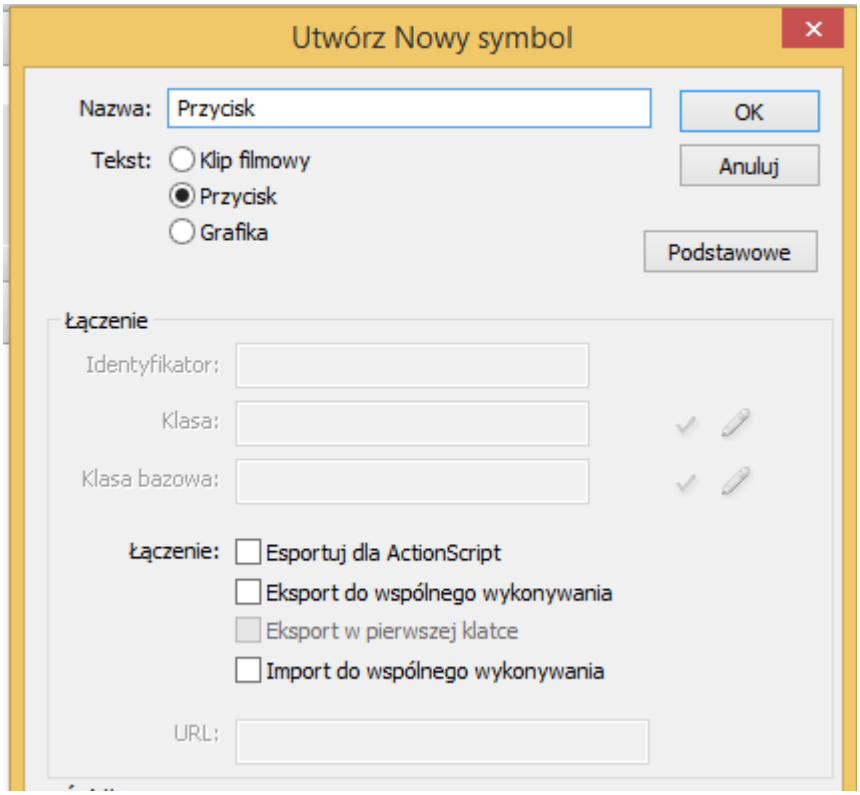
1. Tworzenie przycisków w Adobe Flash Professional

Przyciski w animacjach Flash spełniają kilka ról: mogą odnosić nas do kolejnych elementów animacji (bądź nawet do konkretnych scen), mogą aktywować/dezaktywować wskazane części animacji, włączać/wyłączać dźwięki, uruchamiać/zatrzymywać animację bądź po prostu mają pełnić rolę elementów, które po najechaniu kursorem myszy się podświetlają/zmieniają swój stan.

W celu utworzenia nowego przycisku klikamy menu Wstaw->Nowy symbol... (skrót CTRL+F8).



Bardzo ważnym jest zaznaczenie odpowiedniego typu symbolu (Przycisk). Chociaż każdy z dostępnych typów może stać się przyciskiem (może nasłuchiwać np. kliknięcia na nim myszą i zareagować na nie odpowiednio) to tylko ten typ posiada unikatowe cechy, które czynią z niego prawdziwy przycisk. Nazwę nowego symbolu podajemy dowolną (na zrzucie widoczna nazwa to po prostu Przycisk). Po utworzeniu symbolu narzędzie automatycznie przeniesie nas do jego edycji. Edycja przycisków różni się od edycji scen czy innych symboli – przyciski nie posiadają linii czasu złożonych z poszczególnych klatek. W zamian za to posiadają 4 części animacji:

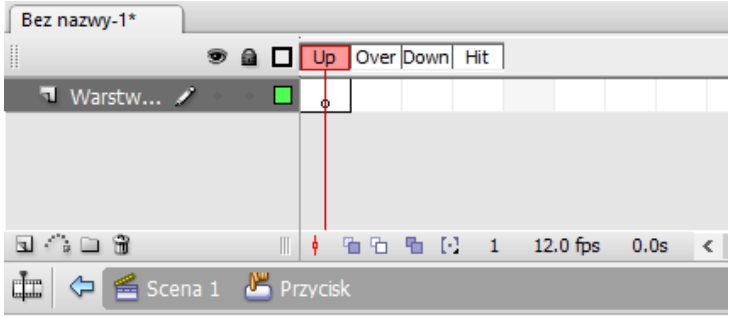
a) Up – odpowiada za wygląd przycisku w czasie gdy nie ma nad nim aktywności użytkownika (kursor myszy jest poza nim)

b) Over – aktywuje się w chwili gdy kursor znajduje się nad przyciskiem.

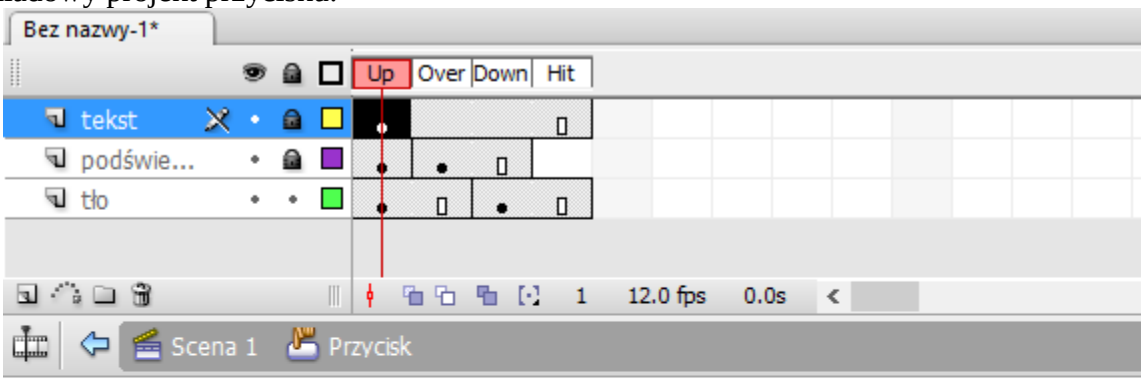
c) Down – odpowiada za stan przycisku gdy użytkownik naciśnie przycisk myszy, a kursor znajduje się nad przyciskiem

d) Hit – najczęściej niezauważalny stan. Wywoływany jedynie w chwili aktywowania przycisku (hit – uderzenie).

Odpowiednio skonfigurowana może np. nadać poświatę dla tego elementu.



Przykładowy projekt przycisku:



Proszę zauważyć, że każdy element tworzonego symbolu jest na osobnych, podpisanych warstwach (dla przypomnienia – nazwy warstw są dowolnie wyznaczane przez projektanta, powyżej to tylko przykład nazwania ich). Części przycisku, które zmieniają się przy jednym z 4 zdarzeń dla niego przeznaczonych, są oznaczone czarnymi kropkami (ramki kluczowe – takie same w pozostałych animacjach). W poszczególnych ramkach można modyfikować kolor czy kształt danego elementu przycisku. Później wystarczy po prostu zapisać swój przycisk i wypróbować go na animacji, tj. wrócić do edytowanej sceny (nacisnąć na napisie Scena 1 bądź na niebieską strzałkę – szczegółowy opis w materiale 2).

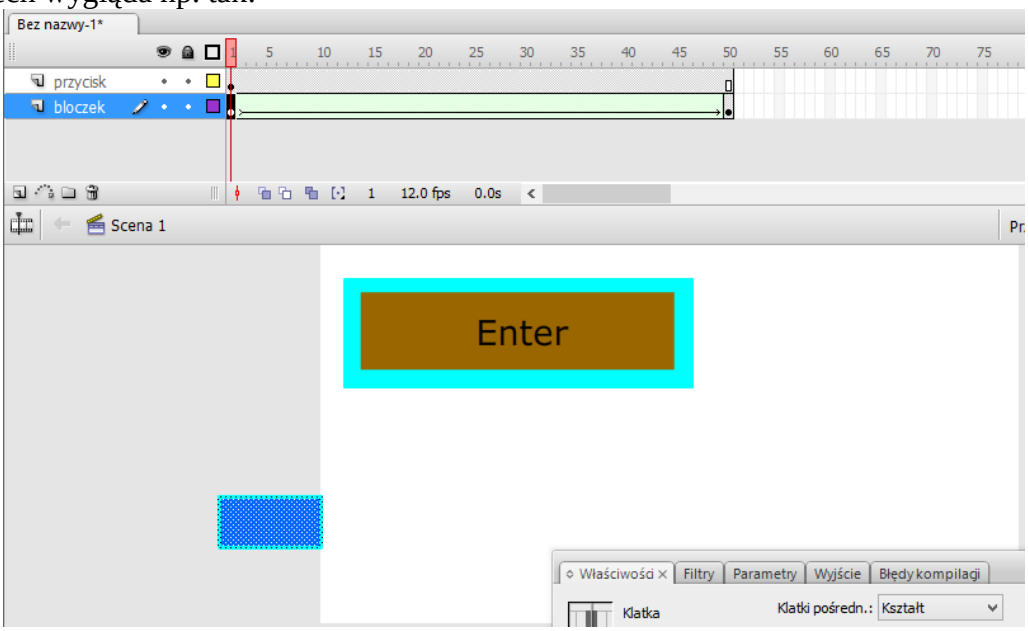
przycisk niepodświetlony

przycisk podświetlony

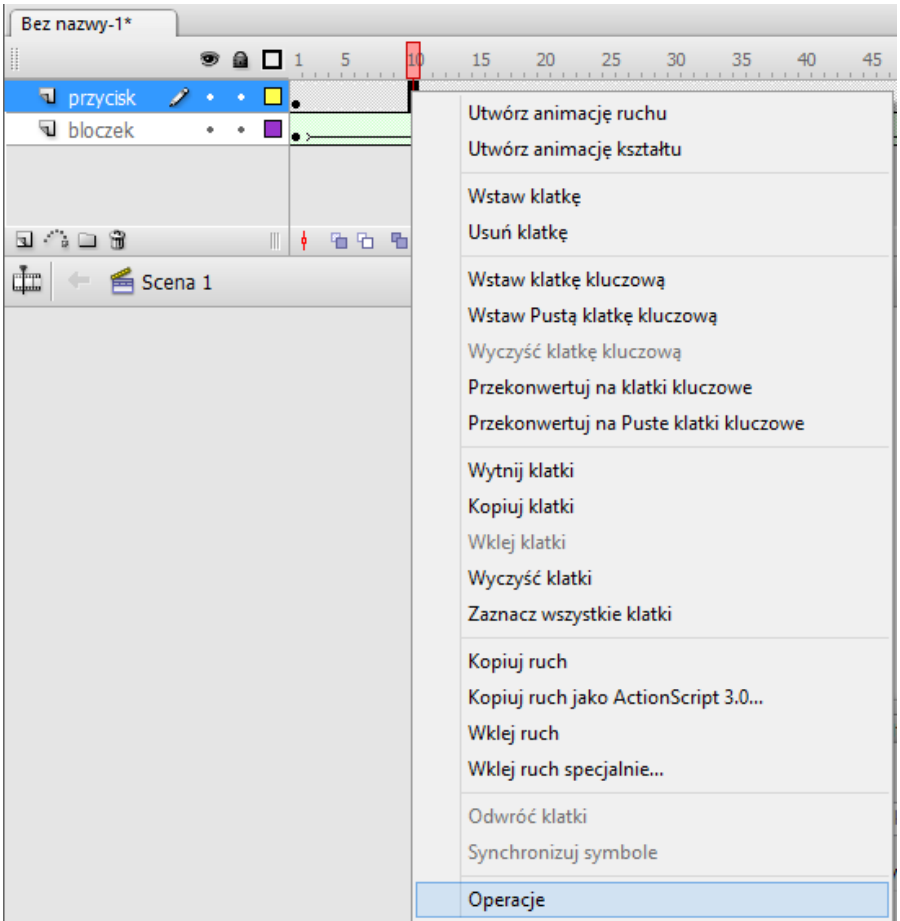
Przyciskom oraz klipom filmowym można dodatkowo nadawać filtry. Dzięki filtrom możliwe staje się np. dodanie poświaty do przycisku, cienia, rozmycia przycisku itp. Dodatkowo narzędzie umożliwia zmianę parametrów poszczególnych filtrów-efektów przycisku.

