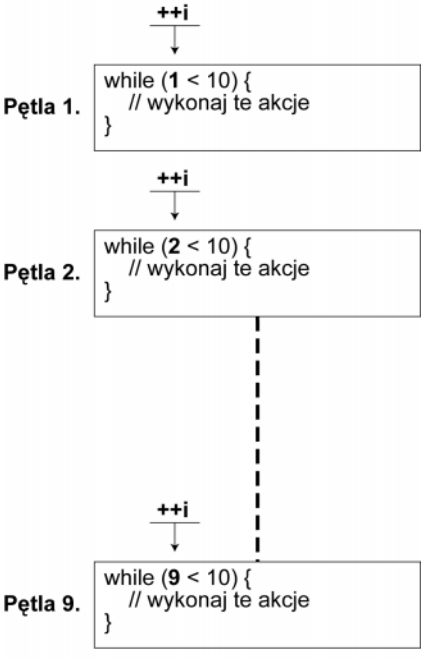
Aby zwiększyć wartość zmiennej, możesz użyć operatora ++. Oto przykład:

```
var i: Number = 0;
while (i < 10) {
  // wykonaj te akcje
  ++i
}
```

Zwiększanie wartości zmiennej `i` powoduje, że pętla wykona 10 iteracji. Wartość ta początkowo jest równa 0. Jednak przy każdym powtórzeniu pętli jej wartość jest powiększana o 1. Przy dziesiątej iteracji `i = 10`, co oznacza, że warunek `i < 10` nie jest już prawdziwy i pętla zostaje zatrzymana. Oto krótszy zapis instrukcji wykonującej to samo zadanie:

```
var i:Number = 0;
while (++i < 10) {
  // wykonaj te akcje
}
```

Powyższa pętla zostaje wykonana 9 razy. Zmiennej `i` jest początkowo przyporządkowywana wartość `0`. Jednak przy każdej iteracji (włącznie z pierwszą) zostaje ona powiększona o 1 wewnątrz instrukcji warunkowej zawartej w samej pętli. Przy dziesiątym powtórzeniu `i = 10`, co oznacza, że warunek `i < 10` nie jest już prawdziwy i pętla zostaje zakończona.



W pętli możesz również użyć operatora `--`. Sposób jego wykorzystania wygląda mniej więcej tak:

```
var i:Number = 10;
while (--i > 0) {
  // wykonaj te akcje
}
```

Możliwe jest też napisanie tego skryptu za pomocą pętli `for`:

```
for (var i:Number = 10; i>0; --i) {
  //wykonaj te akcje
}
```

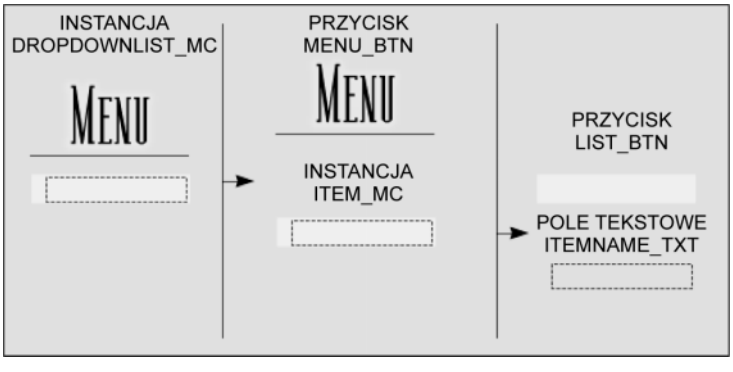
Warunek pętli nie musi zależeć od powiększanej lub pomniejszanej wartości. Może nim być dowolne wyrażenie logiczne lub wynik wywołania funkcji zwracającej prawdę lub fałsz, jak w następującym przykładzie:

```
while (someFunction()) {
  //wykonaj te akcje
}
```

W poniższym ćwiczeniu utworzysz rozwijaną listę, używając pętli `while`.

1. Otwórz plik *pictureShow1.fla*, znajdujący się w folderze *Lekcja09\Assets*.

   Główna listwa czasowa zawiera trzy warstwy: *Background* (tło), *Dynamic Elements* (elementy aktywne) i *Actions* (akcje). Wewnątrz warstwy *Actions* znajduje się cały kod napisany w języku ActionScript. Oczywiście warstwa *Background* zawiera grafikę umieszczoną w tle projektu. W warstwie *Dynamic Elements* umieszczono cztery instancje klipu filmowego. Trzy ponad sceną, w nich znajdują się rysunki, oraz jedną, o nazwie *dropDownList_mc*, umieszczoną na scenie i zawierającą przycisk o nazwie *menu_btn*, a także jeszcze jeden klip, który nazywa się *item_mc*. Instancja *item_mc* powstała z dwóch elementów: pola tekstowego o nazwie *itemName_txt* oraz przycisku widocznego jako półprzezroczyste białe pole o nazwie *list_btn*. Instancja *itemName_txt* odgrywa w tym ćwiczeniu ważną rolę, ponieważ zostanie ona powielona w procesie tworzenia menu.



2. Otwórz panel *Actions*, zaznacz klatkę warstwy *Actions* i dodaj następujące dwa wiersze skryptu:

```
var buttonNames:Array = ["Paris", "New York", "London"];
dropDownList_mc.item_mc._visible = false;
```

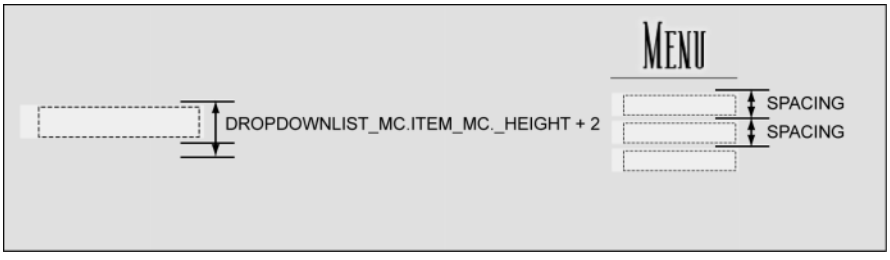
   Pierwsza z powyższych akcji tworzy tablicę o nazwie `buttonNames`, zawierającą nazwy pojawiające się w rozwijanej liście. Druga sprawia, że klip filmowy *item_mc* staje się niewidoczny (ponieważ będzie używany tylko jako szablon do tworzenia duplikatów).

