Flash 5 Action Script.
Techniki zaawansowane

Autor: praca zbiorowa
Tłumaczenie: Łukasz Oberlan, Wojciech Pazdur, Remigiusz Wilk
ISBN: 83-7197-567-8
Tytuł oryginału: Flash 5 ActionScript Studio
Format: B5, stron: 612
Zawiera CD-ROM

Język skryptowy to nie jest zbyt ładna nazwa, ale można za jej pomocą opisać bardzo wiele, JEŻELI (i jest to naprawdę wielkie „jeżeli") wie się, jak jej używać. W firmie Macromedia szybko zdano sobie sprawę z tego, że ładna nazwa bywa przydatna, dlatego nadano językowi sympatyczne miano „ActionScript". Brzmi to znacznie bardziej atrakcyjnie, ale ani trochę nie pomaga w zrozumieniu „jak go używać?". I dlatego pojawiła się ta książka.

Książka podzielona jest na trzy główne części. W pierwszej z nich zajmiemy się opanowaniem podstawowych umiejętności, w drugiej przejdziemy do zadań specjalistycznych. Trzecia część to studium tematu. Naszym pierwszym zadaniem będzie więc nauka poprawnego posługiwania się językiem ActionScript, drugim – pogłębienie tej wiedzy, a na koniec przejdziemy do przykładów ilustrujących sposób, w jaki to wszystko rzeczywiście działa. Autorzy to doświadczeni i aktywni projektanci witryn, programiści albo ludzie łączący obie te funkcje, którzy zgłębili tajemnice Flasha. Teraz postarają się odkryć przed tobą arkana programowania we wspaniałym języku, jakim jest ActionScript.

Część 1. Podstawowe umiejętności
W tej części książki przybliżymy podstawową wiedzę, którą powinieneś opanować, zanim przystąpisz do tworzenia kodu, który będzie cechował się przejrzystością i funkcjonalnością. Nauczymy cię, jak używać nowych konstrukcji, poleceń i pokażemy, ile zasad programowania obiektowego zawiera język ActionScript i jakie to może mieć dla ciebie znaczenie.

Część 2. Specjalizacja
Przejdziemy do szczegółowego omówienia niektórych aspektów podstawowych umiejętności, pokazując na przykładach, w jaki sposób mogą być zastosowane w twoich projektach. Omówione zostaną między innymi nowe obiekty typu Sound i XML, tworzenie prawdziwego engine'u 3D na podstawie tablic, użycie Generatora do uaktualnienia filmów Flasha. Dowiesz się także, jak praca zespołowa może bardzo rozwinąć inwencję twórczą.

Część 3. Studium tematu

To już koniec okresu ochronnego; nie licz więcej na ulgowe traktowanie. W tej części książki najlepsi projektanci odsłonią niektóre ze swoich sekretów, aby pokazać, jak tworzą niesamowite i niepowtarzalne witryny. Znajdziesz tu dwa zaawansowane przykłady, dzięki którym dowiesz się, jak opracować od początku do końca jedyny w swoim rodzaju interfejs oraz przekonasz się, że nowy dodatek do Flasha – XML – może nie tylko zrewolucjonizować wymianę danych, ale także wpłynąć na wygląd witryny. Po tym wszystkim nie będziesz miał problemów z Flashem.

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

Spis treści

Rozdział 8.
Kontrola dźwięku

We wszystkich niemal przypadkach dźwięk jest nieodłączną częścią takich mediów jak telewizja czy kino. Gdy zastanawiamy się, jak zwiększyć potencjał tworzonej przez nas witryny, możemy sobie zadać pytanie o to, co różni Internet od innych mediów. Wprowadzenie dźwięku spotęguje wrażenie, jakie nasz projekt wywoła na użytkownikach. W przeszłości nie było to bardzo popularne, ponieważ nakładało zbyt duże wymagania na ograniczoną przepustowość łączy, ale wraz z szerokim rozpowszechnieniem się różnorakich metod kompresji (na przykład MP3), dźwięk pojawiający się w witrynach stał się bardzo atrakcyjnym dodatkiem.

W tym rozdziale zastanowimy się nad tym, jak można radzić sobie z dźwiękiem za pomocą dołączonych do języka ActionScript Flasha 5 metod obiektów typu `Sound`. Omówimy również kilka przyczyn, które powodują niepoprawne działanie tych instrukcji — i rzecz jasna, zaproponujemy od razu środki zaradcze. Postaramy się, aby wprowadzenie do zagadnień związanych z dźwiękiem było jak najprostsze i przyjazne dla czytelnika.

W tym rozdziale przyglądniemy się tylko dźwiękom sterowanym przez zdarzenia. Flash umożliwia również strumieniowe przesyłanie dźwięku, ale doświadczenie podpowiada, że stosowanie tego w bardzo rozbudowanej witrynie jest bardzo narażone na błędy. Jest to związane z tym, że Flash inicjując strumieniowe przesyłanie dźwięku skupia się tylko na nim (zamiast na listwie czasowej). Jeśli z jakiegoś powodu nie może ona nadążyć za przesyłaniem strumieniowym, Flash pomija kolejne ujęcia, aby dotrzymać kroku dźwiękowi. Można to przyjąć, gdy korzystamy z listwy czasowej bazującej na animacji automatycznej, ale nikt, kto będzie wykorzystywał skrypty nie może sobie na to pozwolić. Jest bowiem bardzo prawdopodobne, że ujęcia, które mogą zostać pominięte zawierają ważne skrypty inicjujące lub istotne algorytmy. W dodatku rozbudowane witryny opierające się na skryptach często zawierają wiele małych pętli w poszczególnych listwach czasowych. To wszystko powoduje, że nie można pogodzić skomplikowanych witryn z dźwiękiem pobieranym strumieniowo.

Jak działa dźwięk we Flashu

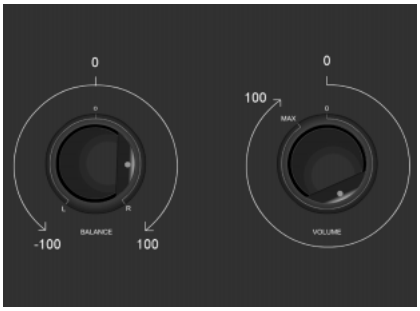
Z pozoru bardzo niepozorny obiekt typu `Sound` znajdujący się we Flashu 5 jest w rzeczywistości bardzo elastyczny, a ponadto umieszczono w nim kilka opcji umożliwiających jego definiowanie i stosowanie. Zanim rozłożymy je na czynniki pierwsze, musimy na początku zrozumieć kilka podstawowych cech używania dźwięku we Flashu, a jest to rzecz, której przyjrzymy się bardzo uważnie w tej części rozdziału.

W pewnym stopniu techniki zastosowane w manipulowaniu dźwiękiem w języku ActionScript przypominają potencjometry balansu i poziomu głośności dźwięku spotykane w typowym sprzęcie hi-fi. Niektóre z przykładów prawdopodobnie będą dla ciebie tylko przypomnieniem, ale przyjrzyj się plikowi *sound.fla*, aby przekonać się, jak w rzeczywistości działają potencjometry. Po uruchomieniu filmu zobaczymy mały obszar zawierający parę pokręteł.

Nie przejmuj się tym, że nie wiesz, jak to działa w języku ActionScript, bo dokładniej przeanalizujemy to nieco później. W tej chwili możemy się nieco pobawić. Na samym początku włącz urządzenie klikając odpowiedni przycisk. Gdy będziesz poruszał prawym pokrętłem (po prostu kliknij i przytrzymaj kursor na potencjometrze w okolicy wskaźnika w postaci kropki, a następnie przesuwaj go w dowolnym kierunku), zobaczysz, że na ekranie pojawiają się dwa głośniki reprezentujące poziom głośności. Gdy będziesz ją zwiększał lub zmniejszał głośniki będą zachowywały się analogicznie.

Teraz za pomocą potencjometru reguluj balans dźwięku w panoramie stereo. Warto zauważyć, że gdy tylko zmienisz położenie, powiedzmy, na prawo od środka, nie wpłynie ono w tym momencie na głośnik po prawej stronie, a raczej zmniejszy poziom głośności głośnika znajdującego się po lewej stronie. Jeśli nie zwrócilibyśmy w tej chwili na to uwagi, to ta dosyć subtelna kwestia mogłaby później spowodować wiele różnych problemów związanych z pojęciami które będą istotne, gdy dojdziemy do określania wartości liczbowych dla obiektów kontrolujących, czyli potencjometrów.

Zwróć również uwagę na współdziałanie poziomu głośności i balansu dźwięku. Pokrętło poziomu głośności ustawia maksymalną głośność obu głośników, natomiast pokrętło balansu zmienia ich głośność w zakresie od zera do wartości ustalonej przez potencjometr poziomu głośności. Zamieniając je na wartości, które będziemy stosować we Flashu uzyskamy wynik, który można zobaczyć na kolejnym rysunku.



Kontrolowanie głośności dźwięku jest proste — zero to jej brak (albo 0%), a jej maksymalna wartość równa jest 100. Z balansem dźwięku sprawa ma się nieco inaczej — w środku, to znaczy wtedy, gdy poziom dźwięku jest w równowadze, ma wartość zero. Gdy jest ustawiony na maksimum z lewej strony, to znaczy wtedy, gdy dźwięk płynie tylko z lewego głośnika przyjmuje wartość minus 100. Gdy natomiast ustawiony jest na maksimum z prawej strony, oznacza to, że dźwięk płynie tylko z prawego głośnika i ma wartość 100.

Na powyższym rysunku przedstawiona została sytuacja w której poziom głośności jest ustawiony na około 50%, przy czym w prawym głośniku usłyszymy dźwięk z pełną mocą, a lewy wykorzystuje tylko około 50% swojej mocy. Zamieniając to na wartości używane przez obiekty typu `Sound` w języku ActionScript Flasha 5 można by powiedzieć, że oba potencjometry są ustawione w pobliżu 50. Jest to wszystko, co powinieneś wiedzieć w tej chwili. Tymczasem zajmiemy się czymś innym.

Tworzenie obiektu typu Sound

Abyś mógł spokojnie zapoznać się z zagadnieniem, w tej części nie będę korzystał ze wszystkich zasad dotyczących definiowania obiektów typu `Sound` ani przedstawiał ich prawdziwego znaczenia. Biorąc pod uwagę język ActionScript dysponujemy trzema możliwościami tworzenia dźwięku we Flashu. Można to zrobić:

- za pomocą techniki znanej z Flasha 4, czyli dołączania dźwięków do listw czasowych i kontrolowania ich poprzez czasowe obwiednie dźwiękowe;
- za pomocą techniki wprowadzonej we Flashu 5, czyli dołączania dźwięków do obiektów typu `Sound`;
- za pomocą połączenia obu powyższych technik.

Biorąc pod uwagę potencjalnych czytelników, nie będziemy dogłębnie zajmować się pierwszą z tych możliwości, ponieważ zakładamy, że co nieco już o niej wiedzą. Z drugiej jednak strony pozostałe rozwiązania warte są szczegółowej analizy i ich zastosowanie zostanie przedstawione na wielu przykładach w następnych rozdziałach.

Aby otrzymać nadający się do użytku obiekt typu `Sound`, musimy zrobić dwie rzeczy: stworzyć ten obiekt i dołączyć do niego określony dźwięk. W tej drugiej kwestii możemy posłużyć się dwoma sposobami — albo odnieść się do rzeczywistego pliku z dźwiękiem, albo do listwy czasowej, która go zawiera. Pierwsza możliwość jest prostsza, ale

druga pozwala na kontrolowanie wielu dźwięków za pomocą tego samego obiektu typu Sound (i co powinniśmy zobaczyć później — umożliwia zastosowanie „starych" cech dźwiękowych z Flasha 4 razem z obiektami typu Sound nowej wersji programu). W niej również kryje się kilka subtelnych błędów, o czym musisz wiedzieć. Będziemy się nimi zajmowali, gdy się pojawią.

W kilku następnych ćwiczeniach wstępnie można zastosować plik *specialFX.fla*. Znajduje się w nim kilka dźwięków już zaimportowanych do biblioteki, ale możesz równie dobrze zmodyfikować je, a nawet zastosować własne, jeśli masz taką ochotę. Odgłosy, które odnajdziesz w tym pliku zostały stworzone dla gier we Flashu, przypominających *Pac Mana* czy *Mario Bros*. Są one pełne takich miłych dla ucha dźwięków, które stosowano w konsolach do gier z wczesnych lat osiemdziesiątych.

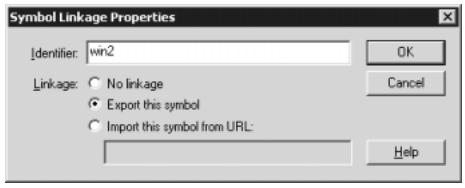
Dołączanie dźwięku z wykorzystaniem współdzielenia

Połączenie obiektu typu Sound z Flasha 5 z plikiem dźwiękowym z biblioteki najlepiej zastosować wtedy, gdy chcesz dołączyć pojedynczy dźwięk do każdego z obiektów w czasie odtwarzania obiektu, o którym mowa. Możesz zastosować tę technikę do tworzenia obiektów typu Sound, które będą odpowiedzialne za dźwięki w grach czy animacjach.

Zanim dołączysz, powiedzmy, plik *Winner_02* do obiektu, musisz wcześniej nadać mu **nazwę współdzielenia**. Nie różni się to w dużym stopniu od nadawania klipom filmowym nazw klonów. Pamiętaj jednak o tym, że do nazwy współdzielenia, a nie nazwy znanej z biblioteki, odnosi się dźwięk w kodzie skryptu. Aby wpisać nazwę współdzielenia możesz:

- wyróżnić dźwięk w bibliotece, a następnie wybrać z wysuwanego menu *Options* opcję *Linkage...*;
- kliknąć prawym przyciskiem myszy dźwięk i wybrać *Linkage...* z menu, które się pojawi.

Niezależnie od tego, jak tego dokonasz, zobaczysz okno dialogowe *Symbol Linkage Properties* (właściwości współdzielenia symboli). Aby nadać dźwiękowi nazwę współdzielenia musisz wybrać opcję *Export this symbol* i wpisać nazwę w polu *Identifier*, które w tej chwili jest aktywne. Mojemu dźwiękowi nadałem nazwę win2.



Warto dodać w tym miejscu pewne ostrzeżenie. Flash nie sprawdza tego, czy dana nazwa jest niepowtarzalna, a zatem warto je gdzieś zapisywać, albo zastosować określone konwencje nazewnictwa. W przeciwnym wypadku możesz zetknąć się z pewnymi niepożądanymi efektami.

Gdy będziemy mieli przygotowany dźwięk, możemy dołączyć go do obiektu typu `Sound`, który został stworzony za pomocą języka ActionScript korzystając z następującego kodu:

```
mySound = new Sound (target);
```

W tym kodzie:

- *mySound* jest nazwą obiektu typu `Sound`. To znaczy, że podlega wszystkim zasadom dotyczącym zmiennych w języku ActionScript. Oznacza to z kolei, że możliwy jest dostęp do niej z dowolnego miejsca poza listwą czasową, na której została zdefiniowana, a także to, że możesz używać ścieżek adresowych w notacji kropkowej;
- *target* jest ścieżką adresową odnoszącą się do listwy czasowej, na której zostanie stworzony obiekt typu `Sound`. Zazwyczaj jego parametry nie są wyszczególnione (w takim przypadku Flash będzie go tworzył na aktualnej listwie czasowej). Czasami warto jest stworzyć wszystkie obiekty dźwiękowe w tym samym miejscu (takim jak główna listwa czasowa czy określony klip filmowy) po to, aby wiedzieć, gdzie wszystkie się znajdują i w jakim punkcie zostały zdefiniowane.