INFORMACJA: Narzędzie Adobe Flash pozwala dodawać Filtry jedynie do przycisków oraz klipów filmowych. Nie ma możliwości stosowania ich do kształtów ani grafik.

Niestety dodany przez nas przycisk nic nie będzie robił poza istnieniem w animacji. W celu dodania mu jakiejkolwiek funkcjonalności utwórzmy w animacji nową warstwę. W warstwie tej narysujmy dowolny kształt (na zrzucie poniżej dodany został prostokąt). Ustalmy dla niego pozycję początkową (np. poza animacją). Dodajmy nową klatkę kluczową dla jego warstwy (np. na 60 klatce). Upewnijmy się, że klatka ta jest nadal wybrana. W niej zmieniamy pozycję kształtu (np. tak by znajdował się na drugim końcu animacji). Teraz dodajemy animację kształtu (NIE RUCHU!). Całość niech wygląda np. tak:

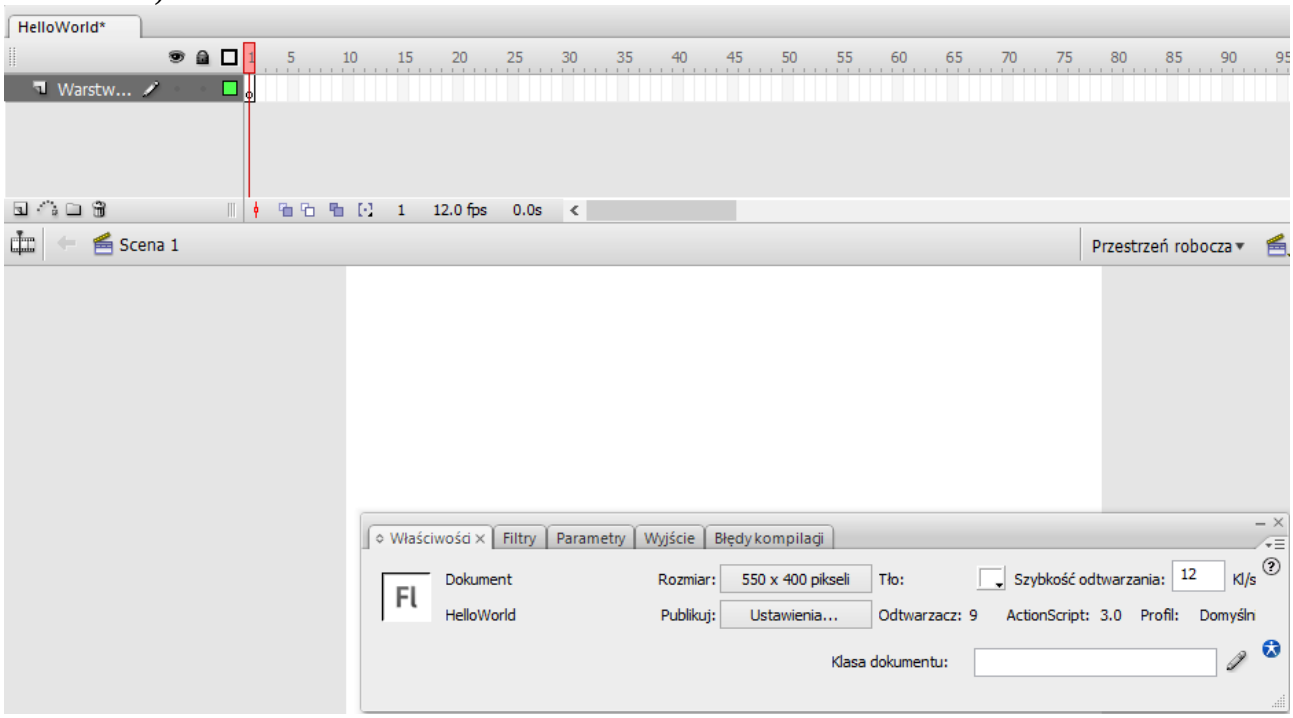


UWAGA! Proszę zwrócić uwagę na fakt, że warstwa z przyciskiem także musi mieć swoje klatki podczas animowania. Ich brak spowoduje, że przycisk zniknie ze sceny (jeżeli nie wstawimy klatki nad drugą klatką kluczową animacji).
Przetestujmy animację. Jeżeli wszystko jest poprawnie to utworzony kwadrat powinien pojawiać się z lewej do prawej strony. Teraz możemy dodać akcję do naszego przycisku. W tym celu klikamy na dowolną klatkę warstwy z naszym przyciskiem prawym przyciskiem myszy i wybieramy opcję „Operacje”.

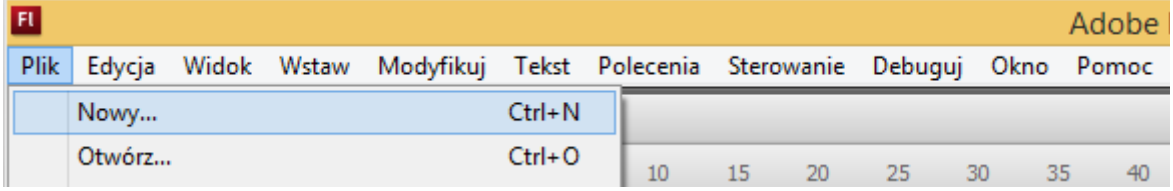


Pojawi się okno kodu ActionScript. To w nim będziemy wpisywać nasz kod dla danej klatki.

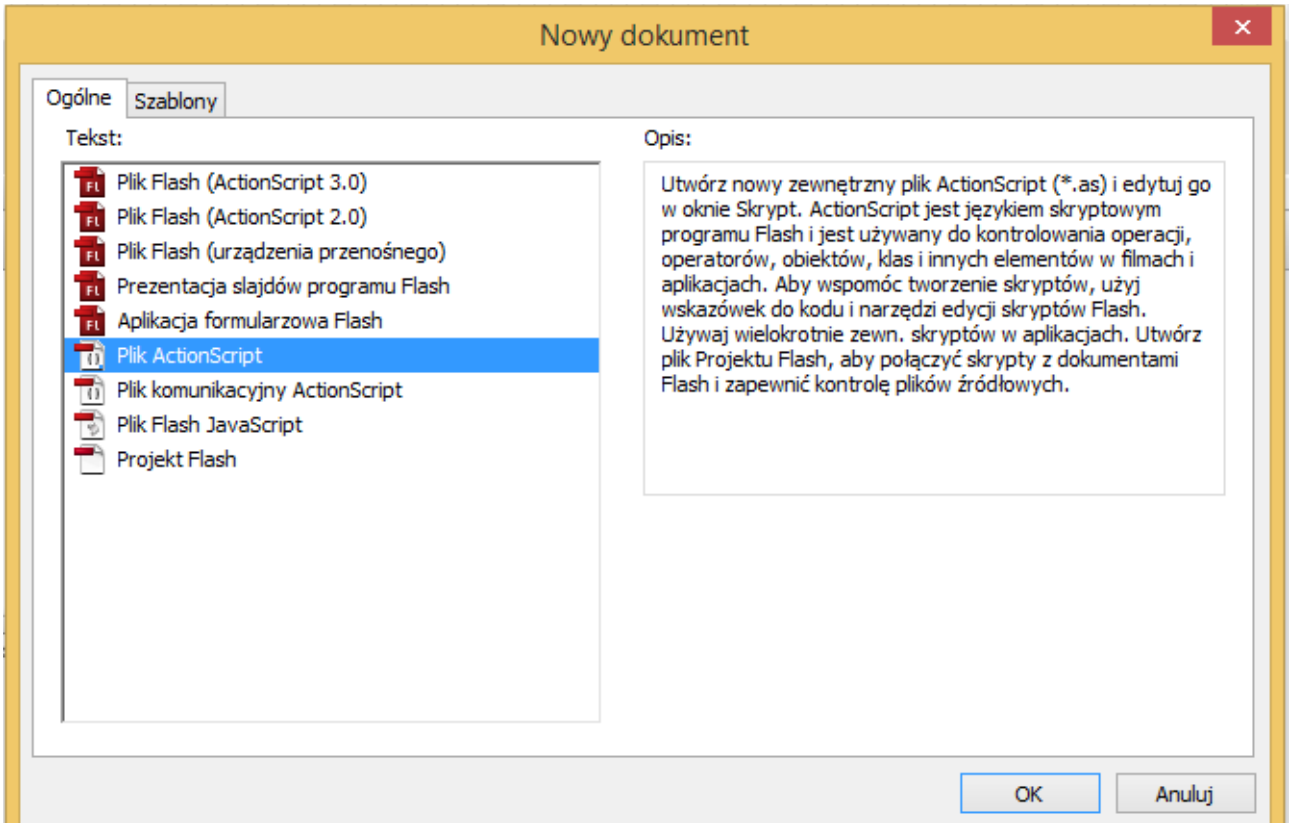
INFORMACJA: W ActionScript 3.0 wprowadzono tzw. hermetyzację (uszczelnienie) kodu. Kod ActionScript można dodawać jedynie dla poszczególnych klatek w warstwach sceny. Kod tego typu działa do czasu życia ramki kluczowej oraz jej ramek pośrednich (wchodzących w skład animacji). Aby dodać globalny kod (oddziałujący na całą animację, włączając w to każdą scenę czy symbol) należy dodać swoją definicję głównej klasy i nakreślić jej metody oraz pola (właściwości). BARDZO WAŻNE jest by plik nazywał się tak samo jak nazwa klasy (np. jeżeli stworzyliśmy klasę HelloWorld to plik powinien nosić nazwę HelloWorld.as). Poza tym plik takiej klasy zawsze musi być zapisany tam gdzie plik fla naszej animacji (ten sam katalog). Kiedy spełnimy już te wymagania możemy zastąpić klasę generyczną naszej animacji klasą stworzoną przez nas. W tym celu klikamy na animacji tak by żaden element warstwy nie został wybrany (klikamy na ikony oka + kłódki by wyłączyć widoczność wszystkich elementów warstw oraz je zablokować). Gdy w oknie Właściwości będziemy mieć wybrany Dokument z nazwą naszej animacji powinniśmy mieć także dostęp do pola Klasa dokumentu. W nim podajemy nazwę wcześniej stworzonej klasy bazowej. Wszelkie pytania narzędzia dotyczące operacji potwierdzamy. Od tego momentu tworzona animacja będzie korzystać właśnie z tej klasy (a tym samym z jej metod i pól). Pewną niedogodnością będzie potrzeba każdorazowego dołączania potrzebnych modułów. Opisany wyżej proces najlepiej zilustrują zrzuty ekranowe:
1) utworzenie nowego projektu i zapisanie go (nazwa dowolna – w przypadku ze zrzutów ekranu to HelloWorld)



2) tworzymy nowy plik (CTRL+N)



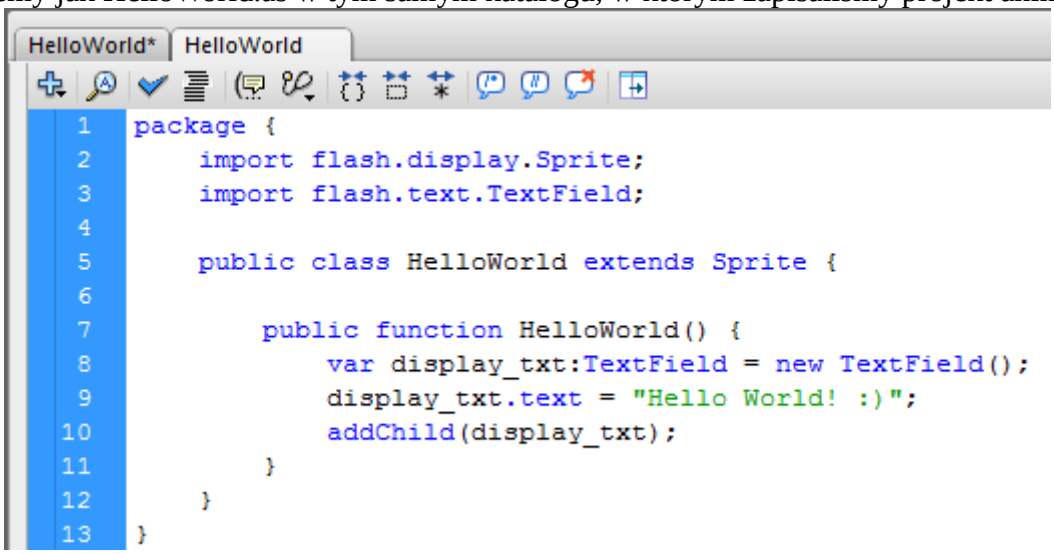
3) wybieramy plik ActionScript



4) w nowo otwartym pliku dodajemy następujący kod:

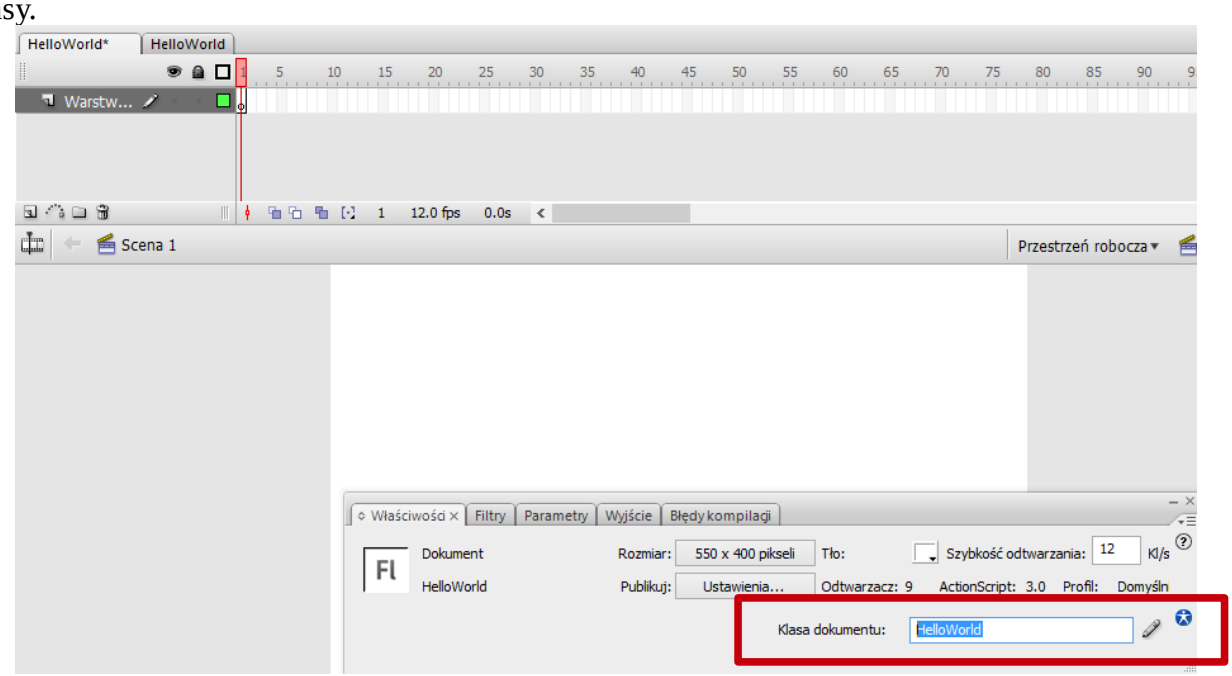
```
package {
	import flash.display.Sprite;
	import flash.text.TextField;

	public class HelloWorld extends Sprite {

		public function HelloWorld() {
			var display_txt:TextField = new TextField();
			display_txt.text = "Hello World! :)";
			addChild(display_txt);
		}
	}
}
```

i zapisujemy jak HelloWorld.as w tym samym katalogu, w którym zapisaliśmy projekt animacji

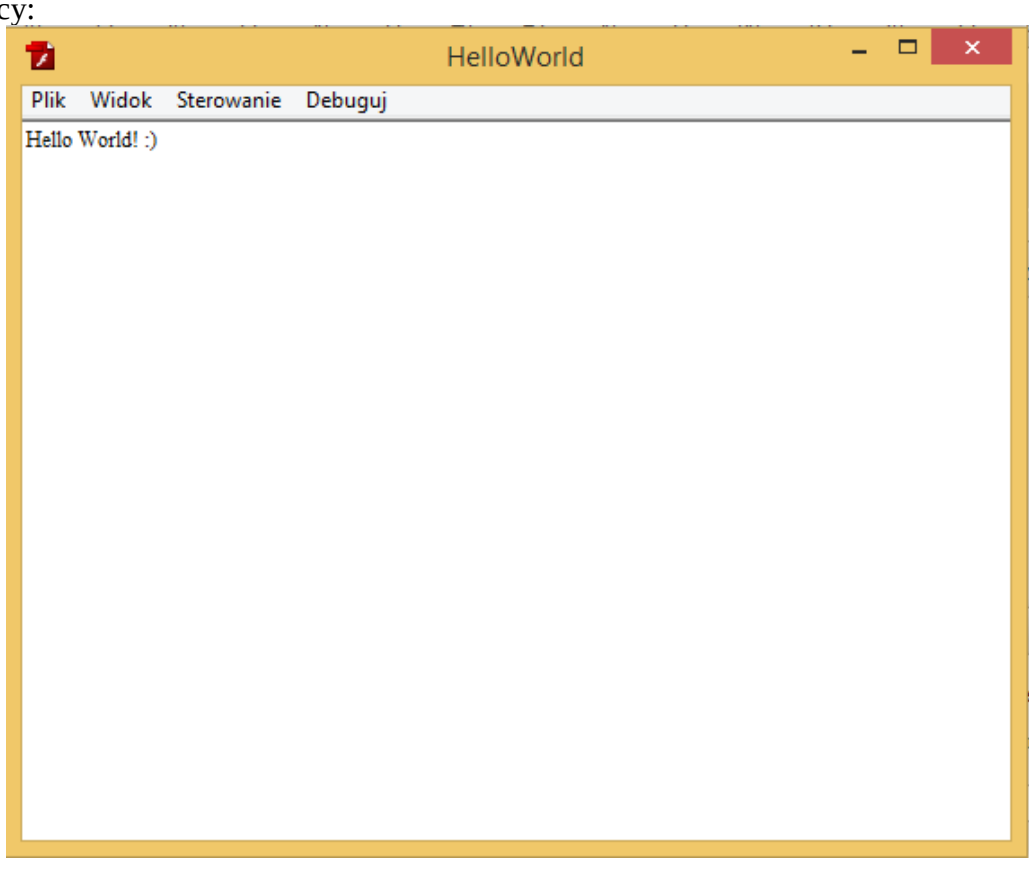


Powyższy kod tworzy konstruktor domyślny klasy (metoda HelloWorld()), w którym tworzony jest nowy obiekt klasy TextField. Następnie właściwość text otrzymuje nową wartość (może być dowolna). Na samym końcu poprzez metodę (funkcję) addChild() utworzony obiekt dodawany jest do sceny (bez tego nawet po utworzeniu nie byłby widoczny na uruchomionej animacji).

5) teraz wracamy do projektu naszej animacji. Wybieramy ponownie jej właściwości (jeżeli się nie wyświetlają to klilamy na scenę w dowolnym jej punkcie). Wpisujemy nazwę utworzonej przez nas klasy.

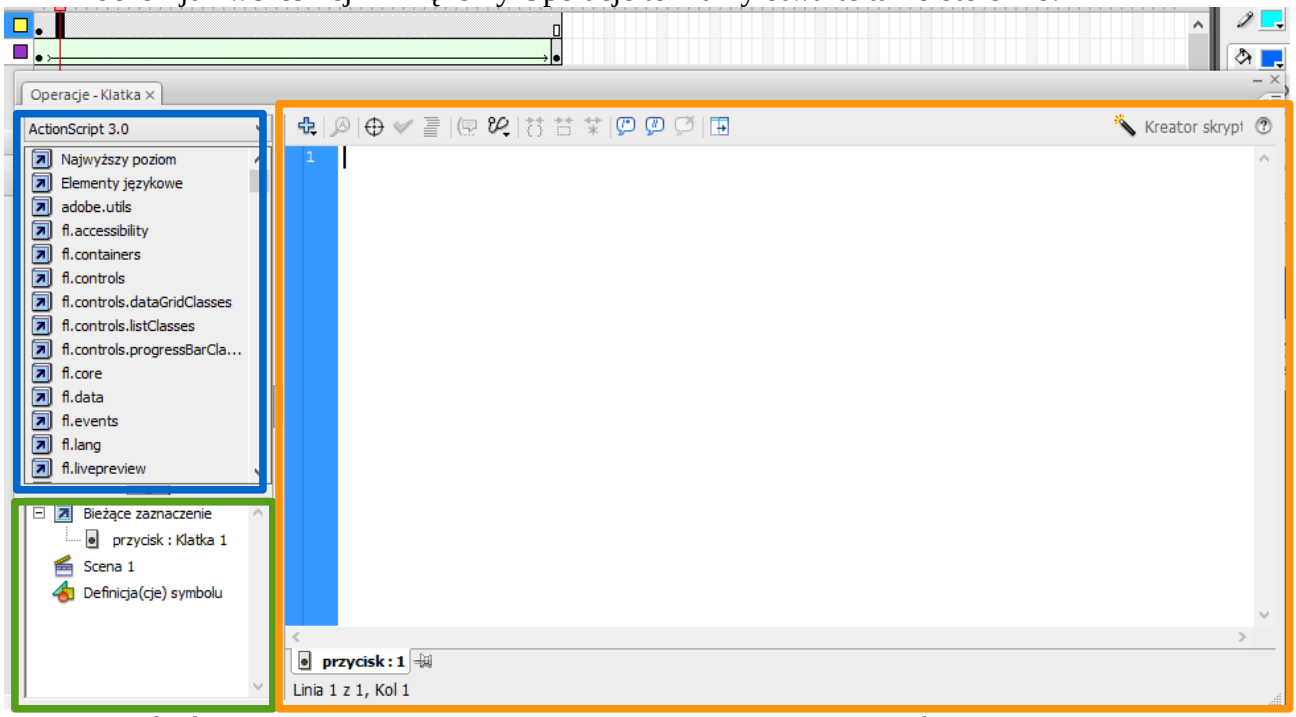


Jeżeli po naciśnięciu ołówka (Edytuj definicję klasy) obok nazwy klasy dokumentu program przeniesie nas do utworzonego pliku naszej klasy – wszystko w porządku, klasa domyślna została zastąpiona. Można przetestować działanie utworzonej w ten sposób animacji. Efekt powinien być następujący:



Wróćmy jednak do naszego przykładu. Nie musimy w nim tworzyć tego typu rozwiązania by wykorzystywać ActionScript. Nasza animacja swobodnie może wykorzystywać rozwiązania, jakie domyślnie przewidzieli twórcy narzędzia.