3. Pod skryptem umieszczonym w punkcie 2. dodaj następującą funkcję:

```
function populateList() {
  var spacing:Number = dropDownList_mc.item_mc._height + 2;
  var numberOfButtons:Number = buttonNames.length;
}
```

   Powyższa funkcja powiela klip filmowy *item_mc*, umieszcza jego duplikaty pod przyciskiem *Menu*, a następnie zmienia tekst wyświetlany na ich instancjach.

Ponieważ generuje ona listę opcji, musimy wziąć pod uwagę odstęp w pionie między przyciskami. W pierwszej akcji funkcja tworzy zmienną o nazwie `spacing`. Jej wartość zostaje określona przez dodanie liczby 2 do wysokości klipu *item_mc*. Pętla wykorzystuje ją do ustawienia odstępu między górnymi krawędziami powielonych instancji.



Wartość zmiennej `numberOfButtons` jest określana przez długość tablicy `buttonNames` (tzn. liczbę zawartych w niej elementów). Pętla `while` (dodana w następnym punkcie), użyta do tworzenia listy, wykorzystuje ją do określenia liczby iteracji. Ponieważ długość tablicy `buttonNames` jest aktualnie równa `3` (zawiera ona trzy nazwy miast), pętla zostanie wykonana trzy razy, przy każdym powtórzeniu tworząc nowy przycisk listy. Jeśli w tablicy umieścimy kolejną nazwę miasta, wartość właściwości `length` (długość) będzie wynosiła `4`, co oznacza, że pętla automatycznie się dopasuje i utworzy cztery przyciski.

4. Dodaj do funkcji `populateList()` następujący skrypt, zaraz za ostatnim wierszem kodu:

```
var i:Number = -1;
  while (++i < numberOfButtons) {
  var name:String = "item" +i;
  dropDownList_mc.item_mc.duplicateMovieClip(name, i);
  dropDownList_mc[name].itemName_txt.text = buttonNames[i];
  dropDownList_mc[name]._x = 0;
  dropDownList_mc[name]._y = i * spacing;
  dropDownList_mc[name].pictureID = i + 1;
  dropDownList_mc[name].list_btn.onRelease = function() {
    itemClicked(this._parent.pictureID);
  };
}
```

Właśnie napisałeś pętlę `while`, która powiela instancję klipu filmowego *item_mc* dla potrzeb rozwijanej listy, umieszcza duplikaty w odpowiednich miejscach i nadaje im nazwy.

Przed definicją pętli skrypt utworzył zmienną lokalną o nazwie `i` oraz przyporządkował jej wartość `–1`.

Uwaga *Litera „i” jest często używana jako nazwa zmiennej wykorzystywanej w pętli.*

Kolejny wiersz skryptu zaczyna się od pętli `while` i definiuje warunek decydujący o tym, czy ma ona zostać wykonana. Mówi: „Dopóki wartość zmiennej `i` jest mniejsza od wartości zmiennej `numberOfButtons`, kontynuuj przeglądanie tablicy”. Na początku pętli zmienna `i` posiada wartość `0`, mimo iż początkowo jest jej przyporządkowywana wartość `–1`, operator `++` powoduje jej zwiększenie o 1 przed każdą iteracją (także przed pierwszą). Ponieważ zmienna

`numberOfButtons` posiada wartość `3` (co opisaliśmy w punkcie 3.), zaś `i` jest przy każdej iteracji zwiększana o 1, warunek pętli okaże się fałszywy po jej trzech wykonaniach.

Pierwsza akcja skryptu tworzy zmienną o nazwie `name`, której wartość zależy od aktualnej wartości zmiennej `i`. Ponieważ jest ona powiększana przy każdej iteracji, w pierwszym wykonaniu pętli zmiennej `name` zostaje przyporządkowana wartość `item0`, w drugiej — `item1` itd. Kolejny wiersz używa metody `duplicateMovieClip()` do utworzenia nowej instancji klipu filmowego *item_mc*. Jej dwa parametry są rozdzielone przecinkiem. Pierwszy przyporządkowuje tworzonemu duplikatowi nazwę instancji, zaś kolejny — jego pozycję. Jako nazwy używamy wartości zmiennej `name`, zaś jako pozycji — wartości zmiennej `i`.

W następnych czterech akcjach umieszczone w pętli wyrażenie `dropDownList_mc[name]` jest odniesieniem do nazwy utworzonego duplikatu. Jak już wspominaliśmy w lekcji 6., „Tworzenie i przetwarzanie danych", taka składnia umożliwia w języku ActionScript tworzenie referencji do nazwy zmiennej. W pierwszej iteracji pętli zmiennej `name` zostaje przyporządkowana wartość `item0`. W tej sytuacji opisywane wiersze skryptu można przepisać w następujący sposób:

```
dropDownList_mc.item0.itemName_txt.text = buttonNames[i];
dropDownList_mc.item0._x = 0;
dropDownList_mc.item0._y = i * spacing;
dropDownList_mc.item0.pictureID = i + 1;
```

W każdej iteracji wartość zmiennej `name` zostaje zaktualizowana i wykorzystywana do nazwania utworzonego duplikatu. Powyższe akcje korzystają z referencji do jego instancji.

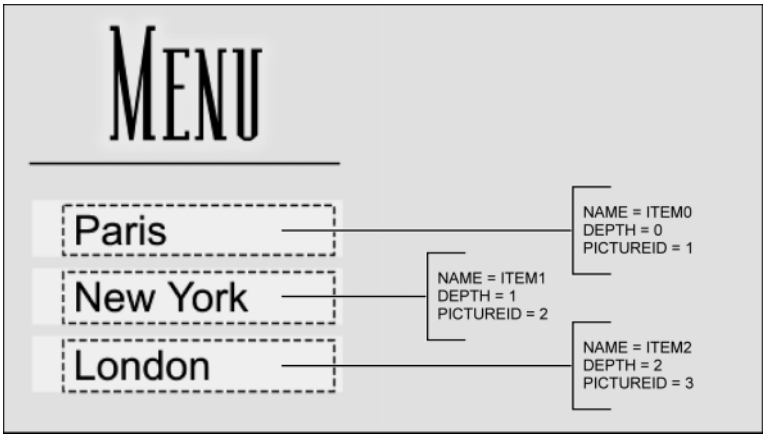
Każdy duplikat zawiera dwa elementy umieszczone w oryginalnej instancji klipu *item_mc* — biały przycisk o nazwie *list_btn* oraz pole tekstowe, noszące nazwę *itemName_txt*. Trzeci wiersz pętli

```
dropDownList_mc[name].itemName_txt.text = buttonNames[i];
```