Zastosowanie dźwięku pochodzącego z biblioteki

Rozpoczynając od pliku *specialFX.fla* i ujęcia pierwszego głównej listwy czasowej dodaj następującą linię kodu:

```
winSound = new Sound();
```

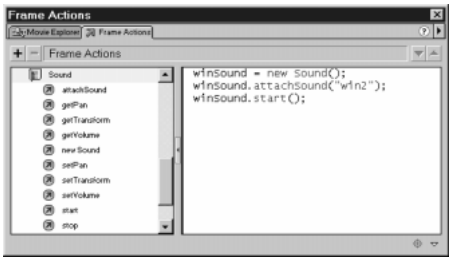
Jak sugerowałem powyżej, jeśli chcesz stworzyć obiekt typu `Sound` w innym miejscu niż listwa czasowa, na której aktualnie się znajdujesz, powinieneś dodać ścieżkę adresową w nawiasach. Gdy chcesz zastosować polecenie `new Sound`, upewnij się wcześniej, czy dana ścieżka istnieje, bo w przeciwnym wypadku obiekt w ogóle nie będzie stworzony. Z tego powodu zwykle lepiej jest tworzyć je z listwy czasowej, do której chcesz je dodać.

Następnie wpisz następujące polecenie, aby dołączyć do obiektu `WinSound` plik dźwiękowy `win2`:

```
winSound.attachSound ("win2");
```

Cudzysłów oznacza, że nazwa współdzielenia jest ciągiem znaków, a nie zmienną czy obiektem (jeśli nie użyjesz cudzysłowu, Flash „będzie sądził", że odnosisz się do ścieżki adresowej, jak to zobaczymy nieco dalej). Gdy już to zrobisz, odtwarzanie dźwięku wymagać będzie bardzo prostej instrukcji:

```
winSound.start();
```



Jeśli przetestujesz teraz film, usłyszysz jednokrotne odtworzenie się dźwięku *Winner_02*. Jeśli chcesz go zatrzymać zanim dojdzie do swojego końca, możesz kilka ujęć dalej na listwie czasowej dodać następującą instrukcję:

```
winSound.stop();
```

Metoda `stop` przerwie nagle odtwarzanie dźwięku w danym ujęciu. Po sprawdzeniu działania usuń dołączone przed chwilą ujęcie razem z dodatkowymi instrukcjami, aby wrócić z powrotem do stanu sprzed kilku chwil.

To może zabrzmieć nieprawdopodobnie, ale w rzeczywistości istnieje kilka subtelnych aspektów zachowania się instrukcji `stop()`. Jednak nie będą one oczywiste, dopóki nie zaczniemy kontrolować wielu obiektów typu `Sound`, a zatem wrócimy do tej metody ponownie.

Metoda Sound.start

Wróćmy jednak do metody `start`, którą zastosowaliśmy w poprzednim przykładzie. Ma ona kilka parametrów, które możemy wymienić, aby uzyskać lepszą kontrolę nad odtwarzanym dźwiękiem:

```
mySound.start(offset, loops);
```

W tym przypadku *offset* jest liczbą odtwarzanych sekund dźwięku, a *loops* jest liczbą powtórzeń. Aby dowiedzieć się, jak długi jest dany dźwięk, zaznacz go, a następnie kliknij w oknie biblioteki pozycję menu *Options/Properities*. Jeśli na przykład *Winner_02* trwa 3,8 sekundy, a chcielibyśmy zacząć w połowie pliku jego odtwarzanie i powtórzyć to dwa razy, należy zastąpić poprzednią instrukcję następującą:

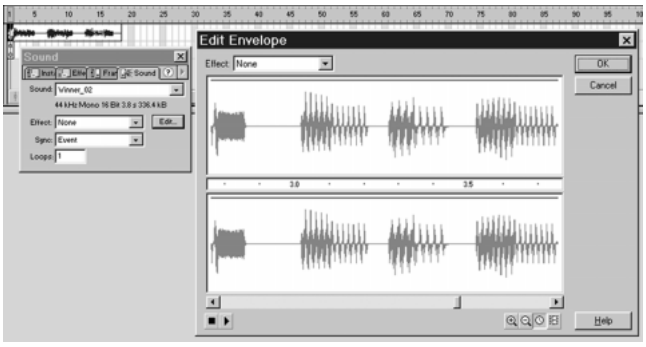
```
winSound.start(1.9, 2);
```

Flash stosuje *offset* w każdej pętli, a zatem dźwięk będzie za każdym razem rozpoczynał się od środka. W przypadku *loops*, najwyższą wartością, jaką możemy zastosować jest 214748. Gdy zostanie ona wpisana, dźwięk będzie powtarzał się niemal w nieskończoność.

Dla większości praktycznych zastosowań wystarcza zwykle ustawienie wartości zapętlenia w okolicach 1000. Ale najwyższą wartość podałem na wypadek, gdyby kiedyś zainteresowała cię granica. Teraz już ją znasz.

Może się zdarzyć, że dźwięk, którym dysponujesz, ma niepożądane opóźnienie na początku albo coś w rodzaju „białego szumu", który chcesz pominąć. Jeśli potrafisz rozróżnić punkt, od którego chcesz zacząć przyglądać się kształtom fali dźwiękowej, dobrym rozwiązaniem może być tymczasowe dołączenie dźwięku do ujęcia, a następnie przejście do okna *Edit Envelope* za pomocą polecenia *Edit* w panelu *Sound*. Zaznaczając ikony znajdujące się na dole po prawej stronie okna możesz osiągnąć kilka celów.

- Możesz dowiedzieć się, po jakim czasie (w sekundach) zaczyna się określona cecha kształtu fali i zastosować ją jako zmienną `offset`. W przedstawionej na rysunku fali można zaobserwować przerwę w dźwięku, która kończy się dokładnie po trzech sekundach. Powiększając za pomocą ikony szkła powiększającego kształt fali można dokładnie zaznaczyć punkt kończący tę przerwę.
- Można przekształcić sekundy na ujęcia włączając w oknie *Edit Envelope* pokazywanie skali czasu w ujęciach. Możesz przełączać te dwa widoki klikając w trzecią i czwartą ikonę, znajdującą się na dole po prawej stronie.



Kontrolowanie potencjometrów głośności i balansu w obiekcie typu Sound

Aby rozróżnić balans dźwięku i głośność obiektu typu `Sound`, tak jak to ma miejsce w przykładzie, którym zajmowaliśmy się na początku tego rozdziału, możemy zastosować metody `setVolume` i `setPan`. W ujęciu pierwszym tego samego filmu dodaj poniższe linie:

```
vol = 100;
pan = 0;
winSound = new Sound ();
winSound.attachSound ("win2");
winSound.start (0, 214748);
```

Potem stwórz nowe ujęcia — drugie i trzecie, a następnie zmień nazwę aktualnej warstwy na *actions*. W ujęciu drugim dodaj:

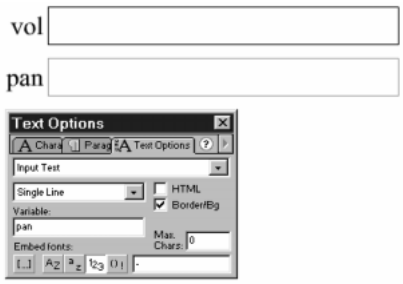
```
winSound.setVolume (vol);
winSound.setPan (pan);
```

A w ujęciu trzecim:

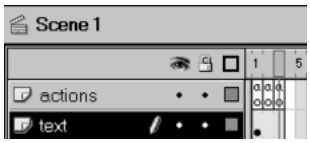
```
gotoAndPlay (2);
```

Skrypt będzie teraz tworzył zmienne nazwane `pan` i `vol`, a następnie obiekt typu `Sound` o nazwie `winSound`, który będzie odtwarzał się w ciągłej pętli. W ujęciu drugim i trzecim zachowane jest dostarczanie zmiennych, aby ustawić balans i głośność dźwięku.

Aby rozróżnić zmienne `pan` i `vol`, stwórz nową warstwę nazwaną `text`, umieść w niej dwa wejściowe pola tekstowe z opcjami pokazanymi na kolejnym rysunku, a następnie nadaj im nazwy zmiennych odpowiednio `pan` i `vol`. Nie zapomnij dodać znaku minus w polu tekstowym znajdującym się na dole po prawej stronie, bo w przeciwnym wypadku nie będziesz mógł wpisywać wartości ujemnych.



W tym przypadku *vol* i *pan* są jedynie statycznymi polami tekstowymi, których zadaniem jest przypominanie o rodzaju pola. Aby się upewnić, czy wszystko jest w porządku, zobacz, czy twoja listwa czasowa wygląda tak jak na rysunku. Skończony film znajdziesz w pliku *specialFXTest.fla*:



To, co zrobimy za chwilę, może spowodować, iż stworzymy bardzo głośny i zniekształcony dźwięk. Jeśli do twojego komputera podłączony jest zewnętrzny wzmacniacz, zmniejsz poziom głośności do około 10%, zanim przejdziesz dalej.

Gdy uruchomisz film, usłyszysz odtwarzaną w kółko melodyjkę. Będziesz miał również możliwość wprowadzania aktualnych wartości głośności i balansu dźwięku za pomocą pól tekstowych. Wprowadzenie do pola `vol` wartości z zakresu od 0 do 100 pozwoli na kontrolę głośności od 0 do 100%, zaś umieszczenie w polu `pan` wartości z zakresu –100 do 100 będzie dzieliło dźwięk między prawy a lewy głośnik.

Umieściłem w tym przykładzie pola tekstowe zamiast suwaków z jeszcze jednego powodu — Flash umożliwia wykroczenie poza ustalone wartości maksymalne zarówno głośności, jak i balansu dźwięku. Jeśli wprowadzisz wartości spoza zaprezentowanego wcześniej przedziału, odkryjesz, iż Flash generuje przesterowane wersje próbek dźwiękowych. Nie jest to pożądany efekt, gdy uzyskasz go przez przypadek, ale jeśli specjalnie zastosujesz do tego próbki gitary albo basów, może się to okazać przydatne w kompozycji dźwiękowej.

Tworzenie efektu pogłosu

W pewnej chwili przez twoją głowę mogła przemknąć myśl, iż nawet tak proste polecenia, jak te, które poznaliśmy do tej pory, mogą posłużyć to stworzenia dosyć zaawansowanych efektów, takich jak na przykład pogłos. Za pomocą pokazanego poniżej skryptu można uzyskać lekki efekt pogłosu. Jako punkt wyjścia zastosuj niezmodyfikowany plik *specialFX.fla*, a do jego pierwszego ujęcia dodaj następujący kod:

```
winSound = new Sound ();
winSound2 = new Sound ();
winSound.attachSound ("win2");
winSound2.attachSound ("win2");
winSound.start (0, 1);
winSound2.start(0.05, 1);
```

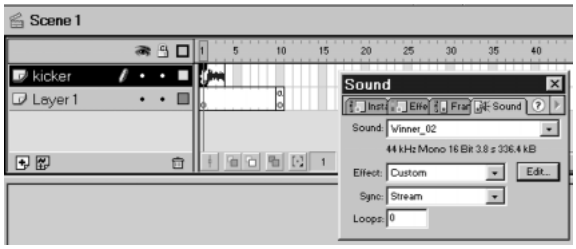
W efekcie utworzone zostaną dwa obiekty typu `Sound` powiązane z tym samym dźwiękiem, które będą odtwarzane z niewielkim opóźnieniem jednego względem drugiego. Wydawać by się mogło, iż jeśli zmienisz ostatnią linię, tak jak to pokazano, to nie usłyszysz żadnego opóźnienia:

```
winSound.start (0, 1);
winSound2.start(0, 1);
```

... ale w rzeczywistości, gdy to przetestujesz, będzie je nadal słychać.

Jest to spowodowane tym, iż we Flashu zdarzenia dźwiękowe i dźwięki wywoływane przez język ActionScript nie są synchronizowane, a zatem dwa „identyczne" dźwięki rozpoczynające się w tym samym ujęciu niekoniecznie zaczynają się czy kończą w tym samym czasie. Nawet nie trwają tak długo, jak powinny. Ten problem zniechęcił wielu projektantów do kompleksowego zastosowania dźwięku we Flashu, ale można temu zaradzić w dosyć prosty sposób.

Można „zmusić" odtwarzacz dźwięku we Flashu do synchronizacji z listwą czasową, gdy odtwarzany będzie dźwięk *przesyłany strumieniowo*. Musi on być zsynchronizowany z listwą, a zatem rozwiązanie naszego problemu polega na dołączeniu krótkiego dźwięku tego typu do początku każdej listwy czasowej, w której chcemy zsynchronizować dźwięk.



W ujęciu pierwszym twojej listwy czasowej, wstaw krótki dźwięk wczytywany strumieniowo. Może to być dowolny dźwięk. Aby nie dało się go usłyszeć, ustaw głośność na zero w oknie *Edit Envelope*. Dźwięk musi trwać przez kilka ujęć, po których możesz bezpiecznie zacząć wydawać polecenia dźwiękowe w języku ActionScript wiedząc, że

są one już poprawnie zsynchronizowane. Zawsze dodaję ten dźwięk do nowej warstwy nazwanej *kicker* („kopniak"), ponieważ dokładnie określa ona swoje działanie — daje odtwarzaczowi dźwięku we Flashu solidnego kopa, aby działał poprawnie!

Musisz zatem przesunąć kod, który wcześniej wpisałeś tak, aby znajdował się poza „kopniakiem" (sam usunąłem go z ujęcia pierwszego i umieściłem w dziesiątym):

```
winSound = new Sound ();
winSound2 = new Sound ();
winSound.attachSound ("win2");
winSound2.attachSound ("win2");
winSound.start (0, 1);
winSound2.start (0.05, 1);
stop ();
```

Zgodnie z przewidywaniami moja melodyjka odtwarza się z półsekundowym pogłosem. Aby sprawdzić, czy występuje prawidłowa synchronizacja, trzeba ustawić opóźnienie w obu obiektach typu `Sound` na zero. W rezultacie powinniśmy otrzymać oryginalny dźwięk, który będzie oznaczał, że pliki odtwarzane są w tym samym czasie. Dokładnie to usłyszymy. Z uwagi na to, iż ta naprawa jest kluczowa w zaawansowanym tworzeniu dźwięku we Flashu, możesz znaleźć wszystko to, nad czym przed chwilą pracowaliśmy w pliku *synchFX.fla*. Gdybyś nie mógł dać sobie z czymś rady, przyjrzyj mu się uważnie.

Jeśli dysponujesz gitarą elektryczną (sam korzystałem podczas nagrywania z gitary *Fender Strat*), możesz spróbować zająć się wypróbowywaniem niektórych dźwięków, a następnie pobawić się w odtwarzanie ich dodając efekt przesterowania i pogłos. To świetna zabawa! Przypomniała mi się niezwykła noc z moją gitarą, Flashem, kilkoma piwami i ogromnym wzmacniaczem podłączonym do gitary... Wtedy uświadomiłem sobie, że podobne efekty są możliwe. Chociaż nie sądzę, że spotkało się to z życzliwym przyjęciem moich sąsiadów...

Kontrolowanie wielu obiektów typu Sound

Gdy wiesz już, jak można tworzyć podstawowe obiekty typu `Sound`, zademonstruję na praktycznym przykładzie, jak można poradzić sobie z pewnymi prostymi — ale nie opartymi na listwie czasowej dźwiękami. Nie wiem, czy przypominasz sobie grę z zamierzchłych czasów zatytułowaną „Kosmiczni najeźdźcy" (*Space Invaders*[1])? Podczas gry można było usłyszeć głęboki odgłos przypominający bicie serca, który przyśpieszał, gdy pojazdy obcych schodziły coraz niżej w dół ekranu. Postaramy się stworzyć taki efekt. Będziemy sprawdzać położenie kosmicznych najeźdźców, a następnie w stosunku do tego, jak blisko dolnej krawędzi ekranu będą się oni znajdowali, wybierzemy odpowiedni dźwięk. Postaramy się również opracować układ sterowania służący do ustawienia ogólnego poziomu głośności i balansu dźwięku.