Jeżeli już wcześniej kliknęliśmy Operacje to mamy otwarte takie oto okno:



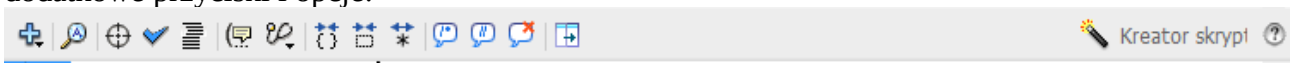
Proszę zwrócić uwagę na to, że okno Operacji złożone jest z trzech części:

a) indeks wszystkich funkcji (niebieski) – można tutaj odnaleźć wszystkie możliwe elementy języka ActionScript wraz z ich opisem. Wszystko poukładane jest wedle przynależności do elementów-klas. Niestety wadą jest brak poszukiwania po wpisanej frazie.

b) wykaz wszystkich klatek w projekcie posiadających kod ActionScript (zielony) – tutaj można przemieszczać się po już dodanych skryptach. Skrypty porozdzielane są na:

- bieżące zaznaczenie (aktualnie zaznaczone ramki przez głowicę odtwarzania bądź ramki zaznaczone na czarno przez użytkownika)
- na sceny (w tym wypadku tylko jedna scena)
- na definicje symbolu (aktualnie nie dodane do biblioteki)

c) przestrzeń kodu (pomarańczowy) – tutaj wpisujemy kod naszego skryptu. Ta część zawiera dodatkowe przyciski i opcje:

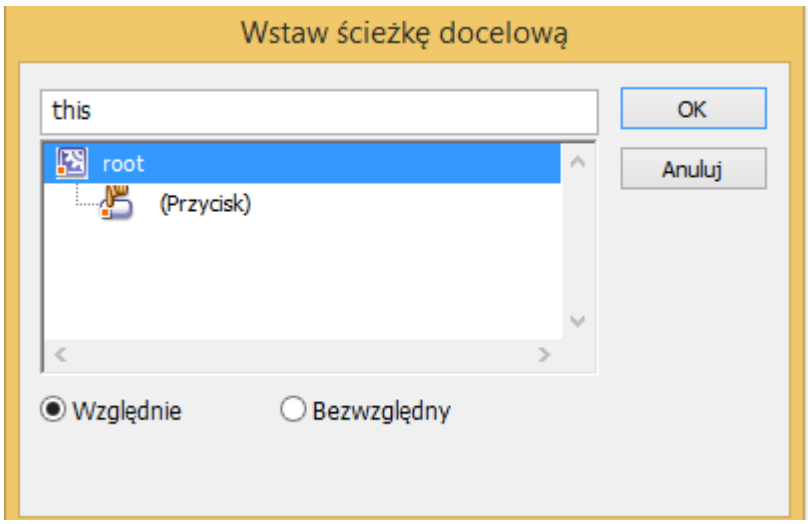


1) 2) 3) 4) 5) 6) 7) 8) 9) 10)11)12)13)14) 15) 16)

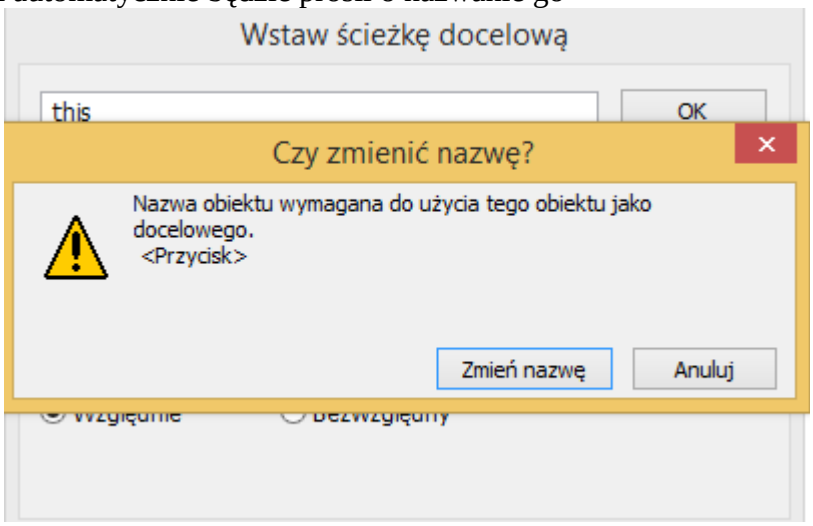
1) dodaje do skryptu jedną z predefiniowanych klas/obiektów ActionScript. Wszystkie klasy są odpowiednio posegregowane (jak w indeksie funkcji). Dodatkowo podzielono je na właściwości i metody oraz podobiekty

2) pozwala na znajdywanie i (opcjonalnie) zamienianie jednego ciągu znakowego na inny

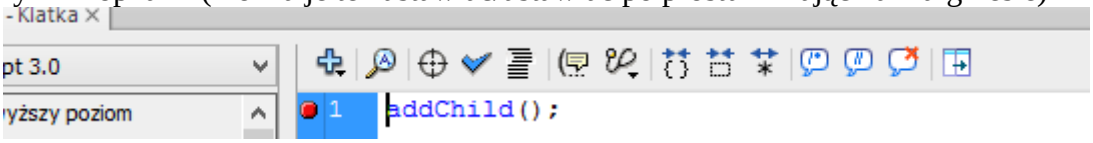
3) wstawia do skryptu ścieżkę do danego obiektu dostępnego w naszej animacji. Okno powinno zawierać wszystkie elementy w danej scenie. Przykładowo dla naszego projektu (na tę chwilę) jego zawartość będzie wyglądać następująco:



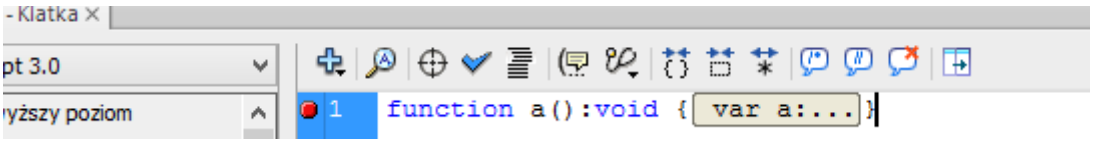
root to główna animacja, (Przycisk) to po prostu dodany do animacji przycisk (nie ma jeszcze nazwy). Opcja Względnie/Bezwzględny pozwala albo na użycie względnej nazwy (dla root zaproponowana this) bądź bezwzględnej (będzie to po prostu root). W przypadku nienazwanego przycisku program automatycznie będzie prosił o nazwanie go



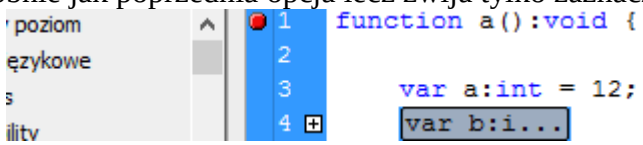
lecz na tym etapie nie będziemy mu nadawać nazwy (zrobimy to później).
WAŻNE: Proszę zauważyć, że okno ni zawiera dostępu do prostokąta. Kształty nie mogą być bezpośrednio obsługiwane przez AS!
4) sprawdza czy skrypt nie posiada błędów w składni (niestety dosyć nieporadnie)
5) automatycznie dodaje wszystkie brakujące elementy skryptu (niedomknięte nawiasy, średniki na końcu linii itp.; niestety często dosyć nieporadnie)
6) pokazuje wskazówki do kodu (możliwe parametry przy funkcjach, dostępne metody oraz właściwości; niestety w narzędziu CS3 działa to bardzo nieporadnie i nie zawsze otrzymamy oczekiwaną podpowiedź do kodu; skrótem do tej opcji jest CTRL+SPACJA)
7) wstawia bądź usuwa punkty przerwań w kodzie; dzięki punktom przerwań możemy przerwać działanie kodu we wskazanym momencie i sprawdzić jaką np. wartość posiada dana zmienna, gdzie kod naszego skryptu nas przeniesie itd. Punkty przerwań są oznaczone na marginesie kodu czerwonymi kropkami (można je też ustawiać/ustawiać po prostu klikając na marginesie)



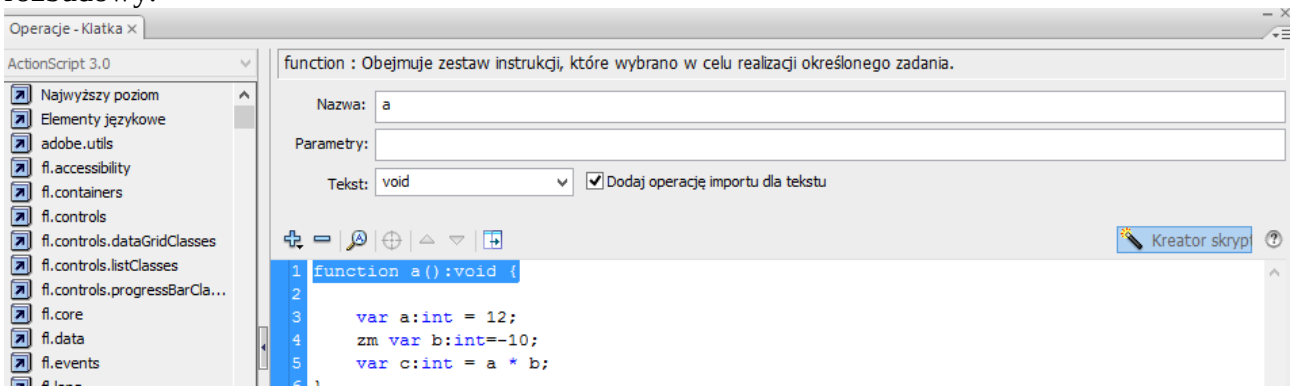
8) zwija zawartość w nawiasach klamrowych, w których aktywny jest kursor tekstu (czyli np. aktywna funkcja lub warunek). Zwinięty kod wygląda np. tak:



9) podobnie jak poprzednia opcja lecz zwija tylko zaznaczony fragment kodu:



10) rozwija zwinięty kod z nawiasów/zaznaczenia
11) wstawia komentarz blokowy (/**/)
12) wstawia komentarz linii (//)
13) usuwa wstawiony komentarz (jeżeli takowy znajduje się w linii kursora/zaznaczenia)
14) pokazuje/ukrywa przybornik (lewa strona okna)
15) ułatwia dodawanie skryptów/pozwala na ich edycję i zrozumienie. Przykładowe działanie kreatora to zaznaczanie już napisanego kodu i sprawdzanie jego poprawności/możliwych działań rozbudowy:

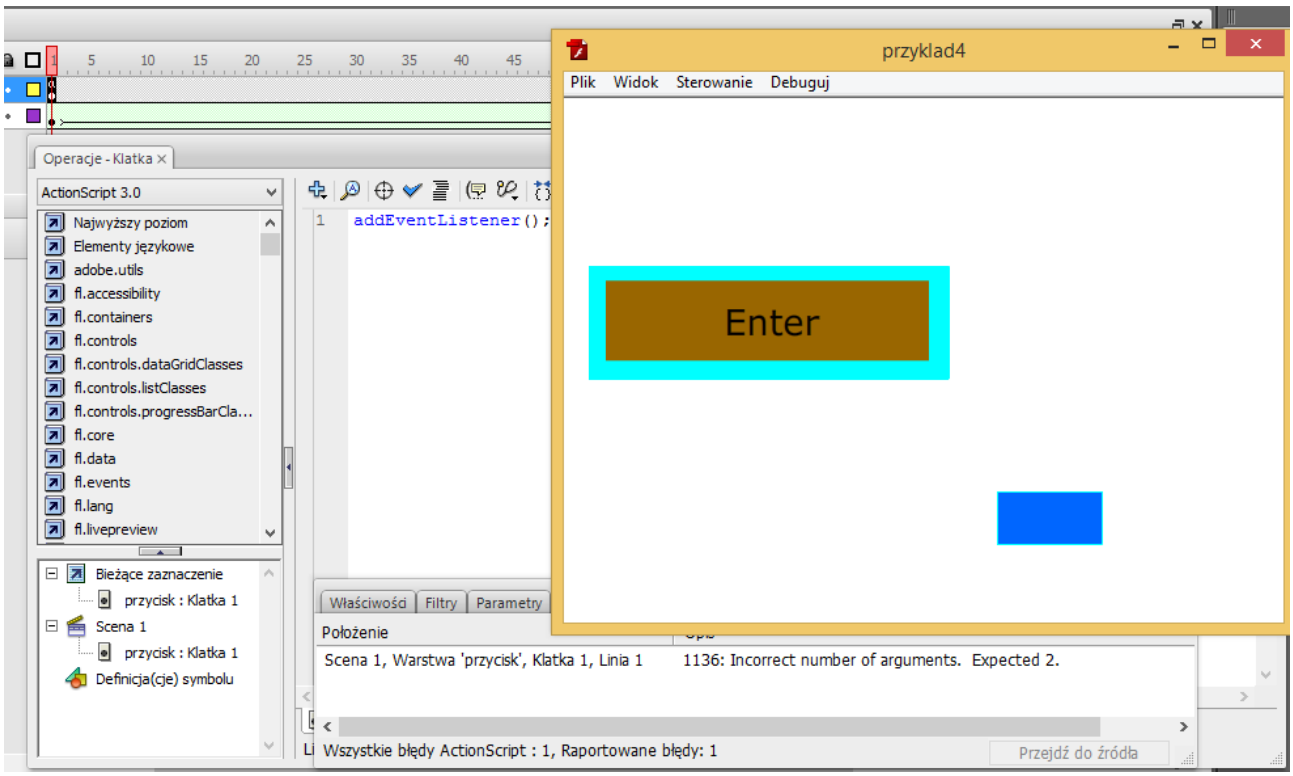


16) otwiera okno wbudowanej pomocy Adobe Flash Professional (dokumentacja języka w wersji angielskiej)