przyporządkowuje wartość właściwości *itemName_txt.text* (oznaczającej tekst wyświetlany na przycisku), znajdującej się w duplikacie. Wartość ta zostaje ustalona przez użycie aktualnej wartości zmiennej `i` w referencji do ciągu znaków znajdującego się w tablicy `buttonNames`, utworzonej w punkcie 2. W pierwszej iteracji pętli wartością `i` jest `0`, co oznacza, że właściwości *itemName_txt.text* zostanie przyporządkowana wartość przechowywana w polu `buttonNames[0]`, czyli w pierwszym elemencie tablicy. Jest nią tekst „Paris".

Kolejne dwa wiersze skryptu umieszczają kopię instancji na scenie. Jak widać w skrypcie, każdy z duplikatów posiada tę samą współrzędną *x*, wynoszącą `0`. Współrzędna *y* jest ustalana dynamicznie. Zostaje określona przez pomnożenie zmiennej `i` przez wartość odstępu między duplikatami. Efektem tej akcji jest ich równomierne rozmieszczenie w pionie.

Po wykonaniu dwóch wierszy skryptu, umieszczających klip w odpowiednim miejscu, pętla tworzy wewnątrz kopii instancji zmienną `pictureID`. Zostaje jej przyporządkowana wartość oparta na aktualnej wartości zmiennej `i` powiększonej o 1. Zostanie ona użyta w kolejnym ćwiczeniu do określenia sposobu wyświetlania zbioru rysunków.



Na końcu pętla przyporządkowuje znajdującemu się w każdym z duplikatów przyciskowi *list_btn* procedurę obsługującą zdarzenie `onRelease`. W momencie puszczenia przycisku powoduje ona wywołanie funkcji `itemClicked()` oraz przekazanie jej zmiennej `pictureID`. Funkcja ta zostanie dodana w kolejnym ćwiczeniu.

Wywołanie funkcji `itemClicked()` wewnątrz procedury obsługującej zdarzenie `onRelease` wykorzystuje referencję do zmiennej `pictureID` przez następującą instrukcję:

```
this._parent.pictureID;
```

Aby zrozumieć jej składnię, należy przyjrzeć się ścieżce między zmienną `pictureID` i przyciskiem *list_btn*. Zmienna ta znajduje się wewnątrz duplikatu klipu filmowego, który zawiera również wymieniony przycisk. Nie istnieje ona jednak wewnątrz klipu (ponieważ nie może on przechowywać danych), ale wewnątrz jego obiektu nadrzędnego, który jest kopią instancji klipu. Tak więc aby przycisk mógł skorzystać ze zmiennej, musi powrócić o jeden poziom do góry (`_parent`). Stąd wziął się powyższy wiersz kodu.

5. Dodaj po funkcji `populateList()` następujący skrypt:

```
dropDownList_mc.menu_btn.onRelease = function() {
  populateList();
};
```

Gdy puścimy przycisk menu, wywołana zostanie funkcja `populateList()`, co spowoduje pojawienie się zbioru przycisków (przez nią tworzonych).

6. Wybierz z menu polecenie *Control\Test Movie*. Kliknij przycisk menu, aby przetestować działanie skryptu.

Gdy klikniesz przycisk menu, wywołana zostanie funkcja `populateList()`, tworząc i odpowiednio rozmieszczając kilka kopii instancji klipu filmowego *item_mc*, na których następnie zostanie umieszczony tekst. Wszystko to dzieje się niemal natychmiast.

7. Zamknij testowany film i zapisz projekt jako *pictureShow2.fla*.

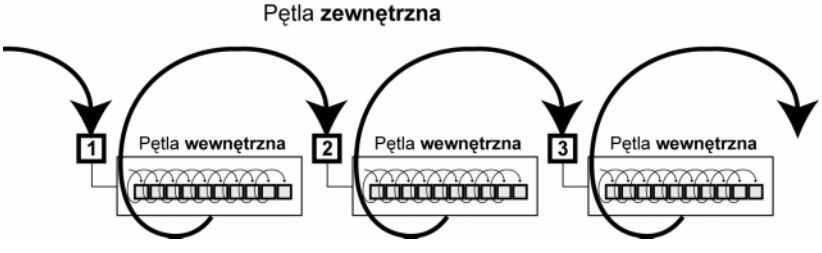
Korzystając z pętli `while`, utworzyłeś rozwijaną listę. W kolejnym ćwiczeniu, definiując reakcję na klikanie jej elementów, sprawisz, że będzie ona działać.

Pętle zagnieżdżone

Pętle są doskonałym sposobem na automatyzację pisania skryptów. Mogą one jednak robić również coś więcej niż powtarzanie zbioru akcji. Pętla zagnieżdżona, czyli umieszczona wewnątrz innej pętli, może być użyteczna przy tworzeniu sekwencji wykonującej zbiór akcji, zmieniającej nieco parametry, ponownie wykonującej zbiór akcji itd. Oto przykład pętli zagnieżdżonej:

```
var i:Number = 0;
while (++i <= 10) {
  var j:Number = 0;
  while (++j <= 10) {
    //wykonaj te akcje
  }
}
```

Akcje wewnątrz pętli zostaną wykonane 100 razy. Zewnętrzna pętla (wykorzystująca zmienną `i`) wykonywana jest 10 razy. Przy każdej iteracji następują dwa zjawiska — zmiennej `j` zostaje przyporządkowana z powrotem liczba 0, która pozwala na dziesięciokrotne wykonanie pętli wewnętrznej. Innymi słowy, w pierwszej iteracji pętli zewnętrznej dziesięciokrotnie zostaje powtórzona wewnętrzna. W drugiej — pętla wewnętrzna znów wykonywana jest 10 razy itd.