[1] Gra pochodzi z początku lat osiemdziesiątych. Początkowo funkcjonowała na konsolach w salonach gier, później została przeniesiona na wszystkie platformy sprzętowe (od ośmiobitowych po komputery typu PC). Polegała na zestrzeliwaniu pojazdów kosmicznych, które pojawiały się na górze ekranu, a w trakcie gry stopniowo obniżały swój lot. Gra kończyła się, gdy któryś z obcych zderzał się ze statkiem pilotowanym przez gracza — *przyp. tłum.*

Skończony przykład znajduje się w pliku *invader.fla*, ale jeśli chcesz krok po kroku śledzić to, jak powstawał, lepiej będzie, gdy wykorzystasz do tego celu plik *invaderSound.fla*, gdyż zawiera on zaimportowane do biblioteki pliki dźwiękowe. Niezależnie od tego, co wybierzesz, podzielimy to ćwiczenie na dwie części:

- tworzenie efektu dźwiękowego;
- wprowadzenie możliwości sterowania głośnością i balansem dźwięku.

Efekt dźwiękowe „Kosmicznych najeźdźców"

Gdy przyjrzysz się bibliotece, zauważysz pięć dźwięków nazwanych od *Background_Heartbeat1* do *Background_Heartbeat5*. Mają one przypominać przyśpieszające bicie serca. Będziemy dobierać odpowiedni dźwięk do określonego położenia najeźdźców na ekranie.

Za pomocą okna dialogowego *Symbol Linkage Properties*, z którym zapoznaliśmy się wcześniej, nadaj tym pięciu dźwiękom nazwy współdzielenia `heart1` do `heart5`. Jak napisałem nieco wcześniej, Flash nie sprawdza tego, czy identyfikatory są niepowtarzalne. Co więcej — jeśli popełnisz błąd w zapisie jakiejś nazwy program nie wyświetli żadnego komunikatu mówiącego o błędzie (taki sztuczny obiekt po prostu nie będzie działał) — a zatem uważaj na to!

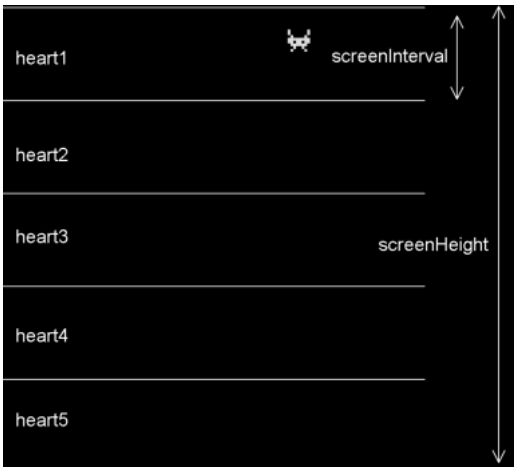
Do ujęcia pierwszego, które znajduje się na głównej listwie czasowej, dołącz poniższy kod:

```
// Inicjowanie obiektów typu Sound
sound1 = new Sound();
sound1.attachSound("heart1");

sound2 = new Sound();
sound2.attachSound("heart2");

sound3 = new Sound();
sound3.attachSound("heart3");

sound4 = new Sound();
sound4.attachSound("heart4");

sound5 = new Sound();
sound5.attachSound("heart5");
```

Dzięki temu stworzymy obiekty typu `Sound`. Następnie zostaną do nich dołączone dźwięki, które będą nam potrzebne. Zwróć uwagę na to, że w przypadku, gdy nie określamy ścieżek adresowych na etapie tworzenia obiektów, wszystkie dźwięki są umieszczone na głównej listwie czasowej. Będziemy się na tym opierać w przyszłości.

Następnie musimy napisać skrypt, który będzie odpowiadał za nasz efekt dźwiękowy. Aby tego dokonać, stworzymy klip filmowy, który będzie się poruszał w dół ekranu. Podczas jego ruchu będziemy się przełączać między pięcioma odgłosami bicia serca i wybierać coraz szybsze, im niżej będzie znajdował się kosmiczny najeźdźca.

Aby zobaczyć, jak powinno to wyglądać, spójrz na rysunek.



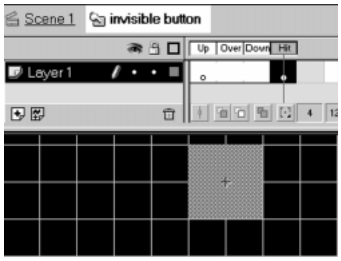
Jeśli podzielimy całą wysokość ekranu na pięć równych przedziałów, z których każdy będzie miał szerokość równą `screenHeight /5`, możemy zacząć od ustawienia dźwięku `heart1` w najwyższym przedziale, a potem — po przekroczeniu granicy następnego — przejdziemy do `heart2` i tak dalej aż do `heart5` w najniższym przedziale. Aby nasz film był nieco prostszy, ruch kosmicznego najeźdźcy będzie odbywał się za pomocą przeciągania myszą.

Do skryptu, który stworzyliśmy wcześniej w ujęciu pierwszym, dodaj następujący kod:

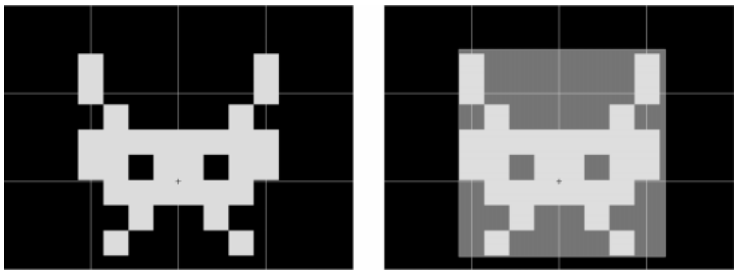
```
// inicjowanie zmiennych globalnych
screenHeight = 400;
screenWidth = 550;
```

Zdefiniowałem w tym miejscu zmienne służące do przechowywania wysokości i szerokości ekranu, ponieważ wiem, że wcześniej czy później będą nam potrzebne.

Aby obcego dało się przeciągać po ekranie, musimy zastosować określony przycisk. Najbardziej odpowiedni będzie niewidoczny przycisk, z którym zapoznaliśmy się w poprzednim rozdziale. Stwórz jeden przycisk tego typu i nadaj mu nazwę *invisible button*.



Teraz stwórz klip filmowy o nazwie *alien1* i w pierwszym ujęciu znajdującym się na warstwie *layer 1*, umieść rysunek najeźdźcy z kosmosu (chciałem, aby mój przypominał wyglądem wizerunki obcych, które są znane ze starych gier, ale nie musisz się w ten sposób ograniczać; możesz być bardziej twórczy, jeśli masz na to ochotę). Gdy będziesz już zadowolony z wyglądu swojego najeźdźcy, przeciągnij na niego niewidzialny przycisk, a następnie dopasuj do rozmiarów najeźdźcy, jeśli zajdzie taka potrzeba.



Aby klip filmowy dało się przeciągać, dołącz do przycisku poniższy skrypt:

```
on (press) {
   startDrag ("");
}
on (release, releaseOutside) {
   stopDrag ();
}
```

Na końcu musimy dodać kod przekształcający aktualną pozycję obcego w odpowiedni obiekt typu `Sound`. Przeciągnij klon klipu *alien1* na obraz znajdujący się na nowej warstwie *graphic* w ujęciu pierwszym i dodaj do niej następujący kod:

```
// inicjowanie
onClipEvent (load) {
   screenInterval = _root.screenHeight/5;
   oldHeight = -1;
}
```

Gdy ten klip się wczyta, Flash będzie „przyglądał się" zmiennej globalnej `screenHeight` i tworzył zmienną lokalną o nazwie `screenInterval`. Nawiasem mówiąc — jest to lepsza technika niż pozornie prostsze zastosowanie polecenia:

```
screenInterval = 400 / 5;
```

Zastosowałem takie rozwiązanie, bo wziąłem pod uwagę to, że jeśli będziesz budował kilka klipów filmowych, z których każdy będzie używał zmiennej `screenInterval` (czy wywodzących się z niej innych zmiennych), będziesz za każdym razem miał wiele pracy chcąc zmienić rozmiar ekranu. Umieszczenie takich zmiennych jak `screenHeight` w ujęciu pierwszym głównej listwy czasowej na dłuższą metę pozwala oszczędzić wiele czasu, ponieważ w ten sposób masz do dyspozycji jedno, sensowne miejsce, w którym możesz wprowadzać zmiany.

Druga zmienna — `oldHeight` — reprezentuje ostatni znany przedział, w którym umieszczony był obcy. W przedziale pierwszym `oldHeight` równa się `1` i pewnie sam możesz się domyślić, jakie wartości przyjmować będzie w kolejnych przedziałach — drugim, trzecim, czwartym i piątym. Porównując tę zmienną z aktualną wysokością

(nad którą będziemy pracować w następnym ujęciu) możemy ustalić, czy kosmiczny najeźdźca przeszedł do kolejnego przedziału (a zatem, czy należy rozpocząć odtwarzanie kolejnego dźwięku). Inicjując zmienną `oldHeight` na początku z wartością `-1`, czyli wskazując przedział, który nie istnieje, zmuszamy Flasha, aby rozpoczynał odtwarzanie dźwięku od razu. Zastosowanie „pozornej" wartości pozwala się upewnić, że kod „zajmuje się" czymś na początku filmu. Jest to jeszcze jedna, często spotykana technika charakterystyczna dla pisania skryptów oraz jeszcze jedna sztuczka, której warto się nauczyć.

Za kodem inicjującym dodaj:

```
onClipEvent (enterFrame) {
   height = Math.floor (_y / screenInterval) + 1;

   if (height != oldHeight) {
      _root.sound1.stop ("heart1");
      _root.sound2.stop ("heart2");
      _root.sound3.stop ("heart3");
      _root.sound4.stop ("heart4");
      _root.sound5.stop ("heart5");

      if (height == 1) {
         _root.sound1.start(0, 999);
      } else if (height == 2) {
         _root.sound2.start(0, 999);
      } else if (height == 3) {
         _root.sound3.start(0, 999);
      } else if (height == 4) {
         _root.sound4.start(0, 999);
      } else {
         _root.sound5.start(0, 999);
      }
      oldHeight = height;
   }
}
```

Pierwsza linia tego detektora zdarzeń działa w obrębie przedziału, w którym się znajdujemy. `Math.floor` nadaje wartość całkowitą w zakresie od 0 do 4, a dodając do niej 1 upewniamy się, że `height` jest liczbą z zakresu od 1 do 5, która odpowiada naszym pięciu dźwiękom (od `heart1` do `heart5`).

Warunek, który pojawia się w pierwszym wyrażeniu warunkowym `if`, porównuje wartość zmiennej `height` z `oldHeight`, aby stwierdzić, czy obcy przekroczył granicę przedziału. To porównanie zawsze będzie miało wartość logiczną `true`, jeśli będzie to pierwsze uruchomienie skryptu, ponieważ w tym wypadku `oldHeight` jest równe `-1`.

Jeśli przekroczyliśmy granicę, to pierwszą rzeczą, jaką musimy zrobić, jest wyłączenie aktualnie odtwarzanego dźwięku. Możemy do tego celu wykorzystać różne tablice, albo użyć innych sposobów, ale znacznie prostszym i szybszym rozwiązaniem jest wyłączenie ich wszystkich.

Ostatnia instrukcja `if...else if...` sprawdza zatem wartość zmiennej `height`, aby określić, który z obiektów typu `Sound` uruchomić ponownie. Na końcu ustawiamy wartość zmiennej `oldHeight = height`, aby przejść do następnej iteracji.

Zauważ, że to, co w tym przypadku wywołuje polecenie `stop`, zawiera ścieżkę adresową do listwy czasowej, na której został stworzony obiekt typu `Sound`: `_root`. Wydawać by się mogło, że polecenie `_root.sound1.stop()` będzie wyłączało `sound1`, a `_root.sound2.stop()` — `sound2`, ale metoda `stop` nie działa w ten sposób. Jeśli nie umieścisz czegoś w nawiasach, Flash będzie wyłączał wszystkie aktualnie odtwarzane dźwięki, ale zwykle właśnie o to chodzi. Aby program umiał rozróżniać obiekty typu `Sound` i potrafił wyłączać każdy z nich oddzielnie, musisz umieścić identyfikator współdzielenia w nawiasie, tak jak pokazano to wcześniej. Opowiem o tym nieco więcej w dalszej części rozdziału.

Oczywiście metoda `_root.sound1.stop()`, która będzie zatrzymywała odtwarzanie wszystkich dźwięków jest w tym przypadku dokładnie tym, co zamierzamy osiągnąć. Jednak miałbym spory kłopot, gdybym nawiązywał do każdego obiektu typu `Sound` i próbował każdy z nich osobno wyłączać, aby pokazać, jak należy to zrobić, gdy trzeba zatrzymać dźwięki, które odtwarzają się niezależnie od siebie.

Jeśli uruchomisz teraz film, usłyszysz przerażające bicie serca, które będzie się natężać, gdy będziesz przemieszczał obcego najeźdźcę w dół ekranu. A zatem zakończyliśmy nasze zadanie. Stworzyliśmy bardzo prosty skrypt umożliwiający wybór pomiędzy kilkoma różnymi odgłosami dźwiękowymi na podstawie zdarzeń warunkowych. Może to być znacznie lepsza technika niż struktura opierająca się na głównej listwie czasowej, do której stosowania byliśmy zmuszeni, gdy nie znaliśmy języka ActionScript. Obecnie możemy tworzyć efekty dźwiękowe, które będą reagować na wszystko, co wskażemy, nie muszą być odtwarzane po kolei ani dołączane do prostego przycisku czy klipu filmowego.

Dołączanie dźwięków za pomocą nazw ścieżek adresowych

Do tej pory łączyliśmy dźwięki z biblioteki z obiektami typu `Sound`, ale we Flashu można zastosować dźwięk w sposób znacznie bardziej elastyczny. W rzeczywistości możemy łączyć z tymi obiektami *wszystko*, a pod tym słowem-kluczem kryje się:

- listwa czasowa, taka jak `_root` czy `_parent`;
- wczytany poziom, taki jak na przykład `_level34`;
- listwa czasowa klipu filmowego, taka jak na przykład `_root.myMovie`.

Pozwala nam to kontrolować wszystkie dźwięki znajdujące się na nazwanych poziomach (włączając w to obiekty typu `Sound` i inne dołączone do ujęć dźwięki, które nie korzystają ze skryptów w języku ActionScript). Wyjaśnia to, dlaczego Flash „nalega”, aby obiekty typu `Sound` znajdowały się na listwach czasowych, na których zostały stworzone — tylko wtedy możesz dodawać dalsze zagnieżdżone poziomy kontroli dźwięku oparte na listwach czasowych.

Główny układ sterowania dźwiękiem

W tym przykładzie będziemy korzystać z tych cech Flasha, które pozwalają na dołączenie listwy czasowej do obiektów typu `Sound`, co w efekcie pozwala stworzyć układ sterowania dźwiękiem. Będzie on równocześnie oddziaływał na wszystkie dźwięki, których

używaliśmy do tej pory. Jak zwykle gotowy przykład znajduje się w pliku *invader2.fla*, ale jeśli chcesz, możesz równie dobrze kontynuować pracę wykorzystując wcześniejszy przykład, zaczynając w miejscu, w którym skończyliśmy.

W pierwszym ujęciu na głównej listwie czasowej dodaj dwie linie na końcu listingu, tak jak pokazano to poniżej:

```
// Inicjowanie zmiennych globalnych
screenHeight = 400;
screenWidth = 550;

// Inicjowanie obiektów typu Sound
sound1 = new Sound();
sound1.attachSound("heart1");
sound2 = new Sound();
sound2.attachSound("heart2");
sound3 = new Sound();
sound3.attachSound("heart3");
sound4 = new Sound();
sound4.attachSound("heart4");
sound5 = new Sound();
sound5.attachSound("heart5");

soundAll = new Sound();
soundAll.attachSound(_root);
```

Ten obiekt typu `Sound` jest inny niż pozostałe, ponieważ odnosi się on do całej listwy czasowej, nie zaś do określonego dźwięku. Jako że celem tym razem jest ścieżka adresowa (a nie nazwa podana jako ciąg znaków), nie musi być ona ujęta w cudzysłów.