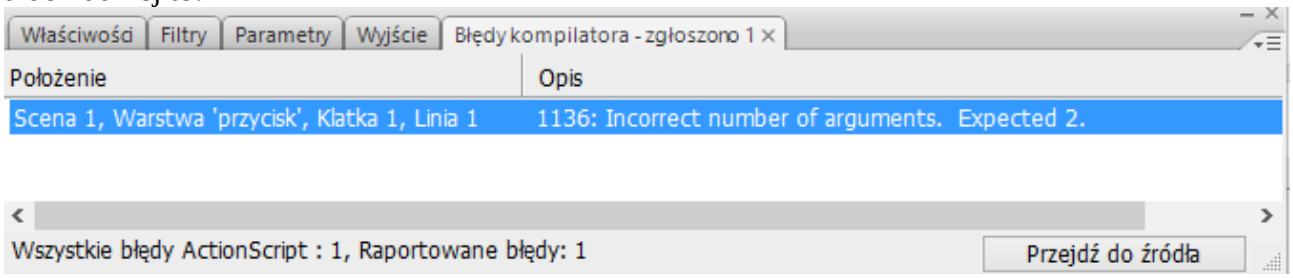
Spróbujmy tak oprogramować nasz przycisk, by jego pierwsze naciśnięcie powodowało zatrzymanie animacji. Drugie zaś niech powoduje jej kontynuowanie.

Przede wszystkim musimy spowodować, że animacja będzie „nasłuchiwać”, czy dane zdarzenie w ogóle miało miejsce (w naszym przypadku czy kliknięto w jej obrębie lewym przyciskiem myszy, a dokładniej czy kliknięcie miało miejsce na przycisku). Aby tego dokonać, w otwartym oknie wpisujemy taką oto metodę:
addEventListener();

Jeżeli tego dokonamy to zwróćmy uwagę na to, że w klatce kluczowej (czarne kółko), dla której dodaliśmy nasz kod pojawiła się mała litera 'a'. Oznacza ona tyle, że dla tej klatki dodano jakiś kod (wiemy, że coś ma się w niej dziać). Teraz uruchommy naszą animację (poprzez menu lub CTRL+ENTER). Niestety „coś” będzie nie w porządku:



a dokładniej to:



Kompilator „mówi” nam, że napotkał błąd w skrypcie – 1 linia znajdująca się w scenie pierwszej, warstwie o etykiecie „Przycisk”, w klatce 1, posiada funkcję, w której występuje niepoprawna liczba argumentów (oczekuje dokładnie 2).
Przejdźmy z powrotem do kodu. Funkcja faktycznie nie posiada odpowiedniej ilości argumentów (w ogóle ich nie posiada). Dokładna definicja użytej funkcji (metody) wygląda następująco:

addEventListener(<zdarzenie>:Object, funkcjaZwrotna:void):void

gdzie pod zdarzenie wstawiamy konkretne zdarzenie, którego oczekujemy, a pod funkcjaZwrotna mają znaleźć się polecenia, które ma wykonać nasza animacja gdy zdarzenie to będzie miało miejsce.

Zacznijmy od zdarzenia. Potencjalne zdarzenia poznaliśmy w poprzednim materiale. Teraz czas na praktyczną realizację. Jeżeli chcemy, by przycisk reagował na pojedyncze kliknięcie lewego przycisku myszy używamy takiej składnie:
MouseEvent.CLICK
jeżeli natomiast chcielibyśmy, aby przytrzymanie klawisza powodowało operację:
MouseEvent.MOUSE_DOWN
jest jeszcze wiele innych możliwości interakcji (opisane poprzednio). My wybierzemy pierwszą opcję z dwóch przedstawionych.

Jeżeli chodzi o funkcję zwrotną to mamy dwie możliwości. Pierwszą z nich jest utworzyć funkcję nazwaną i później odwołać się do jej nazwy. To jednak kiepski pomysł ponieważ funkcję tę

najprawdopodobniej wykorzystamy tylko jeden raz. Dlatego najlepszym pomysłem będzie użyć funkcji nienazwanej, zadeklarowanej i zdefiniowanej bezpośrednio w parametrze. Jej postać powinna być następująca:

```
function(event:MouseEvent):void {
	stop();
}
```

Proszę zwrócić uwagę na fakt, że przyjmuje ona jeden parametr (argument) – zdarzenie, które ją wywołało. Dzięki temu, już we wnętrzu tejże funkcji mamy dostęp do wszystkich elementów tegoż zdarzenia (np. z lokalizacją kursora, kliknięty mi dodatkowymi klawiszami itp.). NAZWA event może zostać zastąpiona na dowolną inną, np. e albo zdarzenie – jedynym STAŁYM elementem jest nazwa klasy MouseEvent! Funkcja nic nie zwraca dlatego pojawia się za nią słowo void (puste). Na tę chwilę funkcja ma za zadanie zatrzymywać całą animację (wywołana metoda stop(); - działa w odniesieniu całej sceny).

Ostatecznie nasz kod powinien wyglądać np. tak:

```
addEventListener(MouseEvent.CLICK,
                    function(e:MouseEvent):void {
                        stop();
                    });
```

Przeniesienie drugiego parametru, funkcji, służy tylko poprawie czytelności kodu (można było wszystko zawszeć w jednej linii!). Przetestujmy naszą animację ponownie. Poklikajmy na animacji myszą (ale nie w przycisk!). Nic nie powinno się wydarzyć. Teraz kliknijmy w przycisk. Animacja powinna zatrzymać się w miejscu. Udało się – nasz kod działa poprawnie lecz połowicznie. Ponowne wybranie przycisku nie wznawia działania naszej animacji. Potrzebna będzie rozbudowa kodu o:

- dodanie nowej zmiennej (przechowującej wiadomość, czy aktualnie animacja jest zatrzymana czy nie)
- dodanie odpowiednich warunków w funkcji wywoływanej przy wystąpieniu zdarzenia (kiedy animacja ma się zatrzymywać, a kiedy wznawiać)

Pierwszy „postulat” można zrealizować za pomocą odpowiednio zadeklarowanej zmiennej dla naszego skryptu. Przykładowo przed funkcją addEventListener dodajemy taką zmienną:

```
var isStop:Boolean = false;
```

Domyślna wartość false będzie mówić, że animacja nie jest zatrzymana (dla przypomnienia – nazwę isStop można zmianić na dowolną, inną np. zatrzymana, stopped czy jakąkolwiek dla nas przystępniejszą; trzeba tylko pamiętać by w konsekwentnie WTEDY TO JEJ używać, nie tej z przykładu).
Drugą część relizujemy poprzed dodanie odpowiedniej instrukcji warunkowej if; zamiast pojedynczej funkcji stop() wstawiamy:

```
if (!isStop) {
	stop();
	isStop = true;
}
else {
	gotoAndPlay(currentFrame);
	isStop = false;
}
```

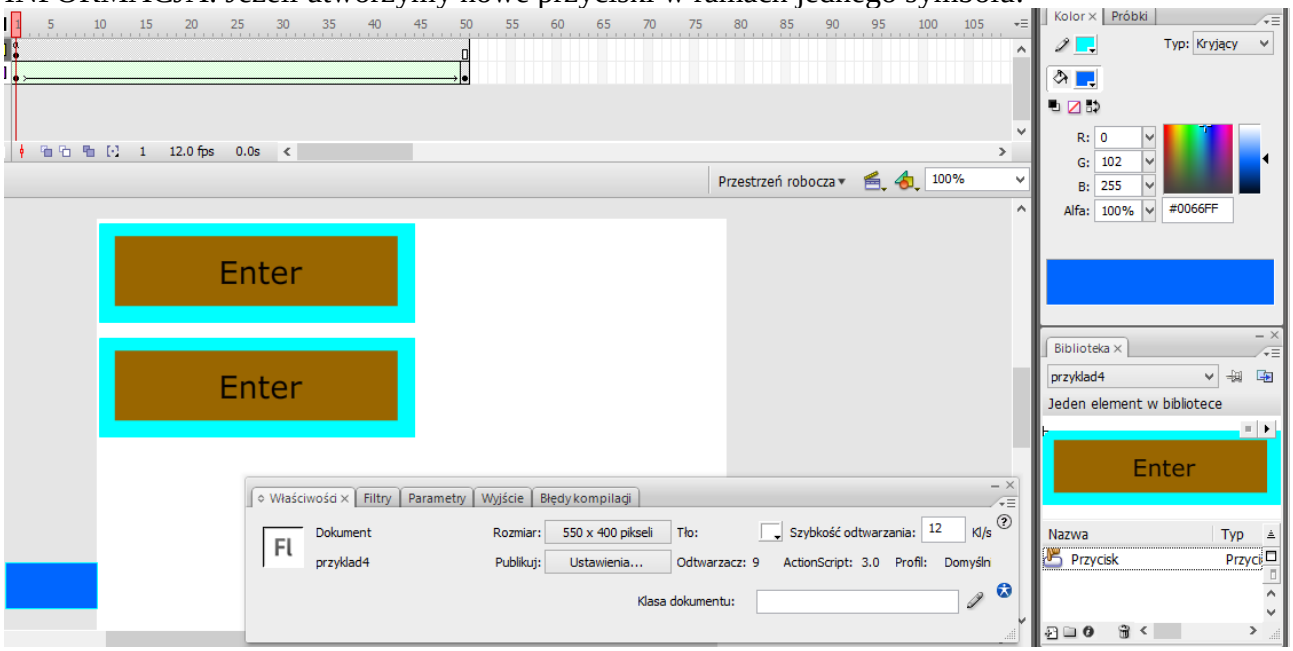
currentFrame to właściwość sceny, która przechowuje numer klatki, nad którą aktualnie znajduje się głowica odtwarzania. Jak łatwo się domyślić, w tej chwili funkcja „decyduje”, na podstawie wartości zmiennej isStop, czy ma zatrzymać animacje, czy też przejść do aktualnej ramki i zacząć ją ponownie odtwarzać (funkcja gotoAndPlay wznawia odtwarzanie od podanej ramki; jeżeli chcielibyśmy zacząć każdorazowo odtwarzanie od początku wystarczy currentFrame zamienić na wartość 1).
Dla przypomnienia – wykrzyknik przy zmiennej isStop w warunku if oznacza, że dany warunek zostanie spełniony w chwili, gdy wartość będzie różna odprawdy (fałsz). Jeżeli brakłoby wykrzyknika to warunek działałby poprzez ciągłe odtwarzanie - wartość isStop nidy nie uległaby zmianie.
Ostateczny kod:

```
var isStop:Boolean = false;

addEventListener(MouseEvent.CLICK,
                        function(e:MouseEvent):void {
                                if (!isStop) {
                                        stop();
                                        isStop = true;
                                }
                                else {
                                        gotoAndPlay(currentFrame);
                                        isStop = false;
                                }
                        });
```

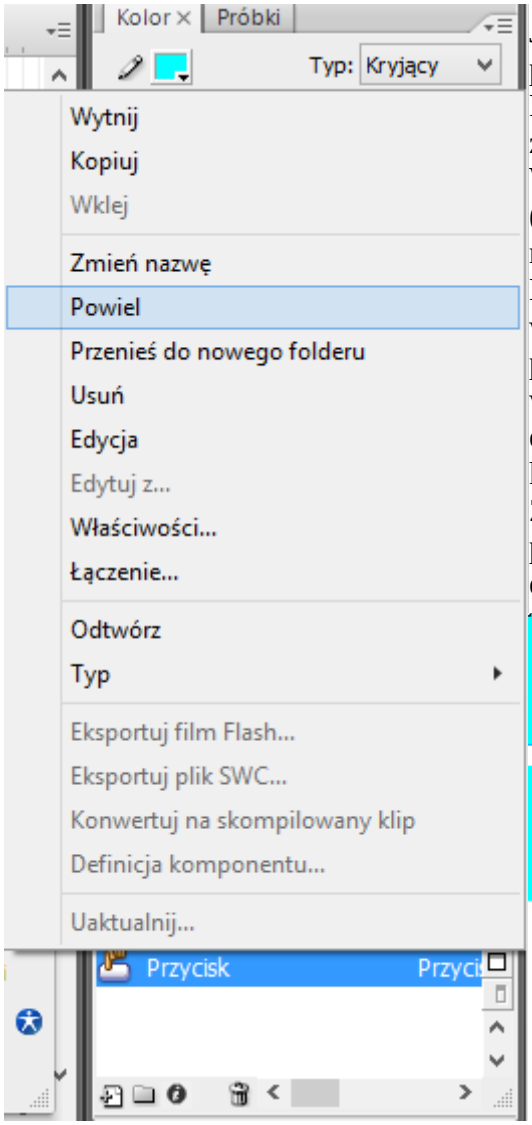
Ponownie testujemy animację. Wszystko powinno działać tak jak sobie życzymy.