Pętle zagnieżdżone są doskonałą metodą dzielenia powtarzających się zadań na procesy ustawione w odpowiedniej hierarchii. Aby zrozumieć tę koncepcję, wyobraź sobie, że piszesz list. Przedstawia on proces oparty na zagnieżdżonej pętli, w którym zaczyna się pisać w pierwszym wierszu, stawia około 100 znaków, przechodzi do wiersza drugiego, stawia około 100 znaków itd. Jeśli list zawiera 25 wierszy, skrypt wykonujący to zadanie może wyglądać następująco:

```
var i:Number = 0;
while (++i <= 25) {
  var j:Number = 0;
  while (++j <= 100) {
    // napisz literę
  }
  // przejdź do następnego wiersza
}
```

Należy pamiętać o tym, że nie jesteśmy ograniczeni jedynie do umieszczenia jednej pętli wewnątrz innej. Jeśli projekt tego wymaga, istnieje możliwość wykorzystania dowolnej ilości poziomów zagnieżdżenia.

W poniższym ćwiczeniu użyjemy zagnieżdżonych pętli do utworzenia siatki złożonej z rysunków pojawiających się w momencie wybrania elementu rozwijanej listy.

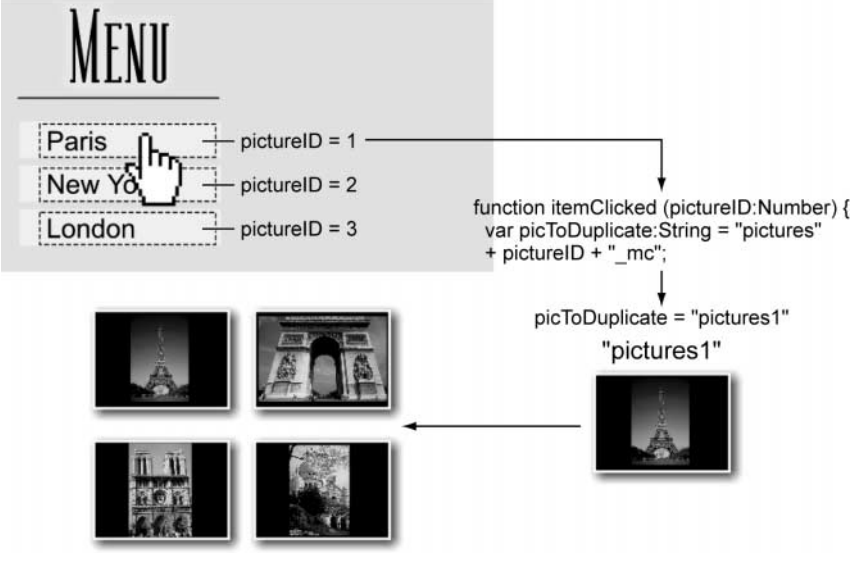
1. Otwórz plik *pictureShow2.fla*.

   Na górze sceny znajdują się trzy instancje klipów filmowych: *pictures1_mc*, *pictures2_mc* i *pictures3_mc*. W tym ćwiczeniu powielimy jeden z nich (w zależności od tego, który z elementów listy został wybrany), tworząc na scenie siatkę złożoną z rysunków.

2. Otwórz panel *Actions*, zaznacz 1. klatkę warstwy *Actions* i dodaj na końcu kodu następujący skrypt:

```
function itemClicked(pictureID:Number) {
  var picToDuplicate:String = "pictures" + pictureID + "_mc";
  var xSpacing:Number = 160;
  var ySpacing:Number = 120;
  var xStart:Number = 190;
  var yStart:Number = 30;
}
```

W poprzednim ćwiczeniu instancje przycisku *list_btn* zostały zaprogramowane w sposób pozwalający na wywoływanie powyższej funkcji i przekazywanie jej parametru (`pictureID`) w momencie ich kliknięcia. Po zakończeniu tej operacji funkcja tworzy cztery kopie jednej z instancji znajdujących się na górze sceny, kreując siatkę o rozmiarach 2×2, a następnie wysyła każdy z duplikatów do innej klatki, co powoduje wyświetlanie na nich różnych obrazów.



Pierwszy wiersz funkcji tworzy zmienną o nazwie `picToDuplicate`. Zostają jej przyporządkowane wartości *picture1_mc*, *picture2_mc* lub *picture3_mc*, oparte na wartości zmiennej `pictureID` (równej 1, 2 lub 3), która została przekazana funkcji. Te trzy ciągi znaków są nazwami instancji zawierających rysunki. Zostają one użyte w dalszej części funkcji do określenia, która z nich ma zostać wyświetlona.

Zmienna `xSpacing` określa rozmiar przestrzeni między lewymi krawędziami klipów filmowych znajdujących się w tym samym wierszu. Zmienna `ySpacing` określa rozmiar przestrzeni między klipami umieszczonymi w tej samej kolumnie. Wartości tych zmiennych są dobierane dowolnie i zależą głównie od pożądanej ilości miejsca między klipami.

Kolejne dwie zmienne, `xStart` i `yStart`, przedstawiają początkowe współrzędne pierwszej kopii klipu względem sceny. Pozycja każdego kolejnego duplikatu zostaje określona względem tego punktu.

3. Dodaj pod definicją funkcji `itemClicked()` następujący skrypt:

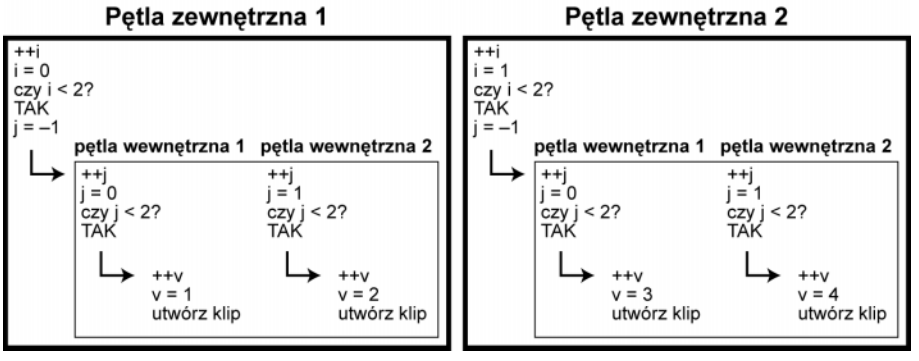
```
var v:Number = 0;
var i:Number = -1;
while (++i < 2) {
  var j:Number = -1;
  while (++j < 2) {
    ++v;
    var name:String = "pictures" + v;
    _root[picToDuplicate].duplicateMovieClip(name, v);
    _root[name]._x = xstart + i * xSpacing;
    _root[name]._y = yStart + k * ySpacing;
    _root[name].gotoAndStop(v);
  }
}
```