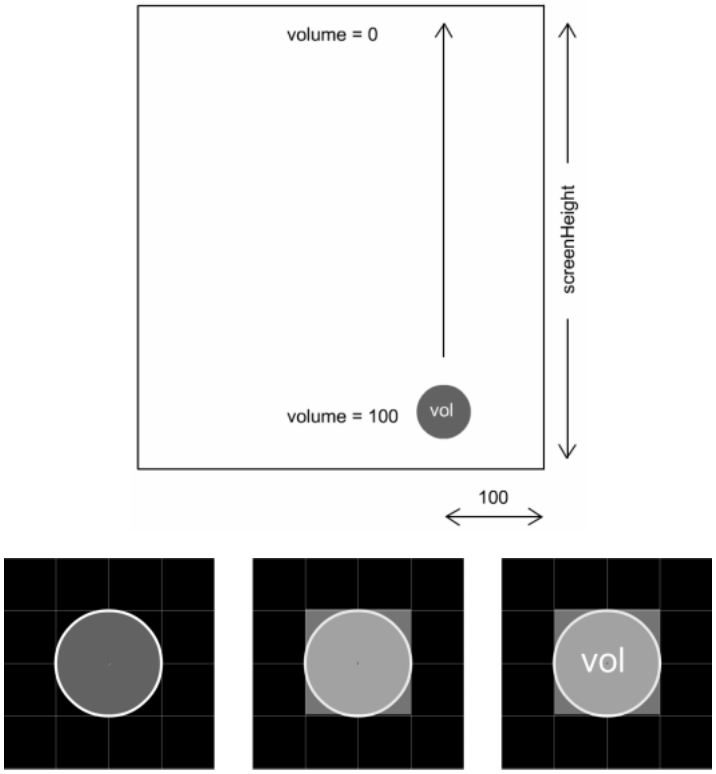
Jakakolwiek manipulacja na obiekcie `soundAll` będzie wpływała na wszystkie dźwięki znajdujące się na ścieżce `_root` — w tym przypadku na wszystkie inne dźwięki, które pojawią się w przykładzie. Oczywiście będzie można kontrolować pojedynczy obiekt typu `Sound`, ale zajmuje się tym również obiekt `SoundAll`, który będzie definiował maksymalną głośność wszystkich dźwięków.

Aby zobaczyć, jak to działa, dodajmy elementy sterujące, które będą odpowiadały za to, aby obiekt wykonywał pewną pracę. Stworzyłem bardzo proste przeciągane przyciski, aby zademonstrować działanie obiektu, ale możesz zastosować jakieś „odlotowe" rozwijane menu z metalizowanymi, bardzo realistycznie wyglądającymi suwakami i wykresami przedstawionymi za pomocą diod LED, kiedy będziesz to samo robił w swojej witrynie.

Na początku stworzymy swego rodzaju suwak służący do regulacji głośności, który będzie można przeciągać w dół i w górę. Jego położenie początkowe będzie znajdowało się na dole ekranu po prawej (100 pikseli od krawędzi). Ruch suwaka w górę będzie zmniejszał całkowitą głośność aż do 0%.

Suwakiem będzie przesuwany krążek, który będzie działał podobnie jak najeźdźca z kosmosu. A zatem stwórz nowy klip filmowy, nazwij go *volWidget*, zmień nazwę jego jedynej warstwy na *text*, a następnie dodaj dwie nowe warstwy — *button* i *graphics*.

W tej ostatniej umieść małe koło, jak pokazano na lewym rysunku znajdującym się poniżej. Upewnij się, czy punkt odniesienia znajduje się w środku tego koła. W warstwie



buttons dodaj inny klon niewidocznego przycisku i dopasowuj jego rozmiar tak, aby pokrył sobą okrąg. Na końcu w warstwie *text* umieść statyczne pole tekstowe z napisem „vol" (jest to skrót od angielskiego *volume* — głośność).

Aby krążek można było przeciągać, musisz dołączyć poniższy kod do skryptu niewidocznego przycisku:

```
on (press) {
   startDrag ("", false, x, 0, x, range);
}
on (release, releaseOutside) {
   stopDrag ();
}
```

Zwróć uwagę na to, że `x` i `range` są zmiennymi zdefiniowanymi gdzie indziej (pierwsza określa położenie poziome, a druga ustala pionowy zasięg suwaka). Ogranicza to przeciąganie dźwigni wewnątrz pola granicznego o zerowej szerokości (a zatem będzie ją można przesuwać tylko w górę i w dół) w granicach określonych wysokością ekranu.

Podobnie jak w przypadku najeźdźcy z kosmosu detektor zdarzenia dołączony do klonu tego klipu filmowego zarządza dźwiękiem, który powstanie podczas ruchu suwaka.

```
onClipEvent (load) {
   range = _root.screenHeight;
   scale = range / 100;
   x = _root.screenWidth-100;
```

```
    _x = x;
    _y = range;
}

onClipEvent (enterFrame) {
    vol = _y / scale;
    _root.soundAll.setVolume (vol);
}
```

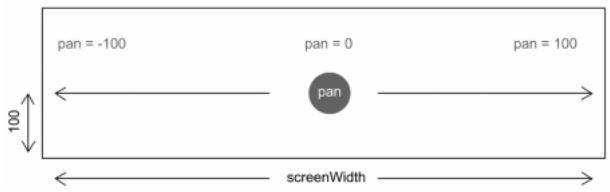
W detektorze zdarzenia `load`, w którym zdefiniowane są również `x` i `range`, `scale` (skala) jest współczynnikiem, który musimy zastosować wraz z aktualnym położeniem naszego suwaka tak, aby uzyskać zasięg od 0 do 100. Dwie ostatnie linie umieszczają suwak w położeniu początkowym, które znajduje się na dole po prawej stronie ekranu, 100 pikseli od krawędzi. Zauważ, że kod może korzystać ze zmiennych określających wymiary ekranu (`screenHeight`, `screenWidth`), które ustawiliśmy na listwie czasowej `_root`.

W detektorze `enterFrame` pierwsza linia przekształca położenie suwaka na wartość mieszczącą się między 0 a 100 dzieląc ją przez współczynnik `scale`. Druga linia następnie nadaje tę wartość głośności wszystkch dźwięków dołączonych do głównej listwy czasowej.

Jeśli chcesz, możesz przetestować w tej chwili film, ale musimy zastosować jeszcze sterowanie balansem, które będzie bardzo podobne *volWidget*. Rozsądnym rozwiązaniem byłoby skopiowanie go i zmienienie w bibliotece nazwy na *panWidget*. Biblioteka powinna wyglądać tak, jak na rysunku.

Suwak balansu będzie przemieszczał się z prawej na lewą stronę, 100 pikseli od dołu ekranu. Gdy znajdzie się pośrodku (czyli osiągnie wartość 0 — będzie w równowadze), dźwięk będzie rozdzielony po równo między głośnikami, a położenia najbardziej oddalone na prawo i lewo będą odpowiadały ustawieniom tylko prawej i tylko lewej strony — prawemu i lewemu głośnikowi.

Aby to zaprogramować, musimy przeciągnąć kopię *panWidget* na obraz tego samego ujęcia i tej samej warstwy, w której znajdują się już dwa inne klipy filmowe i zaprojektować nieco inną parę detektorów zdarzeń:




```
onClipEvent (load) {
   range = _root.screenWidth;
   midPoint = range/2;
   scale = range/200;
   y = _root.screenHeight-100;
   _x = midPoint;
   _y = y;
}
onClipEvent (enterFrame) {
   pan = (_x-midPoint)/scale;
   _root.soundAll.setPan(pan);
}
```

W tym przypadku `midPoint` jest środkowym położeniem `x` na ekranie, a zatem punkt początkowy naszego suwaka znajduje się w kierunku `x`. Tym razem obowiązuje skala od –100 do 100 (200 jednostek), a zatem musimy podzielić zasięg przez 200, aby uzyskać współczynnik skali `scale`.

Pierwsza linia detektora `enterFrame` przekształca właściwość `_x` klipu filmowego w skalę od –100 do 100 poziomu mocy dźwięku, natomiast druga linia dołącza wartość `pan` do obiektu `soundAll` na listwie czasowej `_root`.

Przedostatnim zadaniem jest zmiana skryptu w niewidocznym przycisku na:

```
on (press) {
   startDrag ("", false, 0, y, range, y);
}
on (release, releaseOutside) {
   stopDrag ();
}
```

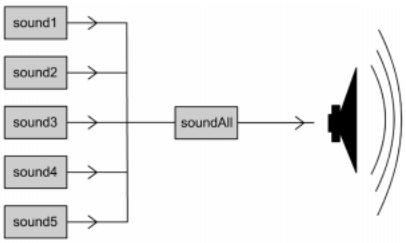
Ogranicza to ruch suwaka do poziomego paska, który znajduje się 100 pikseli nad dolną krawędzią ekranu i utrzymuje go w widocznym obszarze.

Interesującym efektem ubocznym, który uzyskaliśmy zajmując się dwoma suwakami jest to, iż nie ma znaczenia, w którym miejscu obrazu je ustawisz, gdyż za pomocą detektorów `onClipEvent(load)` przemieszczą się one do swoich właściwych pozycji. Jeśli przetestujesz teraz film (albo moją wersję — *invader2.fla*), zobaczysz, że:

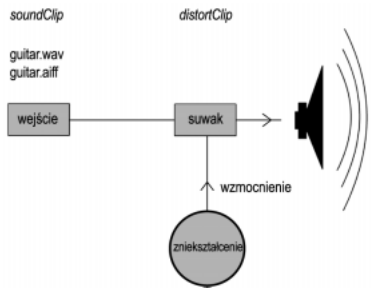
- poruszanie najeźdźcą powoduje wybór odpowiedniego dźwięku do odtworzenia;
- ruch obu suwaków zmienia wartości głośności i balansu niezależnie od tego, jaki dźwięk się aktualnie odtwarza.

Dodatkowe efekty obiektów typu Sound

Film, który właśnie złożyliśmy jest w rzeczywistości całkiem skomplikowany. Na głównej listwie czasowej dokonuje się wyboru jednego z kilku obiektów typu `Sound` (zależnie od pozycji kosmicznego najeźdźcy). Innymi słowy — ten wybór może się opierać na znacznie dziwniejszych warunkach (możemy również zastosować kontrolę balansu i głośności dźwięku pojedynczych obiektów). Rezultat tego wyboru przechodzi również na kolejny obiekt typu `Sound` — `soundAll`. Używaliśmy go jako głównego systemu sterowania, ponieważ pozwalał na manipulowanie wszystkimi dźwiękami dołączonymi do ścieżki `_root` (niezależnie od tego, jak wielu z nich to dotyczyło). Skutecznie stosowaliśmy hierarchię kontroli i efekty następujących po sobie poziomów kontroli są addytywne, co zwiększa możliwości wszystkich rodzajów interesujących efektów.



Nasz system działał zgodnie z założeniem, że dźwięki `sound1` – `sound5` kontrolowały nieskompresowane pliki dźwiękowe, ale następny dźwięk w kolejności — `soundAll` kontrolował *poziom*, który zawierał wszystkie dźwięki. Używając w ten sposób obiektów typu `Sound` do stworzenia hierarchii, możemy przemieścić dźwięk przez kilka różnych obiektów tego typu. Pozwoli to na emulowanie rzeczywistego obwodu filtra elektronicznego, w którym efekty kolejnych dźwięków są dodawane.



Przyjrzyj się uważnie powyższemu rysunkowi. Na listwie czasowej klipu filmowego *soundClip* zdefiniowany jest obiekt typu `Sound`, który z kolei połączony jest z plikiem zawierającym dźwięk gitary. Z kolei *distortClip* jest innym klipem zawierającym obiekt typu `Sound`, dla którego docelową listwą jest listwa czasowa klipu *soundClip*. Ten ostatni obiekt dźwiękowy ma poziom głośności ustawiony na większy niż 100 (korzystając przy tym z wartości odczytywanych z suwaka). Nie musisz się jednak zatrzymywać w tym miejscu, ponieważ możesz stworzyć trzeci klip (nazwany na przykład *reverbClip*), który pobierze dane wejściowe z klipu *distortClip* i doda efekt pogłosu powielając ten klip i dodając sterowane suwakiem opóźnienie między nim a kopią klonu.

Końcowy sygnał wyjściowy będzie natomiast przesterowaną wersją oryginalnego dźwięku z dodanym pogłosem. Wszystko to może być zmieniane w czasie rzeczywistym za pomocą kilku suwaków! Kto jeszcze będzie chciał grać na gitarze, skoro mamy Flasha?

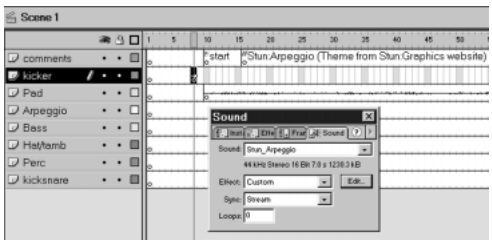
Tworzenie i kontrola ścieżek dźwiękowych

Zaleta, jaką jest zdolność określania docelowej listwy czasowej podczas definiowania obiektu typu `Sound`, pozwala na kontrolowanie listwy czasowej z dźwiękiem dołączonym do jej ujęć (za pomocą panelu *Sound* i powiązanego z nim okna *Edit Envelope*). Możesz dzięki temu na listwie czasowej układać w określonym porządku zestaw dźwięków, aby tworzyć ścieżki dźwiękowe i inne kompozycje z pojedynczych próbek. Flash umożliwia nam korzystanie maksymalnie z ośmiu kanałów dźwiękowych odtwarzanych w tym samym czasie, a zatem za pomocą tej metody możesz tworzyć coś, co odpowiada ośmiokanałowej sekwencji próbek (czasami nazywanej przez muzyków **trackerem**).

Tworzenie ścieżki dźwiękowej

W tym miejscu podsumujemy sposoby tworzenia ścieżki dźwiękowej za pomocą kompozycji *stun:arpeggio*, tak jak jest to pokazane w innej książce wydawnictwa friends of ED — *Foundation ActionScript*[2], ale później szybko przejdziemy do zaawansowanej kontroli każdej ścieżki dźwiękowej za pomocą języka ActionScript. Jeśli szczególnie interesuje cię tworzenie ścieżek dźwiękowych, to potrzebne wiadomości znajdziesz w poświęconym temu zagadnieniu rozdziale 11. książki *Foundation Flash*[3].

Listwa czasowa ścieżki dźwiękowej powinna zawierać pojedynczą scenę (utracisz synchronizację, jeśli połączysz ją z inną sceną), więc musisz rozpocząć opracowywanie listwy czasowej od dźwięku, który będzie ją synchronizował („kopniak” — opisany nieco wcześniej).

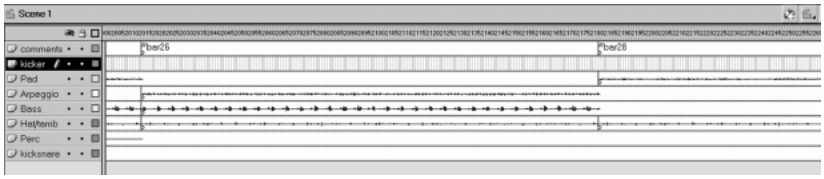


Możesz zatem zacząć dołączanie dźwięków do listwy czasowej (jeden na każdą warstwę tak, jak pokazano to na rysunku). W panelu *Sound* ustaw wartość *Event* w polu *Sync* dla tych wszystkich dźwięków.