Rozbudujmy naszą animację o drugi przycisk. Zmieńmy nazwę pierwszego przycisku na Stop, a drugiego na Start.

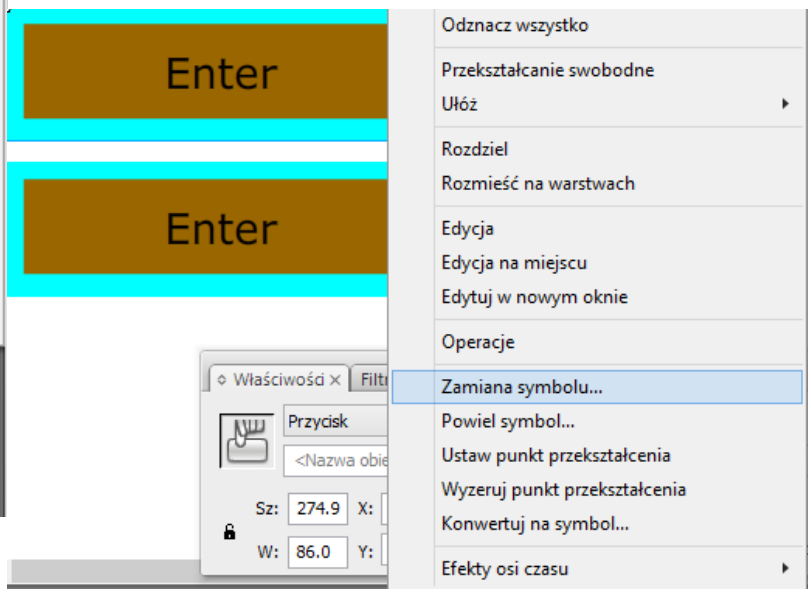
INFORMACJA: Jeżeli utworzymy nowe przyciski w ramach jednego symbolu:



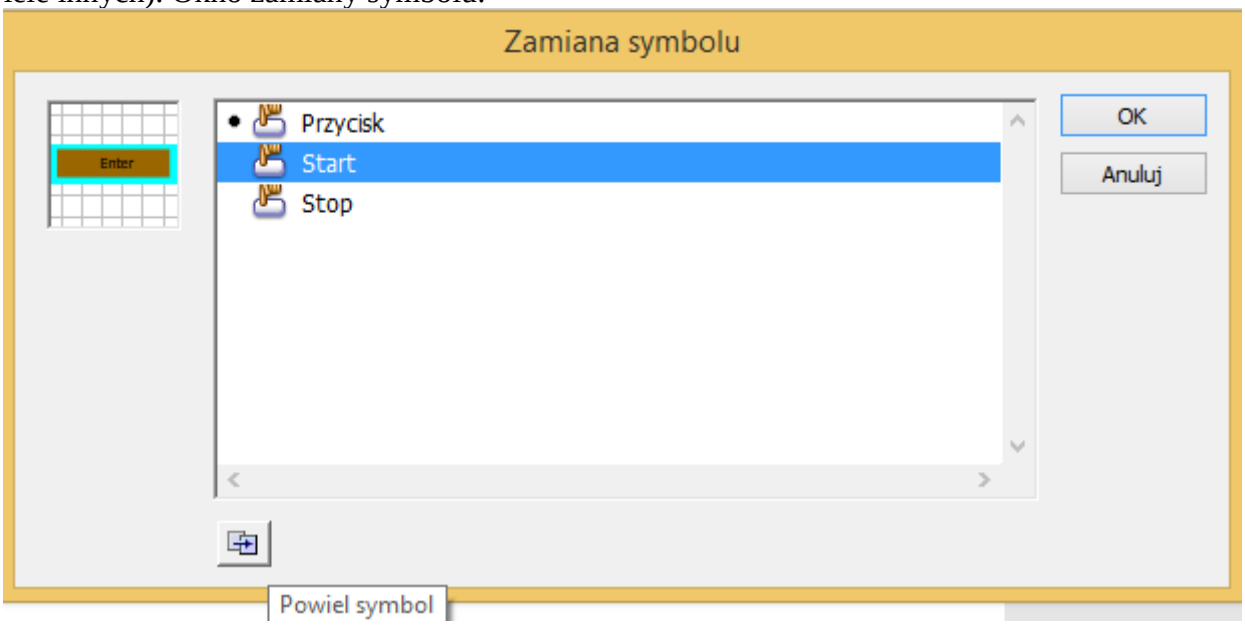
to najprawdopodobniej zmiana nawy jednego z nich skończy się automatycznie zmianą nazwy drugiego z nich. Dzieje się tak dlatego, że chociaż jeden element może być wielokrotnie dodawany do animacji, to kolejne jego instancje są jedynie odnośnikami (aliasami) lecz nie niezależnymi bytami. Dlatego chcąc stworzyć 3 przyciski zmuszeni bylibyśmy dodać 2 nowe symbole do biblioteki. Na szczęście narzędzie pozwala łatwo powielać wskazane symbole:



Jak widać na zrzucie obok wystarczy kliknąć prawym przyciskiem myszy na danym symbolu i wybrać opcje Powiel. Wyskoczy okno znane nam z tworzenia symboli z zaproponowana nazwą – <nazwaSymbolu> kopia. Wystarczy nadać mu własną nazwę, zaakceptować (przycisk OK) i gotowe – nasza biblioteka będzie posiadać nowy, niezależny symbol.
Nie musimy usuwać niczego z animacji (szczególnie ważne, gdy już rozplanowaliśmy położenie poszczególnych elementów na niej albo dodaliśmy do wcześniej dodanych symboli Filtry bądź dodatkowe efekty). Aby podmienić dany symbol na inny wystarczy kliknąć na nim prawym przyciskiem myszy, wybrać opcję Zmiana symbolu..., a z nowo otwartego okna wybrać pożądany przez nas symbol dostępny w bibliotece edytowanej animacji.



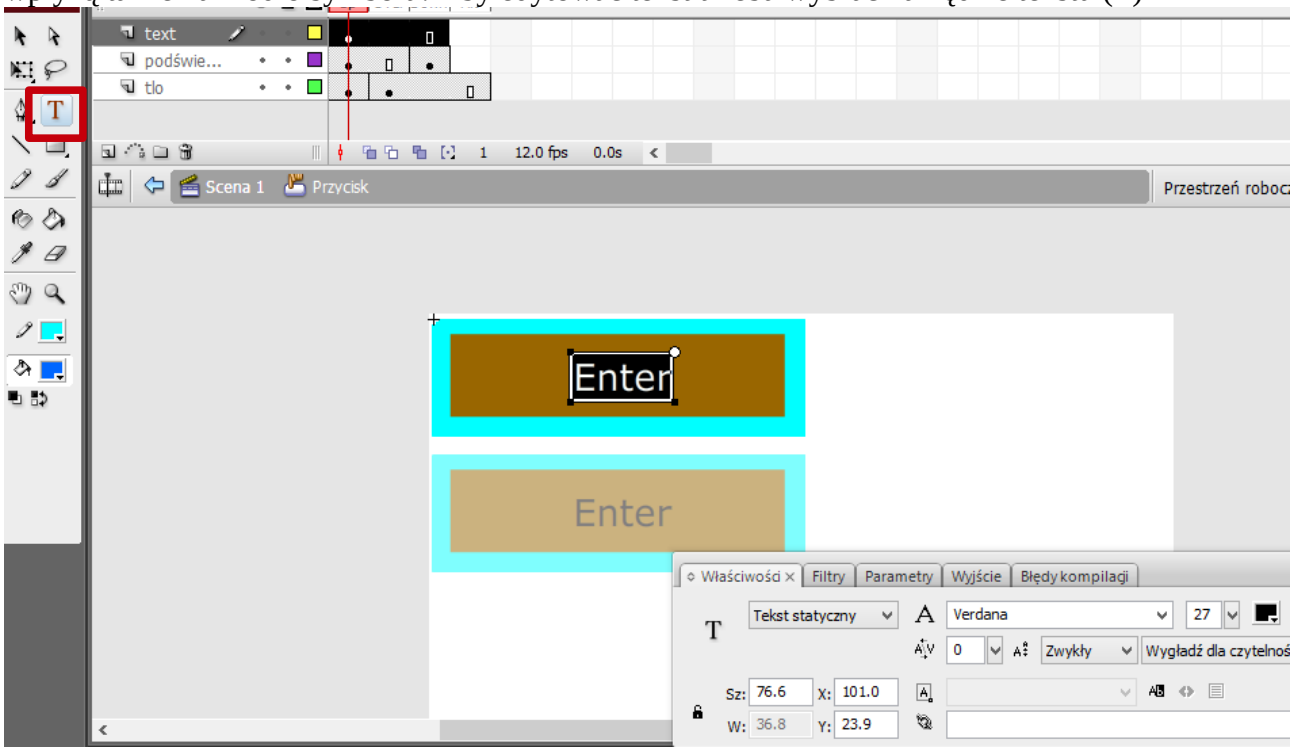
Proszę zauważyć, że symbol możemy powielać także z tego miejsca (nadawać Efekty osi czasu i wiele innych). Okno zamiany symbolu:



Proszę zauważyć ikonę Powiel symbol. Nawet z tego okna możemy dowolnie klonować nasze

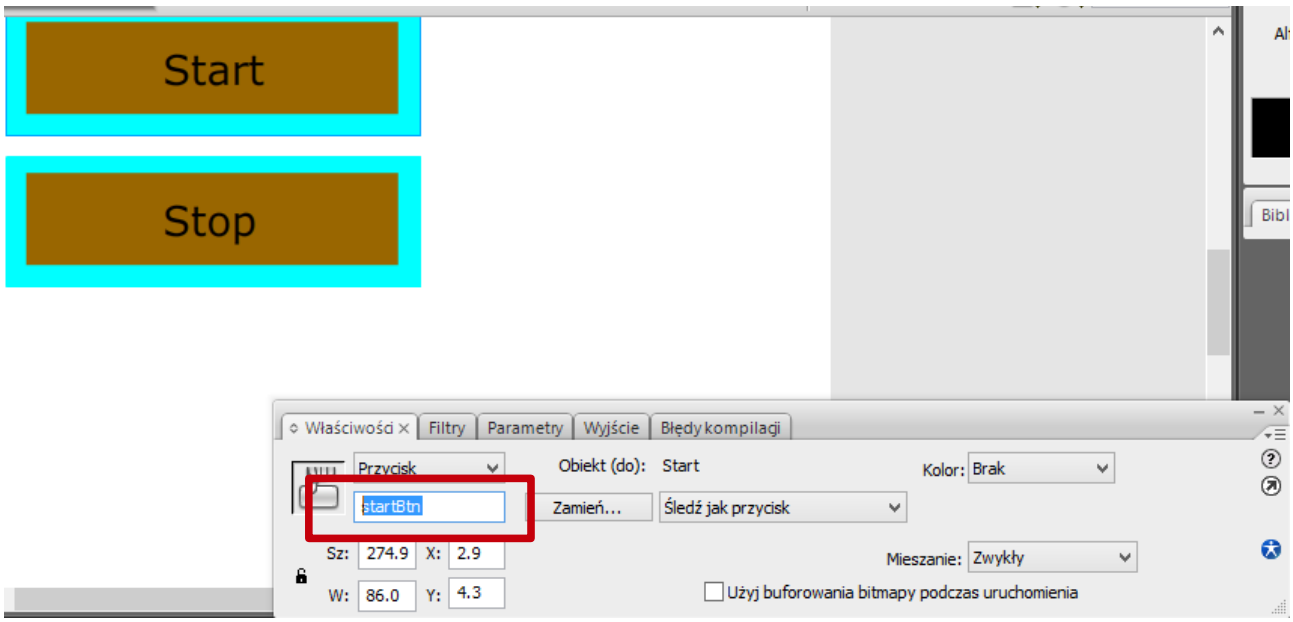
symbole by utworzyć ich dostateczną ilość.