Powyższy skrypt zawiera pętlę zagnieżdżoną — jest nią fragment funkcji tworzący siatkę klipów filmowych o wymiarach 2×2. Pojedyncza pętla mogłaby utworzyć jedynie jedną kolumnę. Za to zagnieżdżenie wewnątrz niej drugiej pętli pozwala na umieszczanie dwóch instancji w każdej kolumnie, przesunięcie jej współrzędnych i wstawienie kolejnych dwóch (opiszemy to za chwilę). Przyjrzyjmy się, w jaki sposób funkcjonuje ten proces.

Pętla zewnętrzna, zaczynająca się w trzecim wierszu skryptu, zwiększa wartość zmiennej `i` o 1 (`++i`), ustawiając jej wartość początkową równą 0. Warunek pętli mówi: „Dopóki `i` jest mniejsze od 2, wykonaj następujące akcje”. Ponieważ `0 < 2`, wykonane zostają akcje umieszczone wewnątrz pętli. Pierwsza z nich przyporządkowuje zmiennej `j` wartość `–1`. Zostaje ona wykorzystana w pętli wewnętrznej, w której z definicji zostaje zwiększona o 1 (`++j`). Następnie pojawia się warunek: „Dopóki `j` jest mniejsze od 2, kontynuuj wykonywanie następujących akcji”. W dalszej części aż do momentu, gdy stanie się on fałszywy, skrypt wykonuje jedynie akcje umieszczone w pętli wewnętrznej. Podczas jej pierwszej iteracji wartość zmiennej `v` zostaje zwiększona o 1 (`++v`), w wyniku czego jest równa 1. Jest ona wykorzystywana w kilku kolejnych wierszach skryptu. Kopiują one i umieszczają w odpowiednich miejscach instancje klipu *pictureID*. Operacja ta powinna być już znana. Podczas drugiej iteracji zmienna `j` znów zostaje zwiększona o 1 (co następuje w instrukcji warunkowej pętli), w wyniku czego zyskuje ona wartość `1`, która jest mniejsza od 2. Powoduje to kolejne wykonanie akcji umieszczonych w pętli.

Wewnętrzna pętla nie może wykonać trzeciej iteracji, ponieważ skutkowałoby to ponownym zwiększeniem wartości zmiennej `j`, która byłaby wówczas równa `2` (warunek wyjścia z pętli wewnętrznej). W ten sposób skrypt powraca do pętli zewnętrznej. W tym miejscu zmienna `i` (wykorzystana w warunku pętli zewnętrznej) zostaje zwiększona o 1 (`++i`), osiągając wartość `1`. Jest ona mniejsza od 2, więc akcje umieszczone w pętli zostają wykonane ponownie. W wyniku tego zmiennej `j` zostaje ponownie przyporządkowana wartość `–1` i instrukcje zawarte w pętli wewnętrznej zostają wykonane jeszcze dwukrotnie.

Opisana koncepcja może być dosyć skomplikowana. Przyjrzyj się jej dokładnie i upewnij się, że wszystko zrozumiałeś.



Aby uzyskać efekt siatki o rozmiarach 2×2, złożonej z klipów filmowych umieszczonych w odpowiednich odstępach, należy użyć skryptu:

```
_root[name]._x = xStart + i * xSpacing;
_root[name]._y = yStart + j * ySpacing;
```

Pierwszy wiersz wykorzystuje aktualną wartość zmiennej `i` do ustawienia odstępów między klipami w poziomie, zaś drugi — zmiennej `j` do ustawienia ich w pionie w momencie powielenia klipu. Dowiedziałeś się już, że w każdej iteracji pętli zewnętrznej dwukrotnie zostaje wykonana wewnętrzna. Gdy zmienna `i` posiada wartość `0`, zmiennej `j` podczas wykonywania wewnętrznej pętli zostają przyporządkowane kolejno liczby 0 i 1. Następnie wartość zmiennej `i` jest zwiększana o 1, zaś `j` znów nadawane są wartości `0` i `1`. Ponieważ znamy wartości zmiennych `xStart`, `xSpacing`, `yStart` i `ySpacing` oraz sposób zwiększania wartości zmiennych `i` oraz `j` w pętli, możemy określić położenie każdego klipu.

Powielenie pierwszej instancji:

```
_x = 190 + 0 * 160; // współrzędnej x zostaje przyporządkowana wartość 190
_y = 30 + 0 * 120;  // współrzędnej y zostaje przyporządkowana wartość 30
```

Uwaga

Należy pamiętać o tym, że w wyrażeniach matematycznych mnożenie jest zawsze wykonywane przed dodawaniem.

Powielenie drugiej instancji:

```
_x = 190 + 0 * 160; // współrzędnej x zostaje przyporządkowana wartość 190
_y = 30 + 1 * 120;  // współrzędnej y zostaje przyporządkowana wartość 150
```

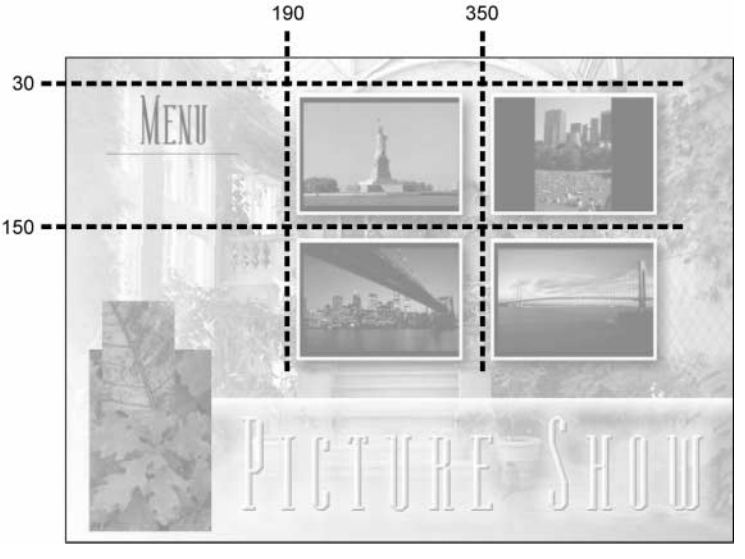
Powielenie trzeciej instancji:

```
_x = 190 + 1 * 160; // współrzędnej x zostaje przyporządkowana wartość 350
_y = 30 + 0 * 120;  // współrzędnej y zostaje przyporządkowana wartość 30
```