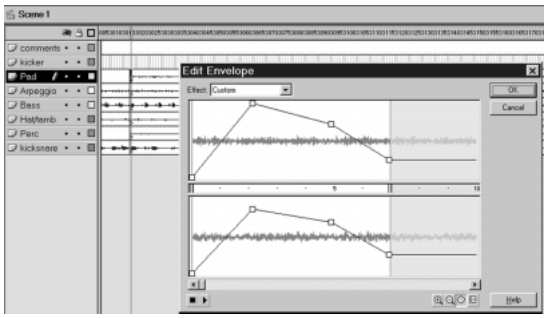
---

[2] Książka została wydana nakładem wydawnictwa Helion, pod tytułem *Flash 5 ActionScript. Podstawy — przyp. red.*

[3] Książka została wydana nakładem wydawnictwa Helion, pod tytułem *Flash 5. Podstawy — przyp. red.*



Możesz wykorzystać okno *Edit Envelope* do wyciszania lub zmiany balansu każdego dźwięku. Używam tej funkcji na końcu ścieżki dźwiękowej, aby ją wyciszyć, jak pokazano to na rysunku.



Warto w tym przypadku zdawać sobie sprawę z tego, że gdy dochodzi do kontroli ścieżki dźwiękowej za pomocą obiektów typu `Sound`, obwiednie głośności w oknie *Edit Envelope* będą dodawane do wszystkich zastosowanych przez nas wartości ustawianych za pomocą obiektów sterujących. Umożliwia nam to zastosowanie obwiedni głośności do kontrolowania mieszanych poziomów w naszej kompozycji i do kontrolowania całego dźwięku za pomocą obiektu typu `Sound`.

Jeśli to, o czym piszę, zaintrygowało cię, możesz również przyjrzeć się sposobowi wykonania kompozycji dźwiękowej we Flashu, który został przedstawiony w witrynie *killersound*. Zastosowane tam metody wykorzystują okno *Edit Envelope* do wyciszania dźwięków, które rozpoczynają się w tym samym czasie i jest to nieco lepszy sposób synchronizacji niż łatwiejsza metoda, która została zaprezentowana wcześniej (Odwiedź witrynę *www.killersound.com*, a później wybierz *HTML version* i przejdź do *Tech Center/Goodies*).

Aby zastosować ścieżkę dźwiękową, musimy stworzyć ją jako klip filmowy albo wczytany poziom. Dla tego ostatniego możesz używać skryptu podobnego do poniższego:

```
loadMovie ("soundtrack.swf", 99);
soundtrack = new Sound ();
soundtrack.attachSound (_level99);
```

Zauważ, że musisz rozpocząć wczytywanie poziomu, zanim dołączysz go do obiektu typu `Sound` (ponieważ poziom nie istnieje, dopóki go nie wczytasz). Zwykle stosuję poziom 99 jako poziom ścieżki dźwiękowej, ponieważ na tyle jest oddalony od poziomu zero, by wpływały na niego (czyli kasowały) dodatkowe poziomy, które strona główna może „chcieć” wczytać.

W celu kontrolowania ścieżki dźwiękowej — jeśli zdecydowałeś się implementować ją jako klip filmowy — możesz po prostu zastosować polecenia:

```
soundtrack = new Sound ();
soundtrack.attachSound(clip);
```

Pamiętaj, że *clip* jest ścieżką adresową do ścieżki dźwiękowej. Ponadto zapoznaj się jeszcze z dodatkowymi wskazówkami:

- Jeśli zdecydujesz się używać wczytywanych poziomów, rozważ zastosowanie zamiast nich wczytywanych celów (to znaczy pustych klipów filmowych, które są umieszczone od razu na obrazie). Pozwoli ci to odnosić się do listwy czasowej ścieżki dźwiękowej (a co za tym idzie — połączyć z nią obiekt typu `Sound`), zanim jeszcze rzeczywiście zaczniesz wczytywać zawartość (jest to — ogólnie rzecz biorąc — wielka zaleta wczytywanych celów, ponieważ zapewniają zewnętrzną kontrolę za pomocą języka ActionScript).
- Zastosowanie wczytywanego poziomu czy celu pozwala wczytywać dźwięki kolejno z głównej strony i umożliwia użytkownikowi wybór wczytywanej ścieżki dźwiękowej bez potrzeby zmiany obiektu typu `Sound` (ponieważ ścieżki adresowe pozostają te same). Daje to użytkownikowi zarówno elastyczność w wyborze ścieżki dźwiękowej, jak i pozwala łatwiej uaktualniać pliki dźwiękowe, ponieważ są one całkowicie odizolowane od głównej zawartości witryny.

Kontrolowanie ścieżki dźwiękowej

Aby kontrolować powyższy obiekt typu `Sound`, musisz zastosować skrypt oparty na tym, czemu przyglądniemy się za chwilę. Zawarłem instrukcje niezbędne do kontrolowania listwy czasowej ścieżki dźwiękowej, która implementowana jest jako wczytywany poziom, ale kontrolowanie wczytywanego celu czy listwy czasowej ścieżki dźwiękowej wewnątrz klipu filmowego będzie bardzo podobne.

Wszystko zaczyna się dosyć prosto — aby ustawić balans i głośność obiektu typu `Sound`, który reprezentuje ścieżkę dźwiękową, musisz wywołać po prostu jego metody `setPan` i `setVolume` tak, jak to robiłeś wcześniej. Aby zatrzymać odtwarzanie ścieżki dźwiękowej, użyj polecenia:

```
soundtrack.stop ();
```

Możesz również rozważyć dodanie instrukcji:

```
_level99.gotoAndStop (1);
```

Powstrzymuje to uruchomioną (ale aktualnie wyciszoną) listwę czasową ścieżki dźwiękowej przed narzuceniem jej w przyszłości konieczności wczytywania się. Jeśli użytkownik zdecyduje, że nie chce wprowadzać w ogóle muzyki, można rozważyć kasowanie poziomu ze ścieżką dźwiękową (za pomocą `unloadMovieNum (99)`), aby zwolnić pasmo, jeśli nadal trwają procesy pobierania strumieniowego (może być to sprawdzone za pomocą warunku `_level99._framesloaded = _level99._totalframes`, który będzie podczas wczytywania przyjmował wartość `false`, a gdy wczytywanie się zakończy będzie miał wartość `true`).

Aby uruchomić ponownie ścieżkę dźwiękową po powyższym zatrzymaniu, należy zastosować:

```
soundtrack.start ();
_level99.gotoAndPlay (1);
```

Jest to niezbędne (zamiast samego `soundtrack.start`), ponieważ dźwięk jest dołączony do listwy czasowej zamiast pliku dźwiękowego i chcemy, aby *listwa czasowa* odtwarzana była po to, żeby usłyszeć ścieżkę dźwiękową.

Aby znów uruchomić ścieżkę dźwiękową od początku, zakładając, że już się odtwarza, trzeba użyć:

```
soundtrack.stop ();
_level99.gotoAndPlay (1);
```

Do tej pory w rozdziale tym prezentowano podstawowe zastosowania obiektów typu `Sound` we Flashu i w połączeniu z dźwiękami opartymi na ujęciach. W następnej części zbierzemy to wszystko razem, aby stworzyć projekt, o który prosi wielu ludzi — stół mikserski we Flashu, dzięki któremu użytkownicy będą mogli tworzyć własne remiksy.

Zaawansowana kontrola dźwięku

Metody obiektu typu `Sound` we Flashu 5, którym przyglądaliśmy się do tej pory, pozwalają z grubsza na ten rodzaj kontroli, którego możesz się spodziewać po przenośnym radiu, czy prostym odtwarzaczu kasetowym z potencjometrami balansu i głośności. Jednak istnieją dwie znaczące funkcje dodatkowe, które można dodać do filmu:

- Większość poważnych systemów hi-fi pozwala na przekształcenie dźwięku stereofonicznego w monofoniczny. To powinno wkrótce stać się kwestią do rozstrzygnięcia dla autorytetów zajmujących się Siecią, ponieważ niektóre przenośne lub bezprzewodowe urządzenia (również zestawy biurowe czy systemy przeznaczone do obróbki grafiki, a nie dźwięku, które są używane w domach wielu projektantów) mogą nie mieć pełnego wyjścia stereo.
- Systemy dźwiękowe przeznaczone do profesjonalnego miksowania dźwięków pozwalają na zmianę balansu bez zmiany całkowitej głośności (albo bardziej technicznie — mocy) dźwięku. To pozwala zmienić obraz stereo danej ścieżki, a jednocześnie zachować pozycję dźwięku, którą ta ścieżka stosuje w całej kompozycji.

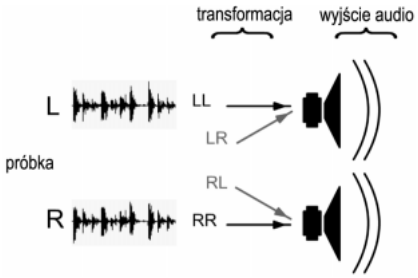
Metodą, która pozwala na włączenie obu tych cech, jest `setTransform`. Chociaż początkowo metoda może nieco zniechęcać do korzystania z niej — oprócz zdefiniowania samego obiektu typu `Sound` musisz bowiem zdefiniować drugi obiekt działający jako **obiekt kontrolujący dźwięk** — to jednak umożliwia ona kontrolę dźwięków dołączonych do filmu na dobrym poziomie. Ponieważ sposób jej działania może być nieco niejasny, wielu ludzi nie chce z niej korzystać, gdyż nie widzą w niej większej wartości niż w stosowaniu kombinacji `setVolume` i `setPan`. Mam nadzieję, że w tym miejscu zaradzimy temu raz na zawsze.

W ostatnim przykładzie w tym rozdziale zajmiemy się tym, co wielu uznaje za „świętego Graala" dźwięku we Flashu. Będzie to budowanie filmu, który interaktywnie pozwala użytkownikowi tworzyć własną mieszankę zestawu dźwięków. Jeśli przeszukasz

Sieć, znajdziesz kilka witryn, które umożliwiają zrobienie tego, ale niewiele z nich, jeśli jakaś w ogóle, korzysta ze stałej kontroli głośności i balansu. A jest to coś, czym dysponuje każdy profesjonalny DJ oraz w co jest wyposażony sprzęt nagraniowy. Na początku jednak musimy sprawdzić, jak ustawić i kontrolować obiekt typu `Sound` tylko za pomocą metody `setTransform`.

Cyfrowe miksowanie dźwięku za pomocą metody setTransform

Przyjrzyj się rysunkowi, który w najprostszy sposób pokazuje, jak wygląda przekształcenie dźwięku.



Pod względem akcji i terminologii, które widziałeś już wcześniej i które (na razie) zignorujemy, na powyższym rysunku można zauważyć kilka dość oczywistych cech. Proces przekształcania dźwięku we Flashu wygląda następująco:

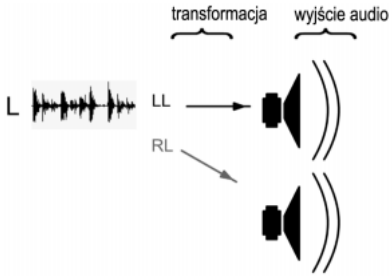
- próbka dźwiękowa dołączona jest do obiektu typu `Sound`; próbka ta we Flashu może być stereofoniczna (jak pokazano na rysunku) lub monofoniczna, co wygląda tak samo z tym wyjątkiem tego, że składnik R[4] albo L będzie zerowy;
- gdy będziesz odtwarzał dźwięk, pewien procent próbki L (LL) wprowadzony zostanie do lewego wyjścia kanału audio;
- podobnie — pewien procent próbki R (RR) wyprowadzony zostanie do prawego wyjścia kanału audio;
- procentowa wartość LL i RR zależy od aktualnej wartości balansu i głośności dźwięku tak, jak widziałeś to w przykładzie *sound.fla* przedstawionym na początku rozdziału.

Różnica w stosowaniu metody `setTransform` polega na tym, że pozwala ona manipulować nie tylko wszystkim, o czym wspomniano do tej pory, ale także dwoma dalszymi składnikami dźwięku — LR i RL.

- LR jest pewnym procentem prawej próbki, która jest odtwarzana w lewym głośniku (albo ilością sygnału prawego kanału, która przechodzi do kanału lewego).
- RL jest pewnym procentem lewej próbki, która jest odtwarzana w prawym głośniku (albo ilością sygnału lewego kanału, która przechodzi do kanału prawego).

[4] Pozostawiłem angielskie oznaczenia L (*left*) — lewy, R (*right*) — prawy — *przyp. tłum.*

Co więcej — w metodzie `setTransform` wartości LL i RR nie są połączone (jak to było w przypadku metody `setPan`, która zwiększała poziom prawego głośnika, gdy zmniejszałeś poziom lewego). To znaczy, że możesz stworzyć kombinację prawego lub lewego poziomu dźwięku (albo obu równocześnie), która nie byłaby możliwa do osiągnięcia przy zastosowaniu metody `setPan`. Na przykład metoda `setTransform` pozwala na wysłanie pewnej ilości próbki monofonicznej do obu głośników, dzięki czemu możesz stworzyć efekt stereo za pomocą języka ActionScript:



W tym przypadku możesz uzyskać dźwięk mono, który pojawia się tylko w lewym głośniku. Jednak definiując niezerowy składnik RL możesz stworzyć sygnał stereo. Rozróżniając LL i RR możesz również dynamicznie rozróżniać obraz stereo.

Ponadto metoda `setTransform` umożliwia przekształcanie sygnału stereofonicznego w monofoniczny *bez utraty informacji dźwiękowej w obu kanałach*. Jest to zupełnie inne działanie niż proste ustawianie balansu po prawej czy po lewej, ponieważ przy zastosowaniu balansu:

- tracisz informację z kanału, który wyciszyłeś do sygnału monofonicznego;
- zmniejszasz dostępną głośność, jeśli użytkownik ma system z obydwoma kanałami — L i R podłączonymi do pojedynczego głośnika (jest to zwykłe ustawienie dla systemów, do których można podłączyć tylko jeden głośnik);
- istnieje możliwość, że użytkownik korzystający tylko z jednego podłączonego głośnika nic nie usłyszy, ponieważ wysłałeś sygnał monofoniczny do złego kanału!

Jak wspomniano wcześniej metoda `setTransform` pozwala wprowadzać zmiany kontrolowania poziomu dźwięku *bez wpływu na całkowitą moc sygnału dźwiękowego*. Jest to bardzo istotne w profesjonalnym miksowaniu dźwięków. Wiele domowych komputerów podłączonych jest obecnie do wysokiej jakości wzmacniaczy i głośników, więc każda witryna, która używa tych cech przetwarzania dźwięku z pewnością będzie stosowana do przekształcania muzyki (być może będzie to odejście od powszechnej opinii dotyczącej tego, że we Flashu nie można dysponować takimi możliwościami).

Kontrolowanie obiektów typu Sound za pomocą metody setTransform

Tworzenie obiektu typu `Sound` służącego do zastosowania z metodą `setTransform` jest dokładnie takim samym procesem, jaki poznałeś wcześniej. Musisz jednak stworzyć drugi obiekt służący do kontrolowania tej metody. Jeśli chcesz wykonać ten eksperyment za pomocą tej samej próbki dźwiękowej, której używałem, poniższy projekt znajdziesz w pliku *setTransform.fla*. Gdy go uruchomisz, nie zobaczysz niczego, ale usłyszysz oryginalną próbkę stereofoniczną odtwarzaną przez dwa kanały poprzez lewy głośnik.

Aby stworzyć obiekt typu `Sound` o nazwie `soundA`, służący do kontrolowania dźwięku w bibliotece z nazwą współdzielenia `loop`, wykorzystamy następujące akcje:

```
soundA = new Sound (_root);
soundA.attachSound ("loop");
soundA.start ("loop");
```

Nie ma w tym niczego, czego nie widziałbyś już wcześniej — ostatnia instrukcja po prostu rozpoczyna jednokrotne odtwarzanie dźwięku. Jeśli używasz innego, krótszego dźwięku, możesz odtworzyć go więcej niż raz, żeby usłyszeć efekt zestawu poleceń, które właśnie dodaliśmy.