Powielenie czwartej instancji:

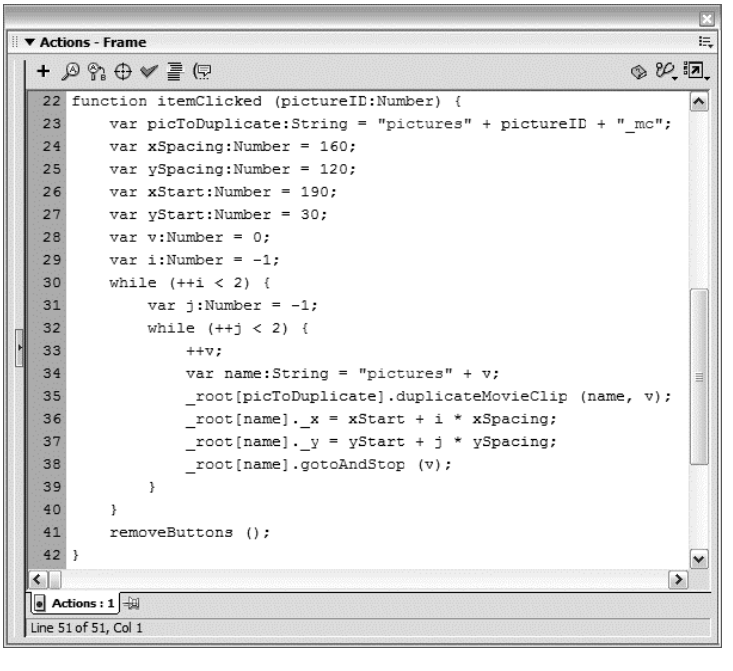
```
_x = 190 + 1 * 160; // współrzędnej x zostaje przyporządkowana wartość 350
_y = 30 + 1 * 120;  // współrzędnej y zostaje przyporządkowana wartość 150
```

(patrz pierwszy rysunek na następnej stronie)



4. W ostatnim wierszu definicji funkcji `itemClicked()` dodaj następującą akcję:

```
removeButtons();
```



Powyższa akcja wywołuje funkcję o nazwie `removeButtons()`, która powoduje usunięcie przycisków z rozwijanej listy po kliknięciu przycisku i utworzeniu siatki. Za chwilę ją napiszemy.

5. Dodaj na końcu kodu następujący skrypt:

```
function removeButtons() {
  var numberOfButtons:Number = buttonNames.length;
  var i:Number = -1;
  while (++i<numberOfButtons) {
    var name:String = "item" + i;
    dropDownList_mc[name].removeMovieClip();
  }
}
```

Ta funkcja używa prostej pętli `while` do kolejnego usuwania wszystkich przycisków (które tak naprawdę są duplikatami instancji klipów filmowych *item0*, *item1* i *item2*) tworzących rozwijaną listę. Działa ona podobnie jak funkcja, której używaliśmy do ich tworzenia. Przyjrzyjmy się jej bliżej.

Wartość zmiennej `numberOfButtons`, która została również użyta w funkcji `populateList()`, pochodzi od właściwości `length` tablicy `buttonNames`. Ponieważ zawiera ona trzy elementy, pętla wykonuje trzy iteracje. W każdej z nich zmiennej `name` przyporządkowuje się aktualną wartość zmiennej `i`. Następnie zostaje użyta metoda `removeMovieClip()`, powodująca usunięcie duplikatów instancji klipów filmowych, czyli pozycji listy wyświetlanej pod przyciskiem *Menu*. W referencji do nich wykorzystana zostaje zmienna `name`.

6. Wybierz z menu polecenie *Control\Test Movie*, aby przetestować projekt.

Kliknij przycisk *Menu*, by wyświetlić listę. Klikanie jej elementów powoduje tworzenie siatki złożonej z obrazków w oparciu o wartość przekazywanej do funkcji `itemClicked()` zmiennej `pictureID`. Zauważ też, że funkcja `removeButtons()` usuwa listę.

7. Zamknij testowany film i zapisz projekt jako *pictureShow3.fla*.

W tym ćwiczeniu użyłeś zagnieżdżonych pętli do utworzenia na ekranie siatki złożonej z obrazków. Mogłeś w tym celu przeciągać ich instancje z biblioteki i umieszczać w odpowiednich miejscach za pomocą myszy. Jednak użycie pętli pozwoliło na zautomatyzowanie tego procesu. Dzięki temu po niewielkich modyfikacjach skrypt, który napisałeś, może zbudować siatkę dowolnej wielkości — nawet tak dużą jak 100×100. Wykorzystywanie zmiennych w ten sposób nie tylko pomaga w automatyzacji procesu, który normalnie jest wykonywany ręcznie. Pozwala też na skalowanie projektu w zależności od warunków zaistniałych w momencie jego uruchomienia.

Wyjątki w pętlach

Ogólnie rzecz biorąc, pętle wykonują iteracje aż do momentu, gdy warunek, którego używają, nie jest już prawdziwy. Aby zmienić to zachowanie, można skorzystać z dwóch akcji — `continue` i `break`.

Instrukcja `continue` pozwala na zatrzymanie aktualnej iteracji (tzn., że po jej uruchomieniu nie zostają już wykonane żadne akcje w obrębie iteracji) i przejście do następnej, np.:

```
var total:Number = 0;
var i:Number = 0;
while (++i <= 20) {
  if (i == 10) {
    continue;
  }
  total += i;
}
```

Instrukcja `while` w powyższym skrypcie tworzy pętlę, w której wartość zmiennej `i` jest kolejno zwiększana od 1 do 20. W każdej iteracji zostaje ona dodana do zmiennej `total` — aż do momentu, gdy zmienna `i` osiągnie wartość `10`. W tym momencie wykonuje się akcja `continue` oznaczająca, że w tej iteracji nie zostanie uruchomiona już żadna instrukcja i pętla przeskakuje do kolejnej. To powoduje wygenerowanie następującego zbioru liczb:

```
1 2 3 4 5 6 7 8 9 11 12 13 14 15 16 17 18 19 20
```

Zauważ, że nieobecna jest liczba 10, co oznacza, że w czasie dziesiątego powtórzenia pętli nie została wykonana żadna akcja.

Akcja `break` jest używana do opuszczania pętli, jeśli nawet jej warunek jest ciągle prawdziwy, np.:

```
var total:Nunber = 0;
var i:Number = 0;
while (++i <= 20) {
  total += i;
  if (total >= 10) {
    break;
  }
}
```

Powyższy skrypt w każdej iteracji zwiększa wartość zmiennej `total` o 1. Gdy osiągnie ona liczbę 10 lub większą, wykonywana jest akcja `break` i pętla `while` zostaje zatrzymana, mimo że powinna ona wykonać 20 powtórzeń.

W poniższym ćwiczeniu wykorzystamy instrukcje `continue` i `break` do budowy prostego mechanizmu wyszukującego.

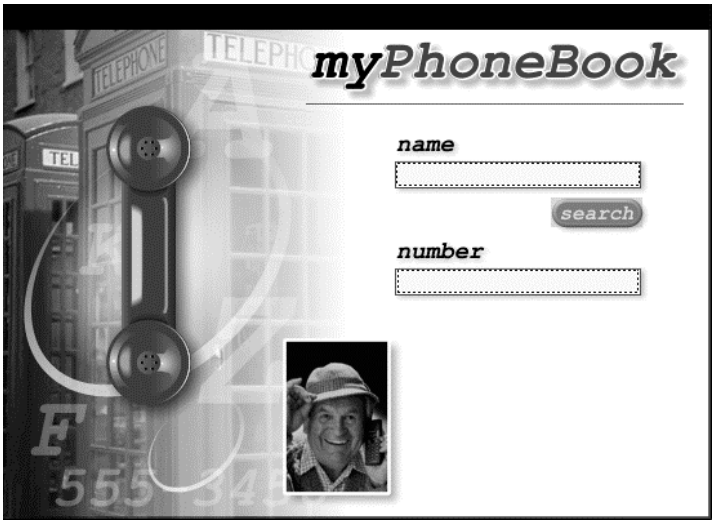
1. Otwórz plik *phoneNumberSearch1.fla*, znajdujący się w folderze *Lekcja09\Assets*.

   Ten plik zawiera dwie warstwy: *Actions* i *Search Assets* (zasoby wyszukiwarki). Na pierwszej z nich jest umieszczona procedura wyszukująca, zaś na drugiej — pola tekstowe, przycisk i grafika potrzebne do utworzenia tego projektu (patrz pierwszy rysunek na następnej stronie).

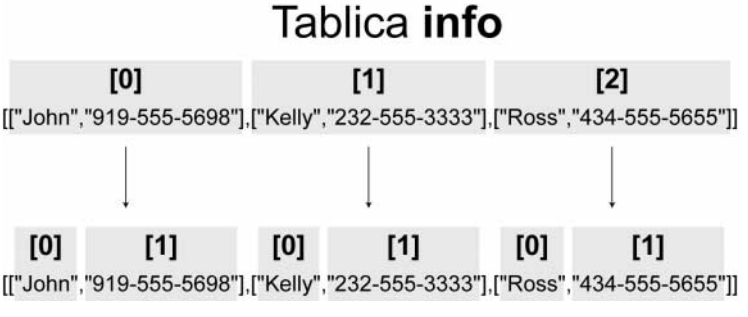
   W tym ćwiczeniu opracujemy prostą aplikację pozwalającą na odnajdywanie numeru telefonu osoby, której imię zostaje wpisane przez użytkownika. Na ekranie znajdują się dwa pola tekstowe: *name_txt* — używane do wpisywania imienia oraz *result_txt* — wyświetlające wyniki wyszukiwania. Nad obrazkiem przycisku *Search* znajduje się niewidoczna instancja przycisku o nazwie *search_btn*. Jest ona używana do wywoływania funkcji wyszukującej.

2. Otwórz panel *Actions*, zaznacz 1. klatkę warstwy *Actions* i umieść w niej następujący skrypt:

   ```
   var info:Aray = [["John", "919-555-5698"], ["Kelly", "232-555-3333"],
   ["Ross", "434-555-5655"]];
   ```



Ten skrypt tworzy dwuwymiarową tablicę o nazwie `info`, zawierającą trzy elementy, z których każdy znajduje się na osobnej podtablicy. Pierwszym elementem każdej podtablicy jest imię, zaś drugim — numer telefonu.



Aby dostać się do pierwszego imienia zawartego w tablicy, należy użyć sformułowania `info[0][0]`, zaś do pierwszego numeru telefonu — `info[0][1]`, co odpowiada imieniu i numerowi telefonu Johna. Ta składnia w skrypcie, który piszemy, odgrywa dość istotną rolę.

3. Dodaj pod tablicą `info` następującą definicję funkcji:

```
function search () {
  var matchFound:Boolean = false;
  var i:Number = -1;
  while (++i < info.length) {
  }
}
```

Zacząłeś definiować funkcję, której zadaniem jest wyszukiwanie określonego numeru telefonu w tablicy `info`. Jej pierwsza akcja tworzy zmienną o nazwie `matchFound` i przyporządkowuje jej wartość początkową `false` (wkrótce zobaczysz, w jaki sposób zostaje ona wykorzystana).

Pętla `while` zostaje wykonana raz dla każdego elementu tablicy.

4. Umieść w pętli `while`, zawartej w funkcji `search()`, następujący skrypt:

```
if (info[i][0].toLowerCase() != name_txt.text.toLowerCase()) {
  continue;
}
result_txt.text = info[i][1];
matchFound = true;
break;
```

W każdej iteracji pętli instrukcja `if` sprawdza, czy nastąpiła rozbieżność między imieniem wpisanym przez użytkownika (w którym wszystkie litery są zamieniane na małe) w polu tekstowym *name_txt* i tym, które znajduje się w tablicy (którego litery również są zamieniane na małe). W tym celu wykorzystana zostaje zmienna `i`.