Rzeczą, którą warto w tym miejscu zapamiętać, jest fakt, że metoda `setTransform` wymaga zastosowania wartości LL, LR, RR i RL przedstawionych na wcześniejszych rysunkach, które muszą znajdować się w *pojedynczym obiekcie*. Ten obiekt może być nazwany jak tylko zechcesz (mój nazywa się `soundAtrans`, ponieważ będzie używany do przekształcania obiektu `soundA`), ale musi zawierać cztery właściwości o nazwach `ll`, `lr`, `rr` i `rl`, odpowiadające wyrażonym w procentach wartościom — odpowiednio — LL, LR, RR i RL. Następne dwie linie kodu w języku ActionScript definiują obiekt kontrolujący z jego czterema właściwościami ustawionymi po to, aby wszystkie dźwięki w próbce przechodziły do lewego głośnika:

```
soundAtrans = new Object ();
soundAtrans = {ll:100, lr:100, rr:0, rl:0};
```

Musimy jeszcze wywołać metodę `setTransform` obiektu `soundA` za pomocą obiektu kontrolującego `soundAtrans` jako jej argument:

```
soundA.setTransform (soundAtrans);
```

Należy zmienić przekształcenie po to, aby odtwarzało dźwięk monofoniczny przez oba głośniki (to znaczy ustawić wszystkie wartości właściwości na 100). W tym celu powinieneś dodać następujące linijki kodu:

```
soundAtrans.rr = 100;
soundAtrans.rl = 100;
soundA.setTransform (soundAtrans);
```

Jeśli chcesz, możesz stworzyć różne efekty stereo za pomocą zmian właściwości dźwięku. Spróbuj ustawić w obiekcie kontrolnym w pliku *setTransform.fla* następujące wartości (na tym etapie polecam używanie mojego pliku FLA, ponieważ wybrałem próbkę dźwiękową w ten sposób, aby przedstawiała pewne cechy, które mogą nie być oczywiste we wszystkich przypadkach):

```
soundAtrans = {ll:100, lr:0, rr:100, rl:0};
```

W efekcie uzyskamy normalny dźwięk stereofoniczny. Dokładnie tak wygląda normalny dźwięk bez żadnego przekształcenia.

```
soundAtrans = {ll:50, lr:0, rr:50, rl:0};
```

To da nam dźwięk stereo odtwarzany z 50% wartością całkowitej głośności.

```
soundAtrans = {ll:100, lr:-100, rr:0, rl:0};
```

Zanim przejdziemy dalej, musisz coś zrozumieć. W rzeczywistości nie usłyszysz nic, ponieważ prawy i lewy kanał są *niemal* identyczne. Wybierając wartość –100 dla składnika prawego głośnika wprowadzoną do lewego głośnika, odejmujemy kształt fali, co daje nam niemal zupełną ciszę.

Dla umysłów technicznych — ujemna wartość głośności ustawia przesunięcie kształtu fali o 180°. Tworząc dźwięk, który rzeczywiście chcesz zmiksować w ten sposób, możesz stworzyć niezłe, awangardowe efekty, ale w większości przypadków będzie to coś, czego nie będziesz chciał otrzymać.

```
soundAtrans = {ll:200, lr:0, rr:200, rl:0};
```

Jest to opcja przeznaczona dla tych, którzy lubią hałas. Takie ustawienie da w efekcie potężny przesterowany basowy dźwięk. Zanim jednak spróbujesz to zrobić, sprawdź wcześniej czy głośność na wzmacniaczu nie jest ustawiona zbyt wysoko. Jeśli w jakimś miejscu zastosujesz ten efekt bez jakiegokolwiek uprzedzenia, podejrzewam, że przysporzy ci to kilku nowych przyjaciół — szczególnie wśród tych, którzy przeglądają sobie strony ze słuchawkami na uszach, albo bawią się Siecią podczas cichej i spokojnej przerwy na lunch.

Zastosowanie przesterowanego dźwięku jest dodatkową cechą, a zatem można to stosować we własnej witrynie, ale bądź ostrożny, jeśli będziesz stosował to na stronie klienta, gdyż może nie działać w następnej wersji odtwarzacza filmów Flasha. Sam lubię nieco hałasu, więc zastosowałem to na stronie, która miała pokrętło głośności z maksymalną wartością 11 (najlepsze tradycje parodii rock'n'rollowych filmów) z małym symbolem czaszki ze skrzyżowanymi piszczelami obok — jako jasne ostrzeżenie.

Na końcu spróbuj tego. Usłyszysz dźwięk tylko wtedy, jeśli będziesz słuchał za pomocą dobrze ustawionego systemu głośników stereo:

```
soundAtrans = {ll:100, lr:0, rr:-100, rl:0};
```

Daje to taki sam efekt, jaki otrzymalibyśmy podłączając system głośników krzyżując dodatni i ujemny przewód w jednym z nich. Uzyskamy wtedy dźwięk z obu głośników, który będzie się wzajemnie znosił, zamiast dodawać. Zanikną wówczas basy, ale jeśli zmienisz balans na wzmacniaczu tak, aby można było usłyszeć tylko jeden głośnik, pojawią się z powrotem. Jest to bardzo podobne do efektu „bez dźwięku", który tworzyliśmy chwilę temu, ale tym razem przesunięcie fazowe pojawia się nie w odtwarzaczu Flasha, ale w samych głośnikach. Jest to jednak efekt, którego normalnie będziesz unikał i ogólnie `ll` i `rr` powinny mieć ten sam znak.

Dynamiczne przekształcanie dźwięku

Gdy zaczniesz dynamicznie kontrolować przekształcenie dźwięku, wtedy łatwiej będzie ci zrozumieć potęgę metody `setTransform`. Takie działanie ma dwie zalety.

- Łatwo jest zdefiniować ustawiony wcześniej efekt. Gdy zdefiniujesz więcej niż jeden obiekt kontrolujący będziesz mógł natychmiast przełączać się między ustawieniami dźwięku. Na przykład poniższy skrypt definiuje dwa obiekty: `stereo` — który ustawia parametry dla stereofonicznej metody `setTransform`, oraz `mono` dla... cóż, pewnie sam wiesz, co jest dalej, prawda?

```
stereo = new Object ();
stereo = {ll:100, lr:0, rr:100, rl:0};
mono = new Object ();
mono = {ll:100, lr:100, rr:100, rl:100};
```

Gdy wprowadzisz metodę `mysound.setTransform (stereo)` lub `mysound.setTransform (mono)`, będziesz mógł łatwo przełączać się między trybem mono i stereo. Jest to ten rodzaj działania, który najprawdopodobniej będzie coraz bardziej potrzebny w przyszłości, ponieważ niektóre rodzaje sprzętu (w szczególności urządzenia bezprzewodowe) mogą nie dysponować możliwością odtwarzania dźwięku stereofonicznego. Aktualnie możesz wyposażyć witryny w kontrolę stereo i mono obok włącznika stereo za pomocą prostego zdarzenia przycisku:

```
on (release) {
   mysound.setTransform (mono);
}
```

- Łatwo można definiować *zmienne* efektów i przekształceń dźwiękowych, kontrolując je za pomocą języka ActionScript. Aby to pokazać, przeanalizujemy prosty przykład.

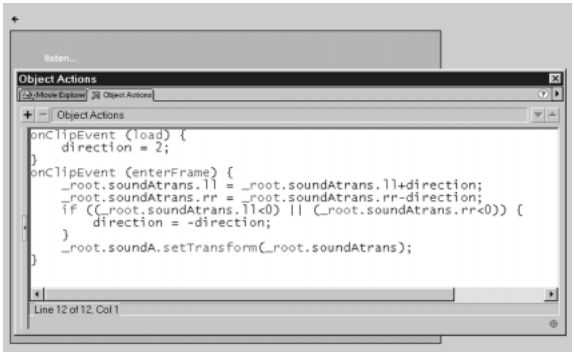
Przykład znajduje się w pliku o nazwie *setTransform2.fla*. Film zawiera klip, który ciągle zmienia dźwięk stereo między prawym i lewym głośnikiem (bez zmiany głośności dźwięku). Jest to bardzo podobne do przykładu znajdującego się w pliku *setTransform.fla* — kod w ujęciu pierwszym (w warstwie *actions*) jest niemal taki sam:

```
soundA = new Sound (_root);
soundA.attachSound ("loop");
soundA.start (0, 999);

soundAtrans = new Object ();
soundAtrans = {ll:50, lr:0, rr:50, rl:0};
soundA.setTransform (soundAtrans);
```

Jedyną różnicą jest to, że tym razem w trzeciej linii dźwięk zapętlony jest 999 razy, czyli wystarczająco długo, aby usłyszeć efekt. Ustawiony został również obiekt kontrolujący `soundAtrans` tak, aby podawał 50% głośności obydwu głośnikom.

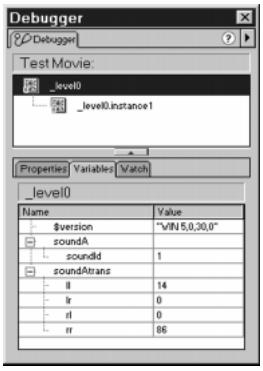
W warstwie *control* znajduje się pusty klip filmowy, który został nazwany *soundcontrol* (możesz dostrzec go ponad obszarem obrazu w lewym górnym rogu). Chociaż jest pusty, ma jednak dołączone pewne detektory zdarzeń. Wybierz ten klip i skorzystaj z edytora skryptów, aby zobaczyć kod.

Gdy będzie miało miejsce zdarzenie `load`, ustawimy zmienną nazwaną `direction` na 2. Jest to współczynnik, z jakim będzie miała miejsce zmiana balansu — 2% na ujęcie (zdarzenie `load` pojawi się po raz pierwszy, gdy Flash „zobaczy" nasz klip filmowy na listwie czasowej; w tym przypadku jest to ujęcie pierwsze, więc reprezentuje to naszą inicjację).



Następny detektor zdarzenia w tym klipie to `enterFrame`. Pojawia się w każdym ujęciu filmu (nawet jeśli ma on tylko jedno ujęcie, jak w tym przypadku), więc dołączyłem główny kod do tego detektora. Dodaje on po prostu zmienną `direction` do właściwości `ll` naszego obiektu kontrolującego i odejmuje tę samą wartość od właściwości `rr`. Sprawia to wrażenie, że balans przesuwa się w stronę lewego kanału. Wyrażenie warunkowe `if` sprawdza to, czy `ll` i `rr` osiągają zero, co oznacza, że balans przeszedł całą drogę do końca. Gdy ma to miejsce, zmienia się znak zmiennej `direction` i balans będzie przemieszczał się w stronę prawego głośnika. Będzie się to powtarzało w nieskończoność (no, dobrze — 999 razy, jeśli wytrzymasz tak długo!). Jeśli wywołasz *Debugger* zobaczysz, jak poziom zmienia się wizualnie.

Jak zauważyłem we wprowadzeniu, ten poziom dźwięku jest różny od tego, który słyszeliśmy za pomocą prostszych metod `setPan` i `setVolume` (w przykładzie *sound.fla*), ponieważ pozwala nam zmieniać obraz stereo bez zmian całkowitej głośności. Ten system nie jest stosowany w domowym sprzęcie hi-fi, ale jest używany przy miksowaniu dźwięku, ponieważ pozwala wprowadzać zmiany w obrazie stereo niezależnie od głośności stereo.



Zaawansowane przydzielanie dźwięku

Poprawne przydzielanie poleceń dźwiękowych może być skomplikowanym zadaniem. Musisz mieć tego świadomość definiując *globalną* kontrolę dźwięku (która działa na wszystkie dźwięki w filmie), kontrolę *lokalną* (działającą na jedną listwę czasową) oraz

pojedynczą (która będzie działała na pojedynczy połączony plik dźwiękowy). Czymś, co może cię dobić, jest fakt, że jeśli nie zdefiniujesz poprawnie swoich obiektów typu Sound, możesz sądzić, że masz pewien typ kontroli, podczas gdy w rzeczywistości go nie masz.

Kontrola globalna i lokalna

Powiedziałem to już wcześniej, ale powtórzę jeszcze raz: jeśli zdefiniujesz obiekt typu Sound na głównej listwie czasowej bez celu, każda akcja, którą zastosujesz na obiekcie, będzie wpływała na wszystkie dźwięki w filmie. Może to mylić, ponieważ pojawia się nawet wtedy, gdy sądzisz, że dostarczyłeś wystarczająco dużo informacji, aby wskazać określony obiekt typu Sound:

```
soundA = new Sound ();
soundA.stop ();
```

Jeśli dołączysz te akcje do głównej listwy czasowej, będą zatrzymywać *wszystkie* dźwięki, ponieważ *nie mają one zdefiniowanego celu*. W rzeczywistości nawet jeśli zdefiniujesz obiekt typu Sound na głównej listwie czasowej, który jest dołączony do pojedynczego dźwięku za pomocą współdzielenia i zrobisz to na przykład tak:

```
soundA = new Sound ("linkage_ID");
soundA.stop ();
```

...metoda stop nadal będzie działała globalnie, co będzie jeszcze bardziej mylące. Jak pamiętamy z analizy przykładu z kosmicznymi najeźdźcami, jest tylko jedna możliwość zatrzymania odtwarzania dźwięku, który jest dołączony za pomocą współdzielenia bez zatrzymywania wszystkiego innego na tej samej listwie czasowej. W poniższym skrypcie instrukcja stop będzie działała tylko na drugi dźwięk:

```
soundA = new Sound ();
soundA.attachSound ("acid");
soundA.start ("acid");

soundB = new Sound ();
soundB.attachSound ("bassline");
soundB.start ("bassline");
soundB.stop ("bassline");
```

Wywołanie polecenia stop musi odnosić się do nazwy współdzielenia, więc — jako logiczne następstwo tego faktu — metody, które podczas wywołania nie pozwalają określić tej nazwy, zawsze będą stosowały swoje akcje do całej listwy czasowej. Nie jest to nigdzie opisane, więc warto to zapamiętać, zwłaszcza, jeśli rozpatrujesz te metody, które *nie* pozwolą ci zdefiniować nazw współdzielenia zawierających:

```
Sound.setPan
Sound.setVolume
Sound.setTransform
```

Jak widzisz, lista ta zawiera wszystkie metody, których używaliśmy do tej pory i można z tego wyciągnąć jasny wniosek: nie możesz osobno kontrolować wielu dźwięków zdefiniowanych za pomocą współdzielenia, jeśli wszystkie one będą znajdywały się na tej samej listwie czasowej. Dopóki nie będziesz chciał po prostu rozpocząć i zatrzymać dźwięku, nie używaj współdzielenia do określania swoich obiektów typu Sound, ponieważ inne instrukcje wymienione powyżej dadzą w efekcie niepożądane rezultaty.

Dołączanie obiektów typu Sound do celów

Aby zdefiniować rzeczywistą lokalną kontrolę, musisz zdefiniować cel, tak jak pokazano to poniżej (ze ścieżki `_root`):

```
soundA = new Sound (myMovieClip);
soundA.stop ();
```

W tym przypadku obiekt `soundA` jest dołączony do **celu**, którym w tym przypadku jest klon klipu filmowego nazwany `myMovieClip`. Wywołanie instrukcji `stop` będzie obecnie zatrzymywało odtwarzanie tych dźwięków, które dołączone są do listwy czasowej klipu `myMovieClip`. Jeśli twój klip jest głęboko zagnieżdżony, możesz określić dowolną ścieżkę w miejscu *myMovieClip*.

Możesz zrobić to samo za pomocą zdefiniowania `soundA` w `myMovieClip`. Jest to świetne rozwiązanie, ale ponieważ definicje twoich obiektów są rozłożone w całym pliku FLA, nie będziesz w stanie ich zsynchronizować. Jeśli synchronizacja jest (czy będzie) wymagana, powinieneś zdefiniować dźwięki w jednym miejscu (nawet jeśli będą się łączyły z osobnymi klipami).