Gdy nastąpi brak zgodności między imionami, wykonana zostaje akcja `continue` i skrypt przechodzi do kolejnej pętli. Dzięki wykorzystaniu metody `toLowerCase()` do zmiany wszystkich liter na małe procedura wyszukiwania nie rozróżnia małych i wielkich liter.

Jeśli imię znajdujące się w tablicy pasuje do zawartości pola tekstowego *name_txt*, akcja `continue` nie zostaje wykonana i instrukcje umieszczone po komendzie `if` zostają uruchomione. Pierwsza z nich wykorzystuje wartość zmiennej `i` do przyporządkowania właściwości *result_txt.text* odpowiedniego numeru telefonu. Następnie zmiennej `matchFound` zostaje nadana wartość `true` i wykonana akcja `break`, powodująca zatrzymanie pętli.

Aby lepiej zrozumieć, jak działa powyższy skrypt, wyobraźmy sobie, że ktoś wpisał w polu tekstowym *name_txt* imię Kelly. Jego pozycja w tablicy jest następująca:

```
info[1][0]
```

W pierwszej iteracji pętli zmienna `i` posiada wartość `0`, co oznacza, że zawarta w pętli instrukcja `if` wygląda następująco:

```
if (info[0][0].toLowerCase() != name_txt.text.toLowerCase()) {
  continue;
}
```

Powyższy kod sprawdza, czy ciąg znaków john (imię umieszczone jako element `info[0][0]` po zmianie wszystkich liter na małe) jest równy słowu kelly (wpisanemu w polu tekstowym *name_txt* po zmianie wszystkich liter na małe). Ponieważ warunek ten jest prawdziwy, wykonana zostaje akcja `continue` i następuje przejście do kolejnej pętli. W każdej iteracji wartość zmiennej `i` zostaje zwiększona o 1, więc następna instrukcja `if` wygląda następująco:

```
if (info[1][0].toLowerCase() != name_txt.text.toLowerCase()) {
  continue;
}
```

Sprawdza ona, czy ciąg znaków `kelly` (imię umieszczone jako element `info[1][0]` po zmianie wszystkich liter na małe) jest równy słowu `kelly` (wpisanemu w polu tekstowym *name_txt* po zmianie wszystkich liter na małe). Ponieważ w tym przypadku występuje równość, akcja `continue` zostaje pominięta i wykonywane są kolejne trzy instrukcje. Pierwsza z nich przyporządkowuje właściwości *result_txt.text* element `info[i][1]`. Zmienna `i` posiada wartość `1`, więc elementem tym jest `info[1][1]` — numer telefonu Kelly. Zostaje on wyświetlony. Następnie zmiennej `matchFound` zostaje przyporządkowana wartość `true` i akcja `break` powoduje wyjście z pętli.

Uwaga

Akcja `break` *nie jest w tym przypadku konieczna, jednak pozwala na skrócenie czasu wyszukiwania. Pomyśl, ile czasu można zaoszczędzić, stosując ją, aby zapobiec wykonywaniu niepotrzebnych iteracji pętli, gdy tablica posiada 10 000 elementów.*

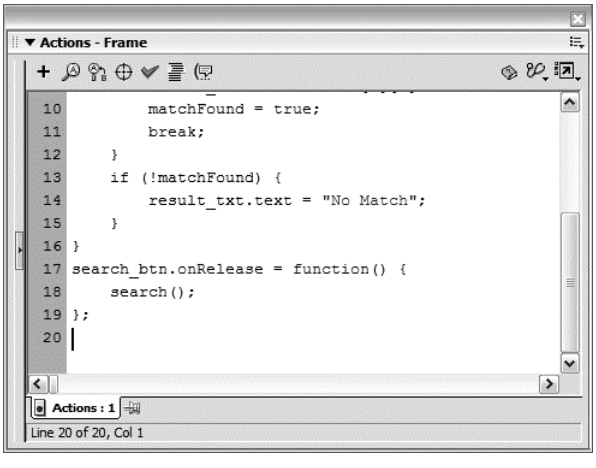
5. Umieść jako ostatnią akcję funkcji `search()` następującą instrukcję `if`:

```
if (!matchFound) {
  result_txt.text = "No Match";
}
```

Ta instrukcja, niebędąca częścią pętli, sprawdza, czy zmienna `matchFound` po zakończeniu pętli nadal posiada wartość `false` (która została jej przyporządkowana na samym początku). Jeśli tak, akcja w instrukcji `if` przyporządkowuje wartości *result_txt.text* ciąg znaków `"No Match"`. Wyrażenie `!matchFound` jest skróconą wersją komendy `matchFound == false`.

6. Dodaj na końcu kodu po funkcji `search()` następujący skrypt:

```
search_btn.onRelease = function() {
  search();
};
```



Powyższy skrypt wiąże procedurę obsługującą zdarzenie `onRelease` z przyciskiem *search_btn*. Gdy ten zostanie przyciśnięty i puszczony, nastąpi wywołanie funkcji `search()`.

7. Wybierz z menu polecenie *Control\Test Movie*. Wpisz w polu *name* imię `John`, `Kelly` lub `Ross` i kliknij przycisk *Search*. Następnie wpisz dowolne inne imię i kliknij ponownie.

Gdy imię znajduje się w tablicy `info`, powinien pojawić się numer telefonu. Jeśli jest ono nieprawidłowe, wyświetlony zostaje tekst `"No Match"`.

8. Zamknij testowany film i zapisz projekt jako *phoneNumberSearch2.fla*.

W tym ćwiczeniu użyłeś wyjątków `continue` i `break` do napisania procedury wyszukującej. W praktyce instrukcja `break` jest używana znacznie częściej niż `continue`. Jest tak dlatego, że programiści często wolą używać instrukcji warunkowej `if` do pominięcia akcji w pętli.

Podsumowanie

W czasie tej lekcji:

- Odkryłeś, jak użyteczne są pętle,
- Poznałeś trzy typy pętli, dostępne w języku ActionScript,
- Użyłeś pętli `while` na kilka różnych sposobów,
- Utworzyłeś i wykorzystałeś pętlę zagnieżdżoną,
- Użyłeś wyjątków `continue` i `break`.