Zaawansowany dźwięk to skomplikowana sprawa we Flashu, ale w moim ostatnim przykładzie w tym rozdziale zobaczysz w pełni działający zaawansowany interfejs kontroli dźwięku. Demonstruje on również to, jak można definiować naprawdę oddzielne obiekty typu `Sound` i kontrolować każdy z nich lokalnie. Dodamy do tego modułowy, oparty na obiektach typu `Smart Clip` interfejs. Baw się dobrze i postaraj się, aby Sieć stała się głośniejszym miejscem!

Stół mikserski we Flashu

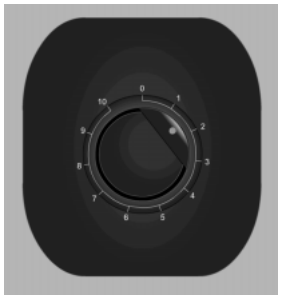
Większość nagrań realizowana jest na kilku ścieżkach (zwykle każda ścieżka odpowiada jednemu instrumentowi, czy rodzajowi dźwięku), które są zestawione za pomocą stołu mikserskiego, który pozwala na składanie dźwięków po to, aby stworzyć z nich kompozycje. Aby zrobić to skutecznie, musisz umieć wykonać takie zadania, jak:

- zmiana głośności każdej ścieżki;
- zmiana balansu każdej ścieżki;
- wyciszenie każdej ścieżki lub natychmiastowe ich przełączanie.

Są to obiekty sterujące, które będziemy mieli na naszym stole mikserskim. Rozsądnym posunięciem jest uzyskanie na początku kontroli nad jedną ścieżką dźwięku, a następnie przekształcenie tego w kontrolę modułową, a później — rozszerzenie na wszystkie ścieżki.

Interfejs użytkownika

Zanim przejdziemy do rzeczywistego zarządzania dźwiękiem, przyjrzyjmy się temu, jak działają potencjometry. Jest to przykład typowego pokrętła stworzonego w pliku *dial.fla*, które może wydawać ci się znajome.



To pokrętło jest klipem filmowym zawierającym okrągły potencjometr, tak jak pokazano to na rysunku. Znajduje się tam również kropka i skośne ścięcie, których celem jest zaznaczenie tego, gdzie pokrętło jest ustawione (jest to aktualna tendencja dominująca w przypadku pokręteł urządzeń hi-fi — coś, co również pasuje do wybranego przeze mnie schematu kolorystycznego). We Flashu jednak kropka i ścięcie mają dodatkowe zastosowanie — sugerują, iż jeśli chcesz zmienić położenie pokrętła, kursor musi znaleźć się obok. Może to brzmieć nieco nieprawdopodobnie, gdy czytasz to na tej stronie, ale testowałem to wielokrotnie i wszyscy uczestnicy eksperymentu (wcześniej nie poinformowani, że biorą w nim udział) poradzili sobie bez żadnych problemów, gdy spotykali taki potencjometr na ekranie.

Tak czy owak — nad ścięciem jest niewidoczny przycisk. Znajduje się on w klipie *Dial* i zawiera następujący skrypt:

```
on (press) {
   buttonPress = true;
}
on (release, releaseOutside) {
   buttonPress = false;
}
```

Skrypt ten po prostu ustawia znacznik nazwany `buttonPress` na wartość `true` albo `false` (zależnie od tego, czy kursor znajduje się nad przyciskiem i czy został przyciśnięty i przytrzymany klawisz myszy) — `buttonPress` pozostaje ustawiony na `true`, dopóki użytkownik nie zwolni klawisza, i pozostaje równy tej wartości, nawet jeśli kursor przesunie się poza obszar przycisku. Ten skrypt zasadniczo sprawdza, czy przycisk został wciśnięty i przytrzymany.

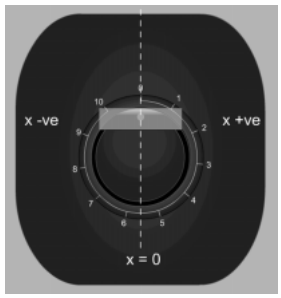
Klon klipu *Dial* ma załączony poniższy skrypt:

```
onClipEvent (load) {
   angle = 0;
   buttonPress = false;
}

onClipEvent (enterFrame) {
   if (buttonPress) {
      if (_xmouse < 0) {
         angle -= 4;
         if (angle < 0) {
            angle = 0;
         }
      }
```

```
        if (_xmouse > 0) {
           angle += 4;
           if (angle > 324) {
              angle = 324;
           }
        }
     }
     _rotation = angle;
  }
```

Detektor zdarzenia `load` inicjuje dwie zmienne: `buttonPress`, o czym pisałem nieco wcześniej i `angle` (jest to aktualny kąt, o jaki obrócony zostaje potencjometr). Początkowo jest ustawiony na zero stopni.

Detektor zdarzenia `enterFrame` wykonuje główne zadanie — odpowiada za obrót pokrętła. Jeśli przyjrzymy się przyciśnięciu i przytrzymaniu nad niewidocznym przyciskiem (za pomocą `buttonPress`), zauważymy po której stronie zaznaczonej na rysunku linii znajduje się wskaźnik myszy. Wtedy możemy obrócić potencjometr w tym kierunku.



Jeśli znajdujemy się po lewej stronie linii (`_xmouse < 0`), zmniejszamy kąt, jeśli po prawej stronie — zwiększamy go (aby stworzyć linię przechodzącą przez środek przycisku, punkt odniesienia filmu musi znajdować się na linii środkowej pokrętła, a najlepiej jest, gdy przechodzi ona przez kropkę znajdującą się na przycisku). Pokrętło przestanie się obracać, jeśli osiągniemy dwa punkty końcowe (w tym przypadku 0 lub 324º).

Jeśli nadal nie jesteś całkiem pewny, jak to będzie działać, pamiętaj, że linia środkowa obraca się razem z potencjometrem. W gruncie rzeczy w tym kodzie jest zapisane „obracaj pokrętło, dopóki linia środkowa nie dojdzie do aktualnego położenia myszy".

Aby zastosować ten potencjometr w swoim własnym filmie, musisz przeskalować kąt do takiego zakresu, jaki wskażesz. Jeśli chcesz mieć na przykład zakres między 0 a 10 (według krawędzi skośnej pokrętła), powinieneś stworzyć inną zmienną (na przykład — `range`) i dodać poniższą linię (natychmiast po linii zawierającej `_rotation = angle`):

```
range = angle /3.24;
```

Jeśli nazwiesz klon pokrętła `mydial` i umieścisz go na głównej listwie czasowej, możesz uzyskać dostęp do `range` z dowolnego miejsca w filmie za pomocą `_root.mydial.range`.

Gdy będziesz eksperymentował z przykładem *dial.fla*, zauważysz, że pokrętło nie porusza się równolegle do aktualnego położenia kursora, ale będzie wykonywało stały obrót, dopóki nie osiągnie właściwego położenia. Zrobiłem to w ten sposób, ponieważ chciałem

odwzorować podobny proces zachodzący w profesjonalnym sprzęcie muzycznym, w którym pokrętło jest spowalniane, aby chronić użytkownika przed nagłą zmianą i pozwala na zrobienie łagodnych przejść wyciszających. Pokrętło również odrobinę błyska, gdy dojdzie do położenia kursora, ale ten efekt jest wystarczająco mały, aby nie wpływał później na mikser dźwięku.

Problemy, problemy

Nasze końcowe zadanie w tym przypadku wymaga rozwiązania kilku pojawiających się problemów i najlepiej będzie, gdy stawimy im czoła już teraz. Jak będziesz mógł się przekonać, niektóre z nich nie są łatwe do rozwiązania.

1. Aby zsynchronizować dźwięk do poziomu dokładności, którego będziemy wymagać (po to, abyśmy mogli mieszać razem dźwięki), musimy zdefiniować wszystkie obiekty typu `Sound` na jednej listwie czasowej. Możemy dołączać je do innych listw za pomocą celów, ale definicje muszą znajdować się w tym samym ujęciu na tej samej listwie.
2. Flash może poradzić sobie z ośmioma ścieżkami w tym samym czasie. Najlepszym rozwiązaniem dla kontroli kolejnych ścieżek jest doprowadzenie jej do formy modułowej, co pozwala poradzić sobie z kilkoma pokrętłami. Może to być skomplikowane, ponieważ każde pokrętło ma dostęp do innego obiektu typu `Sound` i musimy umieć robić to szybko, aby zapewnić właściwą kolejność. Zastosowanie połączeń liczbowych albo ciągów znaków do rozróżnienia nazwy docelowych obiektów typu `Sound` może być nieco za wolne (korzystanie z ciągów znaków ogólnie we Flashu 5 jest wolne), a zatem musimy być ostrożni.
3. Aby stworzyć wiarygodny stół mikserski, musimy umieć wycinać i dodawać dźwięki natychmiast (nie zaś kilka ujęć po tym, gdy użytkownik o to poprosi).

Zajmijmy się tym po kolei, chociaż zapewne widzisz już rozwiązanie pierwszego problemu. Aby synchronizować dźwięki dokładnie, najlepiej jest dołączyć je wszystkie do tego samego ujęcia na głównej listwie czasowej. Dźwięki związane ze zdarzeniami nie są zsynchronizowane z listwą czasową, ponieważ zaprojektowano je dla przypadkowych efektów, które normalnie nie wymagają ścisłej synchronizacji jednego z drugim. Można jednak oszukać odtwarzacz Flasha tak, aby sądził, że ma do czynienia z dźwiękiem przesyłanym strumieniowo (który jest synchronizowany). W tym celu należy ustawić pierwszy dźwięk na listwie jako dźwięk tego typu.

Rozwiązanie drugiego problemu leży w zdefiniowaniu ścieżki zawierającej zmienną i jest rzeczywiście bardzo proste, chociaż nie jest to zapisane w instrukcji Flasha. Po dodaniu zmiennej do ścieżki w notacji kropkowej uzyskujesz w efekcie składnię podobną do tej:

```
_root[sciezka].costam
```

Przypuśćmy, że masz dwa klony klipu filmowego — `jablko` i `banan`. Oba zawierają zmienną o nazwie `costam` i chcesz ustawić jedną jej wersję na 5 (warunkowo). Aby mieć dostęp zarówno do `_root.jablko.costam`, jak i `_root.banan.costam` z tej samej instrukcji (zależnie od instrukcji warunkowej), powinieneś wpisać:

```
if (warunek) {
   sciezka = "jablko";
} else {
   sciezka = "banan";
}

_root[sciezka].costam = 5;
```

Jest to naprawdę przydatna sztuczka, ponieważ pozwala ci nie tylko tworzyć obiekty typu `Smart Clip` odnoszące się do wyjątkowych obiektów (bez odnoszenia się do połączeń ciągów znaków, co jest również wolne), ale zapewnia możliwość tworzenia bardziej ogólnych funkcji. Co więcej, pozwala ci na szybki dostęp do obiektów albo klipów filmowych, które tworzone są w międzyczasie (poprzez `duplicateMovieClip`), a to kieruje nas w stronę superszybkich *engine'ów* do gier, które są przydatne wtedy, gdy masz wielką liczbę wykrywania kolizji do wykonania.

Aby natychmiast zatrzymywać dźwięki, najlepszym rozwiązaniem nie jest zastosowanie polecenia `stop`, ale po prostu wyciszanie dźwięku poprzez ustawienie jego głośności na zero. Tak samo można poradzić sobie ze znikaniem ścieżek z całej kompozycji. Ma to również inne zalety: dźwięk pozostaje zsynchronizowany z innymi (ponieważ w rzeczywistości wcale nie został zatrzymany), a wczytywanie procesów do odtwarzacza Flasha pozostaje stałe, ponieważ dźwięki cały czas się odtwarzają (to znaczy, że dźwięki nie staną się niezsynchronizowane w zależności od obciążenia procesora).

Skończona aplikacja

Przyjrzyjmy się plikowi *mixer.fla*, który jest końcowym plikiem w tym projekcie (możesz nazwać go wersją końcową, ponieważ jasne kolory tła nareszcie odeszły w niepamięć!).

Film został tak zaprojektowany, aby mógł być oglądany na pełnym ekranie, a zatem pozbądź się okna *Bandwidth Profiler*, jeśli chcesz przeczytać opisy przycisków. Zawiera on również próbki dźwiękowe o jakości CD i publikowanie filmu będzie w słabych systemach zajmowało sporo czasu (jeśli nie masz wystarczającej ilości pamięci w Macintoshu, może nawet zawiesić ci się komputer). Jeśli będziesz testował to na komputerze, który ma z tym pewien problem, spróbuj ustawić niższe parametry *Audio Event* (*File/Publish Settings/Flash*) w okolicach 16 – 20 kbps (jakość dźwięku w filmie będzie wtedy kiepska).

Skończony film wymaga dość wysokiej prędkości odtwarzania, aby osiągnąć wymaganą synchronizację dźwięku, a zatem została ona ustawiona na 24 klatki na sekundę w oknie *Movie Properties* (*Modify/Movie*). Po raz kolejny możesz mieć kłopoty, gdy pracujesz na słabszym komputerze, gdyż zaawansowane dźwięki będą obciążać procesor, zwłaszcza gdy nie tylko korzystasz z ośmiu strumieni MP3 w tym samym czasie, ale również spodziewasz się jednoczesnego ich podłączenia w celu dynamicznego miksowania.

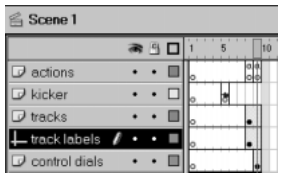
Jeśli nie widziałeś do tej pory stołu mikserskiego to:

- aby natychmiast wprowadzić ścieżkę, musisz ustawić głośność i balans na zadaną wartość i włączyć ścieżkę klikając diodę on/off;
- aby wyciszyć dźwięk, włącz go klikając diodę, a następnie ustaw poziom głośności;
- położenie dźwięku w panoramie stereo każdej ścieżki zmieniane jest za pomocą potencjometru balansu;
- najbardziej popularne utwory są odgrywane w rytmie 4/4, kompozycja będzie brzmiała nieco dziwnie, dopóki nie wprowadzisz każdej nowej zmiany w czwartym uderzeniu czwartego taktu — dopóki oczywiście nie będziesz wiedział wystarczająco dużo o twórczych sposobach łamania zasad! W przeciwnym wypadku trzeba mieć dobre wyczucie rytmu...

Jeśli ukończyłeś już tworzenie swojej kompozycji (co jest trudniejsze, niż mógłbyś przypuszczać, kiedy nie wszystkie ścieżki są prostymi rytmami — bowiem dźwięki powinny być właściwie zbalansowane w całym utworze), zobacz film.

„Kopniak"

Pierwszą rzeczą, którą trzeba zauważyć na głównej listwie czasowej jest to, jak odtwarzacz Flasha jest zmuszony do synchronizowania wszystkich obiektów typu Sound.



Warstwa *kicker* zawiera krótki dźwięk pobierany strumieniowo, który wymusza na innych dźwiękach w filmie odtwarzanie się w tym samym czasie. Aby usłyszeć, co może się stać bez niego, usuń warstwę, a następnie spróbuj wytrzymać kakofonię, którą uzyskasz!

Kod

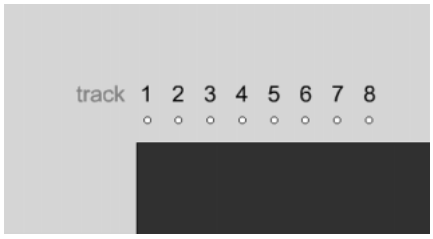
Następnym przystankiem w naszej wycieczce jest warstwa *actions*. Obiekty typu `Sound` zdefiniowane są w ujęciu ósmym:

```
// Ustawienie obiektu kontrolującego dźwięk
mute = new Object ();
mute = {ll:0, lr:0, rr:0, rl:0};

// Percussion1 ścieżka 1
perc1 = new Sound (track1);
perc1.setTransform (mute);

// Percussion2 ścieżka 2
perc2 = new Sound (track2);
perc2.setTransform (mute);

// Bass ścieżka 3
bass = new Sound (track3);
bass.setTransform (mute);

// Main melody ścieżka 4
mel = new Sound (track4);
mel.setTransform (mute);

// Ambient strings ścieżka 5
strings = new Sound (track5);
strings.setTransform (mute);

// Acid bleep ścieżka 6
acid = new Sound (track6);
acid.setTransform (mute);

// Vocal1 ścieżka 7
voc1 = new Sound (track7);
voc1.setTransform (mute);

// Vocal2 ścieżka 8
voc2 = new Sound (track8);
voc2.setTransform (mute);

// Zauważ, że maksymalną liczbą ścieżek jest 8.
// Można dodawać ich więcej, ale tylko osiem będzie
// odtwarzanych w tym samym czasie.
```

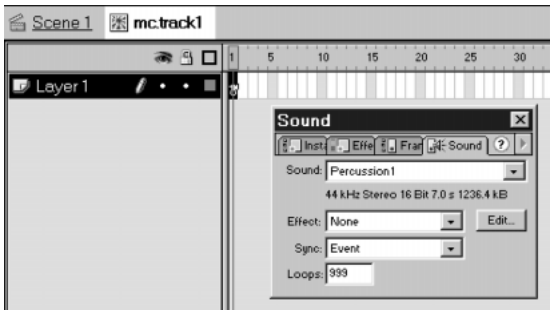
Zaczynamy od zdefiniowania obiektu typu `Sound` nazwanego `mute`. Zawiera on wartości odpowiadające za wyciszenie na początku każdego dźwięku. W chwili, gdy dźwięk się zaczyna, zostają one wprowadzone za pomocą wywołania metody `setTransform` (bez tego obiektu usłyszelibyśmy dźwięk inicjujący).

Dźwięki zdefiniowane są wraz z celami (od `track1` do `track8`). To zapewnia nam, że każdy dźwięk będzie kontrolowany niezależnie, a akcje `setTransform`, które stosujemy w całym filmie, nie będą oddziaływały na obiekty `perc1` do `voc2`, ale wpłyną na listwy czasowe `track1` do `track8`. Jest to jedyna możliwość by zapewnić sobie indywidualną kontrolę za pomocą metod `setTransform`, `setVolume` i `setPan`.

Klipy filmowe `track1` do `track8` umieszczone są po prawej stronie na górze obrazu i pojawiają się w ujęciu ósmym warstwy `tracks`.



Statyczne pole tekstowe zawierające napis „track 1 2 3 4 5 6 7 8” umieszczone jest na warstwie ścieżki nazwanej *track labels* po to, aby było wiadomo, który klip filmowy jest który. Każdy z nich zawiera plik dźwiękowy, który dołączony jest dokładnie do pierwszego ujęcia na swojej listwie czasowej za pomocą panelu `Sound`.

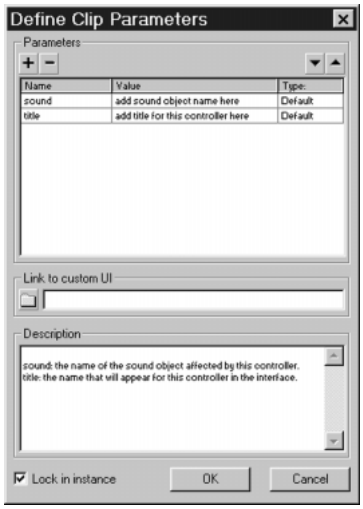


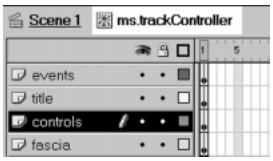
Możesz w tym miejscu zastanawiać się: „to jest Flash 5, więc dlaczego nie można dołączyć obiektu typu `Sound`, zamiast robić to wszystko?”. Cóż, jest to dobry pomysł, ale nie działa. Jak tylko rozpoczynasz definiowanie dźwięku poza główną listwą czasową, tracisz synchronizację, którą osiągnęliśmy za pomocą naszego „kopniaka”. Innymi słowy — tylko na głównej listwie czasowej możesz uzyskać synchronizację. Jeśli spróbujesz zastosować „kopniaka” na innej listwie czasowej, dźwięki nie będą zsynchronizowane z główną listwą, ale z ich własnym dźwiękiem wymuszającym synchronizację.

Ujęcie dziewiąte zawiera akcję `stop` w warstwie *actions* i możemy zobaczyć, że w tym ujęciu pojawia się nasz interfejs w warstwie *control dials*. Każda para potencjometrów, którą możesz ujrzeć, jest klonem obiektu kontrolującego ścieżki typu `Smart Clip` nazwanego *ms.trackController*. Ma on dwa parametry.

Nazwa `sound` jest nazwą obiektu typu `Sound`, który będziemy kontrolować, a `title` jest etykietą pojawiającą się w każdym obiekcie kontrolującym ścieżki w interfejsie użytkownika. Przyglądając się *ms.trackController* możesz zobaczyć następującą listwę czasową.

Dynamiczne pole tekstowe, które jest dołączone do parametru `title`, znajduje się na warstwie *title*, a potencjometry — na warstwie *controls*. Kod, który odpowiada za zachowanie obiektów sterujących, dołączony jest do klipu filmowego znajdującego się na warstwie *events*.





Potencjometry

Potencjometry są niemal dokładnie takie same, jak te, z którymi spotkaliśmy się wcześniej w przykładzie *dial.fla*. Następujący skrypt dołączony jest do pokrętła balansu:

```
onClipEvent (load) {
   angle = 0;
   buttonPress = false;
   pan = 0;
}

onClipEvent (enterFrame) {
   if (buttonPress) {
      if (_xmouse<0) {
         angle -= 4;
         if (angle<-144) {
            angle = -144;
         }
      }
      if (_xmouse>0) {
         angle += 4;
         if (angle>144) {
            angle = 144;
         }
      }
      _rotation = angle;
      pan = Math.round(angle/2.88);
   }
}
```

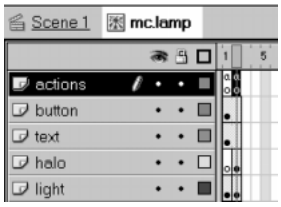
Daje nam to wartość `pan` (w ostatniej linii), która znajduje się między –50 a 50, za czego będziemy później korzystać wywołując `setTransform`. Jak można się spodziewać, potencjometr głośności ma dołączony niemal identyczny skrypt.

```
onClipEvent (load) {
   angle = 0;
   buttonPress = false;
   volume = 0;
}

onClipEvent (enterFrame) {
   if (buttonPress) {
      if (_xmouse<0) {
         angle -= 4;
         if (angle<0) {
            angle = 0;
         }
      }
      if (_xmouse>0) {
         angle += 4;
         if (angle>324) {
            angle = 324;
         }
      }
      _rotation = angle;
      volume = angle/324;
   }
}
```

Daje to wartość `volume` między 0 a 1.

Dioda LED

Warstwa *controls* kryje w sobie jeszcze jedną zmyślną sztuczkę — diodę LED. Przyjrzyjmy się klipowi filmowemu — *mc.lamp*, którego listwa czasowa pokazana jest na rysunku.



Klip przełącza się między stanem włączonym i wyłączonym. Aby stworzyć coś podobnego we Flashu, musisz mieć dwa ujęcia (do obu musi być dołączona instrukcja `stop`). Tak jest i w tym przypadku w warstwie *actions*.

Warstwa *button* zawiera pojedynczy niewidoczny przycisk znajdujący się nad obrazem diody LED. Dołączony jest do niego poniższy skrypt:

```
on (press) {
   gotoAndPlay(1)
}
```

Jeśli znajdujemy się aktualnie w ujęciu pierwszym i zatrzymamy się, przycisk spowoduje przejście do ujęcia drugiego, w którym zatrzymamy się ponownie. Jeśli będziemy w ujęciu drugim, przycisk spowoduje przejście do ujęcia pierwszego, ale akcja `stop` spowoduje zatrzymanie się w tym ujęciu. Obojętnie, w którym ujęciu będziemy się znajdowali, przycisk wymusi przejście do kolejnego.

Warstwa *actions* ma w rzeczywistości dołączoną jeszcze jedną parę instrukcji: `led = 0` (wyłączony, ujęcie pierwsze) i `led = 1` (włączony, ujęcie drugie). Zobaczymy, jak stosowana jest ta wartość, gdy przejdziemy do końcowego skryptu, który kontroluje wszystko.

Skrypt kontrolujący

Skrypt kontrolujący wszystkie obiekty typu `Sound` jest umieszczony w warstwie *events* obiektu typu `Smart Clip` *ms.trackController*. Dołączony jest do niego pusty klip filmowy nazwany *mc.dummy*, który możesz zobaczyć obok (to ta mała, biała kropka).

Gdy stosujesz zdarzenia klipów filmowych, czasami napotykasz na trudności, nie wiesz do czego dołączyć kod. Możemy dołączyć go do obiektu *ms.trackController*, ale wtedy musimy pamiętać o tym, aby dołączyć go do wszystkich ośmiu klonów. Jest to obowiązkowe, ale właśnie przez to nasz obiekt typu `Smart Clip` nie jest w pełni modułowy (wymaga od ciebie dołączenia do skryptu, więc nie jest to dłużej obiekt typu „przeciągnij mnie gdziekolwiek, a będę działał"). Również dodanie ośmiu takich samych skryptów do filmu w chwili, gdy staramy się zastosować tylko jeden, jest marnym wyjściem.

Rozwiązaniem jest natomiast dołączenie tego skryptu do zagnieżdżonego klipu filmowego, jak to zrobiliśmy z *mc.dummy*. Ponieważ nasz skrypt jest dołączony wewnątrz *ms.trackController*, wszystkie problemy znikają. Jest to jeszcze jedna sztuczka, którą warto zapamiętać.

Wróćmy jednak do tematu. Skrypt wygląda następująco:

```
onClipEvent (load) {
   sound = _parent.sound;
   controlSound = new Object ();
   controlSound = {ll:0, lr:0, rr:0, rl:0};
}

onClipEvent (enterFrame) {
   controlSound.ll = (50 - _parent.panControl.pan) *
   ➥_parent.volControl.volume * _parent.lamp.led;
   controlSound.rr = (50 + _parent.panControl.pan) *
   ➥_parent.volControl.volume * _parent.lamp.led;
   _root[sound].setTransform (controlSound);
}
```

Jak zwykle detektor zdarzenia `load` zawiera skrypt inicjujący. Ponieważ parametr obiektu typu `Smart Clip` — `sound` należy do macierzystego klipu filmowego *ms.track Controller*, przeniosłem go do *mc.dummy*, aby nie zawracać sobie głowy ścieżkami adresowymi. Pomaga to również w usuwaniu błędów, gdyż „wyłapanie" zmiennych z obiektu macierzystego pozwala na sprawdzenie, czy są one rzeczywiście przesyłane poprawnie i czy mogą być rozpoznawane przez klip potomny. Następnie definiujemy obiekt kontrolujący. Będziemy go używali wywołując metodę `setTransform`. Powinieneś rozpoznać początkowe wartości `ll`, `lr`, `rr` i `lr` jako te, które będą wyciszały nasz dźwięk.

Detektor `enterFrame` zawiera główny skrypt, w którym właściwości obiektu kontrolującego `ll` i `rr` ustawione są bardzo podobnie:

- `(50 - _parent.panControl.pan)` nadaje naszej właściwości `ll` wartość pochodzącą z pokrętła balansu (wartość `rr` będzie się równała 50 minus wartość `ll` w związku ze zmianą znaku);
- następnie mnożymy balans razy głośność (pochodzącą z potencjometru głośności) `_parent.volControl.volume` (pamiętaj, że jest to ułamek dziesiętny z zakresu od 0 do 1), więc mnożenie balansu większego od zera razy zerowa wartość głośności dawać będzie zero, a mnożenie razy 1 — maksymalna głośność — dawać nam będzie pełen udział;
- na końcu mnożymy razy stan diody, `_parent.lamp.led`; może to być 0 (wyłączone) albo 1 (włączone); jeśli dioda LED jest wyłączona mnożymy razy 0, co daje nam zero dla `ll` i `rr`, a jeśli jest włączona, dostajemy pełen udział, co świadczy o tym, że dioda działa jak włącznik (jak mogliśmy się zresztą tego spodziewać) sygnału.

Wszystko inne zostanie zastosowane do naszych nowo obliczonych wartości `ll` i `rr` po wywołaniu metody `setTransform`. Poniżej zaprezentowana akcja ustala ścieżkę dostępu do listwy czasowej zawierającej zadany dźwięk, który następnie przekształca.

```
_root[sound].setTransform (controlSound);
```

Ten film zawiera mnóstwo sztuczek, więc pobaw się nim troszeczkę i zobacz, co jeszcze możesz odkryć. Jest również tak zaprojektowany, że łatwo możesz zmieniać zastosowane dźwięki tak, abyś mógł dodawać swoje własne kompozycje. Upewnij się jednak, że wszystkie dźwięki są dokładnie tej samej długości (za pomocą narzędzia podobnego do trackera *SoundForge*) i są tak opracowane, aby można było je zapętlić.

Zasoby dźwięku we Flashu

Ostatnimi czasy również pod względem komercyjnym zastosowanie dźwięku we Flashu staje się popularne i ma to odzwierciedlenie w ilości i różnorodności witryn. W Sieci można znaleźć wiele zasobów odpowiednich dla tych z nas, którzy nie są muzykami i chociaż większość z oferowanych dźwięków jest marnej jakości (mam nadzieję, że nie tworzono ich z taką myślą!), to prawdziwy skarbiec wspaniałych pomysłów można znaleźć w witrynie *www.flashharmonics.co.uk*.

Są to zasoby stworzone specjalnie dla projektantów pracujących we Flashu i jest tego znacznie więcej niż gdzie indziej (typowe dyskotekowe kawałki można znaleźć wszędzie). Nie są one rozpowszechniane za darmo, ale są tak tanie, że nie będziesz miał kłopotu z zakupem.

Jeśli weźmiemy pod uwagę programowanie, to dobre pliki źródłowe można znaleźć w witrynie *www.killersound.com*. Wprowadzają one „kopniaka", którego zastosowałem w tym rozdziale i w chwili, gdy piszę te słowa, dostępny jest samouczek do Flasha 4, w którym znajdują się nadal przydatne rady i wskazówki (na stronie skorzystaj z łącza do *Tech center*, aby dostać się do tych plików).

Jeśli szukasz najświeższych informacji, spróbuj zajrzeć na stronę *www.macromedia.com/support/flash*, która zwykle zawiera nowe informacje techniczne warte przeczytania (w szczególności TechNote#12046 zawierająca najnowsze łącza do stron z dźwiękami).

Na koniec — najlepsze ze wszystkich dostępnych zasobów (jeśli potrafisz je odnaleźć) znajdują się w systemach zubożałych muzyków ze Steinberg Cubase i SoundForge. Odnajdź któregoś z nich, spróbuj się zaprzyjaźnić, a następnie obiecaj, że postawisz mu piwo. W ten sposób będziesz miał dostęp do ciągłego źródła świetnych dźwięków. W moim przypadku to działa!

Kwestie sprzętowe

Na końcu powinienem wspomnieć o tym, że pewne rodzaje sprzętu mają problemy z odtwarzaczem Flasha i pewne sterowniki kart dźwiękowych w *Windows9x* będą generowały skrzypiące trzaski za każdym razem (zwłaszcza przy dużej głośności). A zatem recydywiści to:

- *Crystal Audio System (IBM Intellistation E Pro);*
- *Soundblaster Audio PCI 64D;*
- *Soundblaster Audio PCI 128D;*
- *ESS Maestro-2E (laptopy Dell Inspiron);*

Powyższy spis stworzył *Urami*, legendarny guru Flasha, który udziela się w grupie dyskusyjnej *macromedia.flash*, ale jest jeszcze kilka innych laptopów, które mają podobne problemy. Jeśli zauważysz takie zachowanie, pobierz najnowsze sterowniki, a jeśli i to nie podziała, pokrzycz na twórców sprzętu.