Mając już „odseparowane” od siebie przyciski możemy przystąpić do edycji ich nazw. Najprostszym sposobem będzie po prostu dwukrotne kliknięcie na danym przycisku lewym przyciskiem myszy. Proszę zauważyć, że pozostałe elementy animacji będą przyszarzałe. Oznacza to, że nie możemy z nimi nic zrobić (edytujemy tylko ten konkretny symbol). Wszelkie zmiany wpłyną także na źródło symbolu! Aby edytować tekst trzeba wybrać narzędzie tekstu (T)



Po zmianie nazw proszę uruchomić animację. Proszę zauważyć, że:
- każdy z przycisków niejako odziedziczył przechwytywanie zdarzenia
- jeżeli klikniemy dwa razy na jednym z nich to dopiero dwukrotne kliknięcie na drugim z nich zacznie odnosić jakiekolwiek znaczenie (AS nie wie z którego przycisku wpłynęło zdarzenie – próbuje je wykonać dla pierwszego lepszego)
- animacja nie działa jak chcieliśmy

Jak naprawić powyższą niedogodność? Nadać nazwy w AS dla poszczególnych przycisków. Od tego momentu ich nazwy w AS będą odnośnikami do obiektów tychże przycisków pozwalając tym samym na odwoływanie się do nich i ich właściwości/metod. Z poziomu AS symbole przycisków traktowane są tak samo jak klipy filmowe – pochodzą od tego samego obiektu-rodzica. Poniższy zrzut ekranu pokazuje gdzie należy wpisać nazwę naszego symbolu-przycisku (proszę pamiętać o tym że MUSI być zaznaczony tylko jeden przycisk – domyślnie przy wybraniu ich warstwy zaznaczą się oba!). Nazwa może być dowolna, proszę jednak pamiętać by później używać ustalonej na tym poziomie nazwy w kodzie AS.



Po tej operacji (oba przyciski mają swoje nazwy) przechodzimy do kodu. Zamieniamy kod na następujący:

```
startBtn.addEventListener(MouseEvent.CLICK,
                          function(e:MouseEvent):void {
                              gotoAndPlay(currentFrame);
                          });

stopBtn.addEventListener(MouseEvent.CLICK,
                          function(e:MouseEvent):void {
                              stop();
                          });
```

Jak widać wyeliminowana została potrzeba korzystania ze zmiennej pomocniczej oraz warunków. Po uruchomieniu animacji cała działa jak należy (przycisk stop ją zatrzymuje, przycisk start odtwarza ponownie).

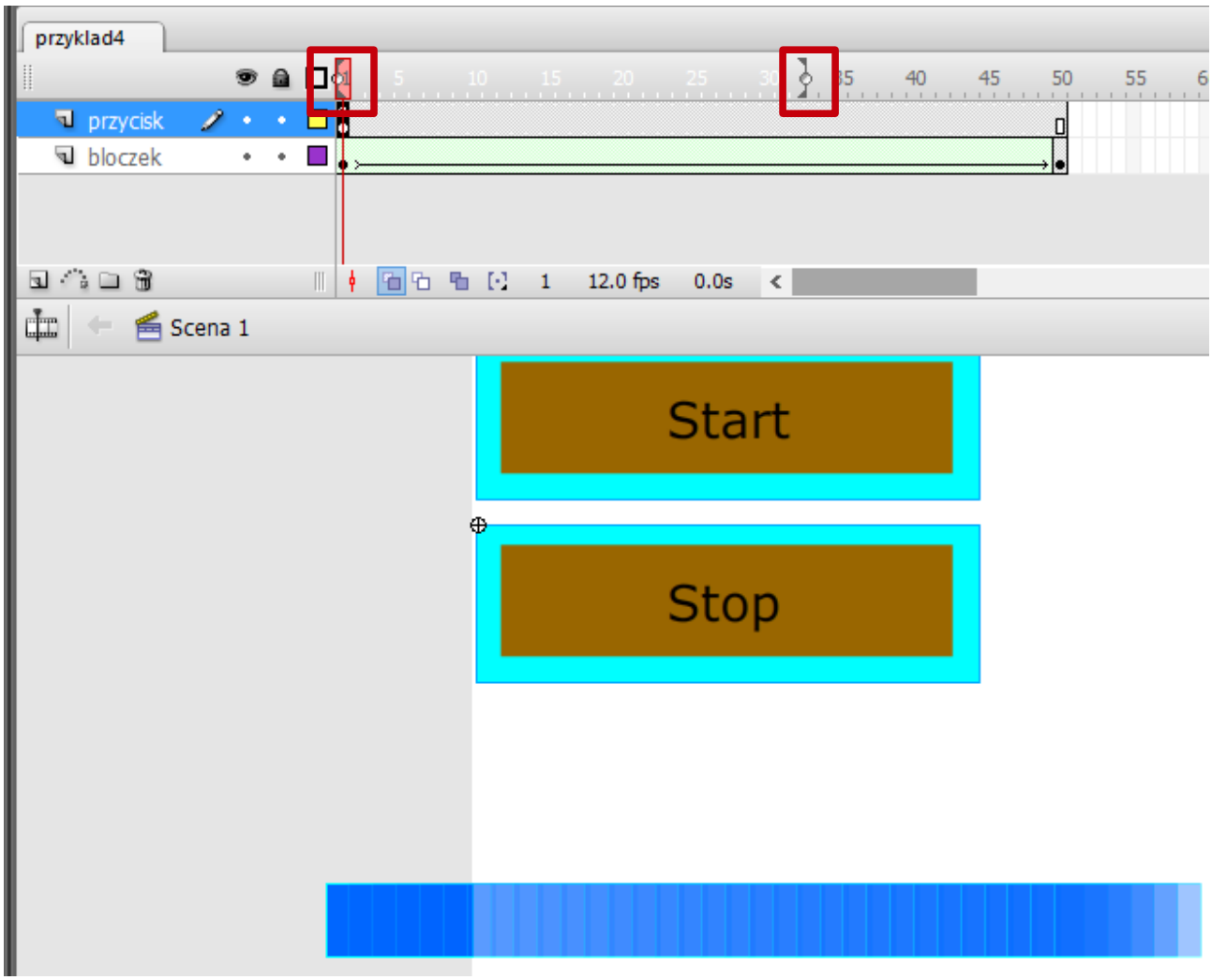
2. Animowanie kształtów i obiektów.

Dotychczas wykorzystywany był prosty system animacji – co klatka kluczowa wskazany element był przemieszczany, jego wielkość i szerokość zamieniana, a program Flash sam obliczał pomiędzy nimi klatki, które powodowały płynne przejście pomiędzy jednym stanem a drugim (powodując płynną animację).

Takie rozwiązanie może być jednak niewystarczające. Czasem zachodzi potrzeba zastosowania o wiele bardziej złożonej animacji,w której stany mogą zmieniać się nieliniowo. Poniżej zostaną omówione i przedstawione na przykładach najpopularniejsze rozwiązania.

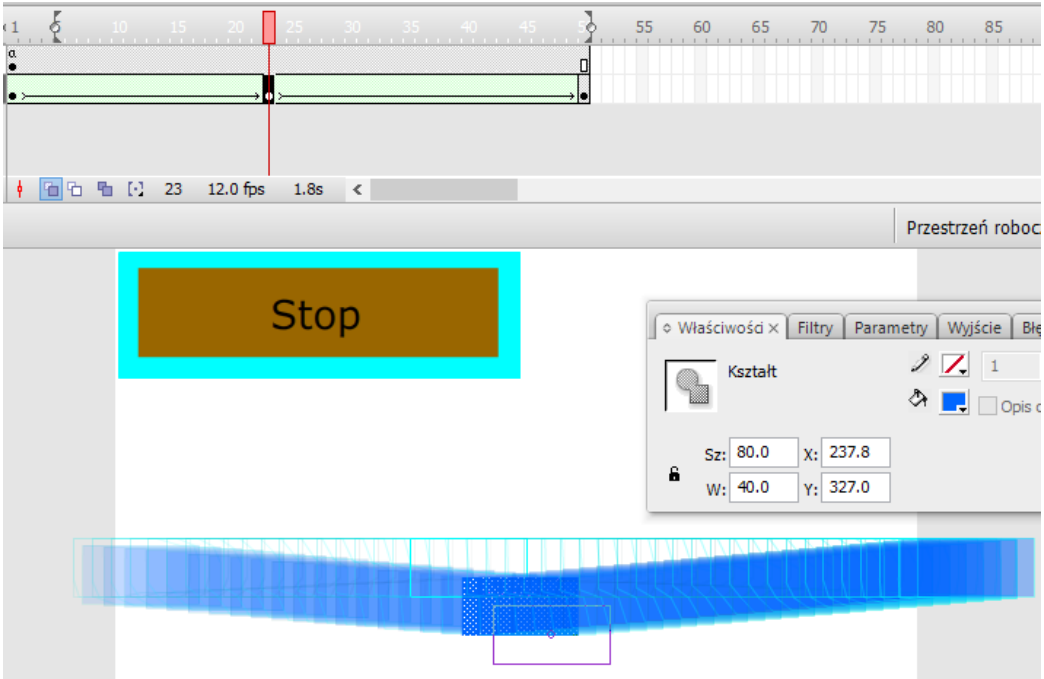
a) wykorzystywanie opcji Łuski cebuli

Gdy stworzymy już animację liniową i chcielibyśmy ją zmodyfikować, lecz nie wiemy w którym momencie powinniśmy edytować animacją, może pomóc nam właśnie ta opcja. Po jej uaktywnieniu na linii czasu pojawiają się dwa nawiasy z kropkami:

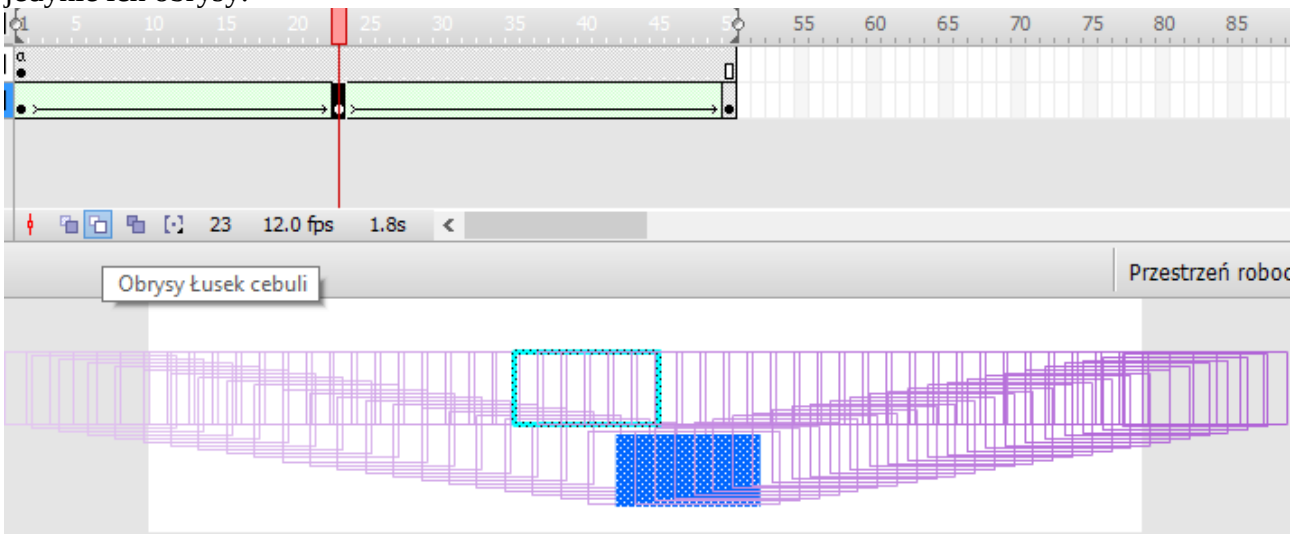


Czerwonymi prostokątami oznaczony został zasięg tychże łusek cebuli. Możemy go dowolnie zmieniać poprzez przesuwanie (wystarczy nacisnąć na jednej z łusek lewym przyciskiem myszy i, mając cały czas wciśnięty klawisz, przesuwać ją w lewo lub prawo). Proszę zwrócić uwagę na wygląd animowanego prostokąta – uwidocznione zostały jego wszystkie zmiany na kolejnych klatkach animacji (od startu do zakończenia łuski cebuli).

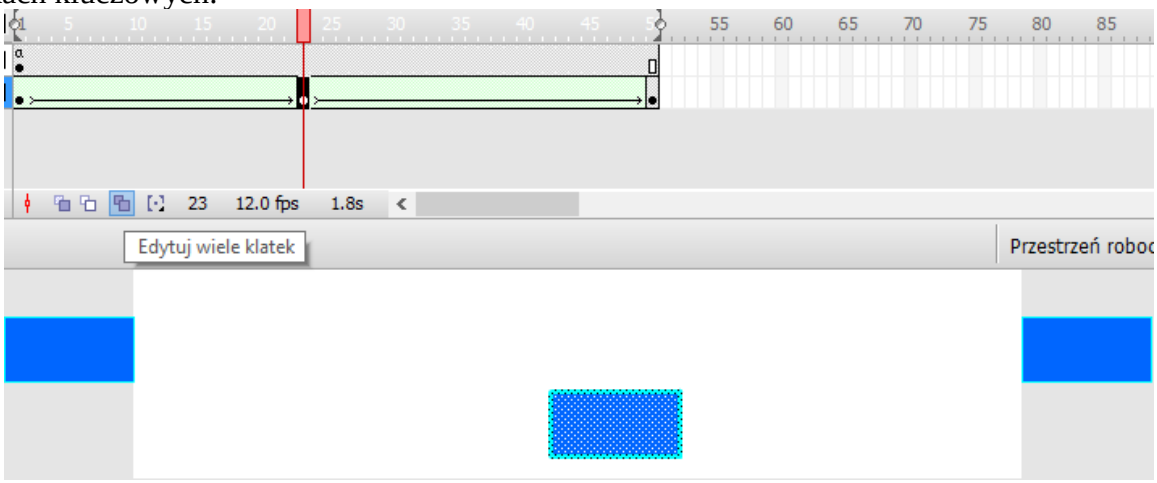
Jak wykorzystać powyższy efekt? Trzeba wybrać dowolną klatkę animacji w obrębie rozrysowanych łusek. Następnie na warstwie animacji naszego kształtu (BARDZO WAŻNE by to była właśnie ta warstwa, nie żadna inna) wstawiamy klatkę kluczową.



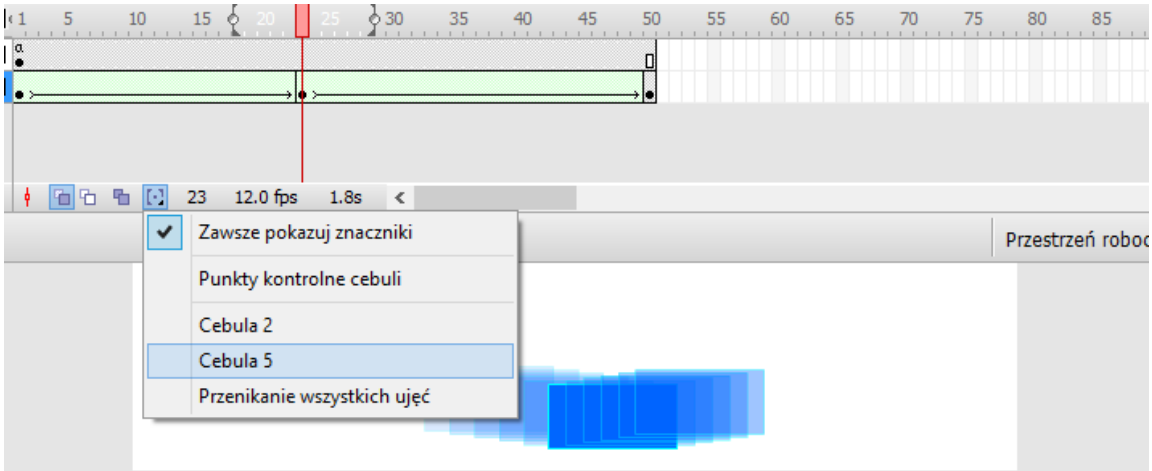
Jak widać na powyższym zrzucie, gdy wybierzemy nowo dodaną klatkę kluczową, możemy swobodnie przenosić fragment animacji w niej zawarty. Dzięki łuskom cebuli dokładnie widzimy jak wpłynie to na ostateczny tor obiektu. Możemy nawet animować w ten sposób obiekty/grupę obiektów klatka po klatce.
Można także wykorzystać opcję obrysu łusek. Wtedy nie będzie widoczny kolor elementów a jedynie ich obrysy:



Aktywacja edycji wielu klatek pozwala z kolei widzieć ustawienie naszego kształtu w kolejnych ramkach kluczowych:

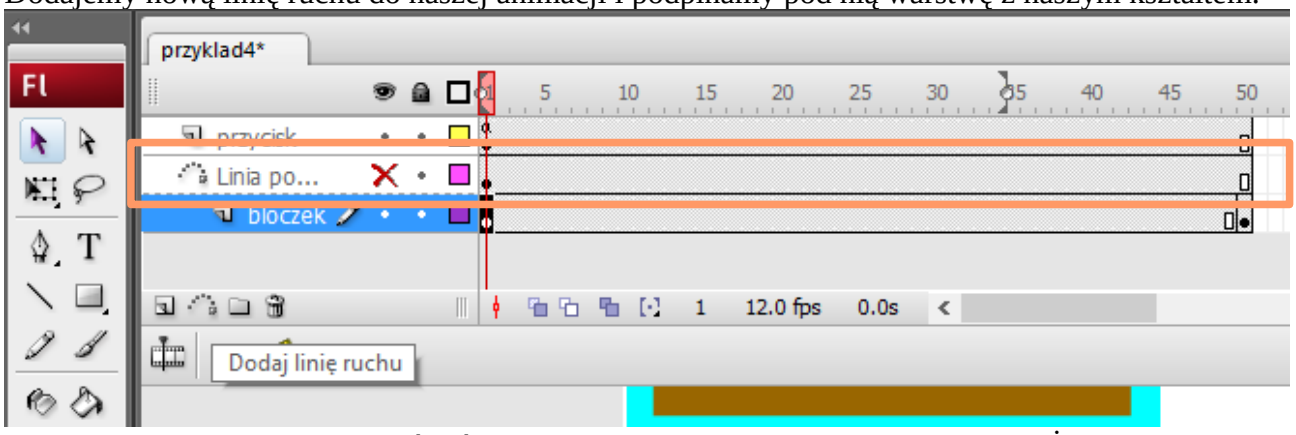


Ostatnia opcja to dodatkowe ułatwienia edycji linii czasu. Można ustalić, że znaczniki cebuli są zawsze widoczne (nawet bez jej trybu), jej punkty kontrolne, a także można zmienić wyświetlanie łusek 2 do przodu/2 do tyłu (łącznie pięć klatek) lub 5 łusek do przodu/tyłu (łącznie 11 klatek). Wybranie przenikania wszystkich ujęć powoduje powrót do ustawień domyślnych. Poniżej przykład wyboru 5 łusek:

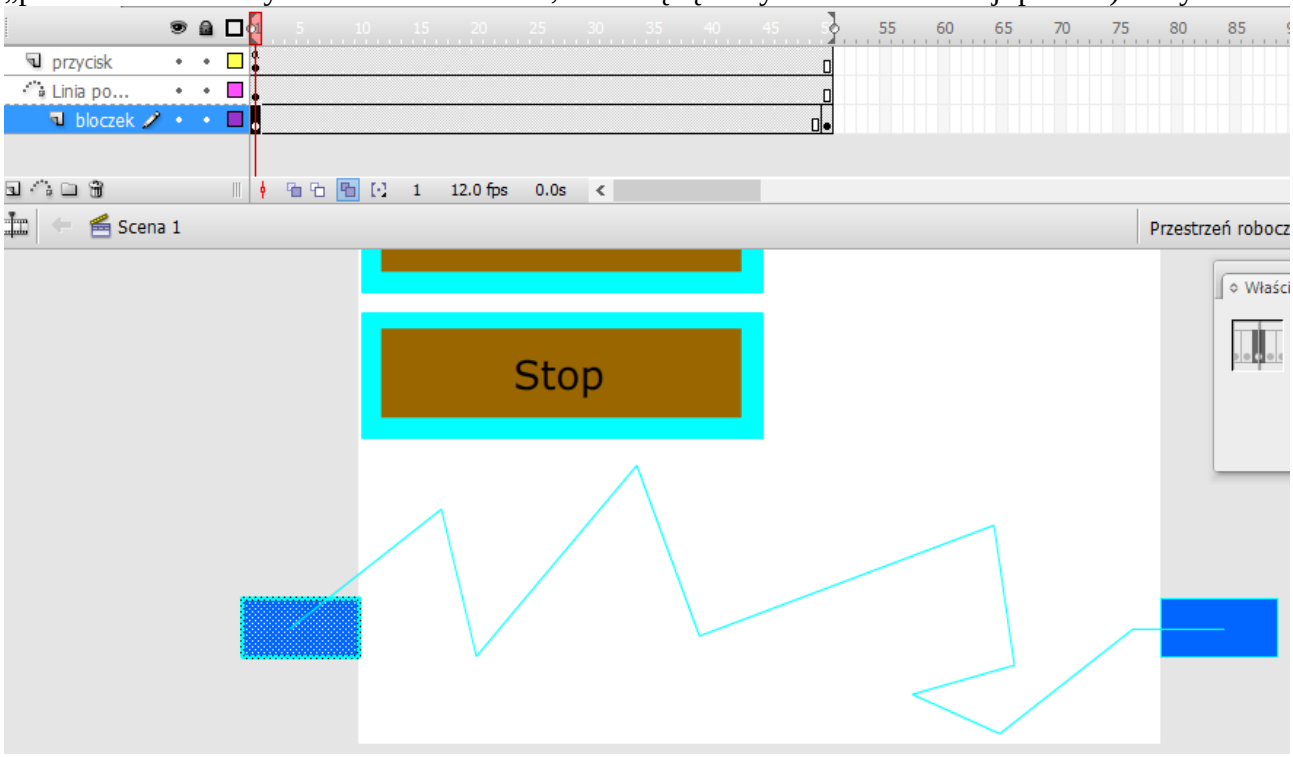


b) animacja po linii ruchu

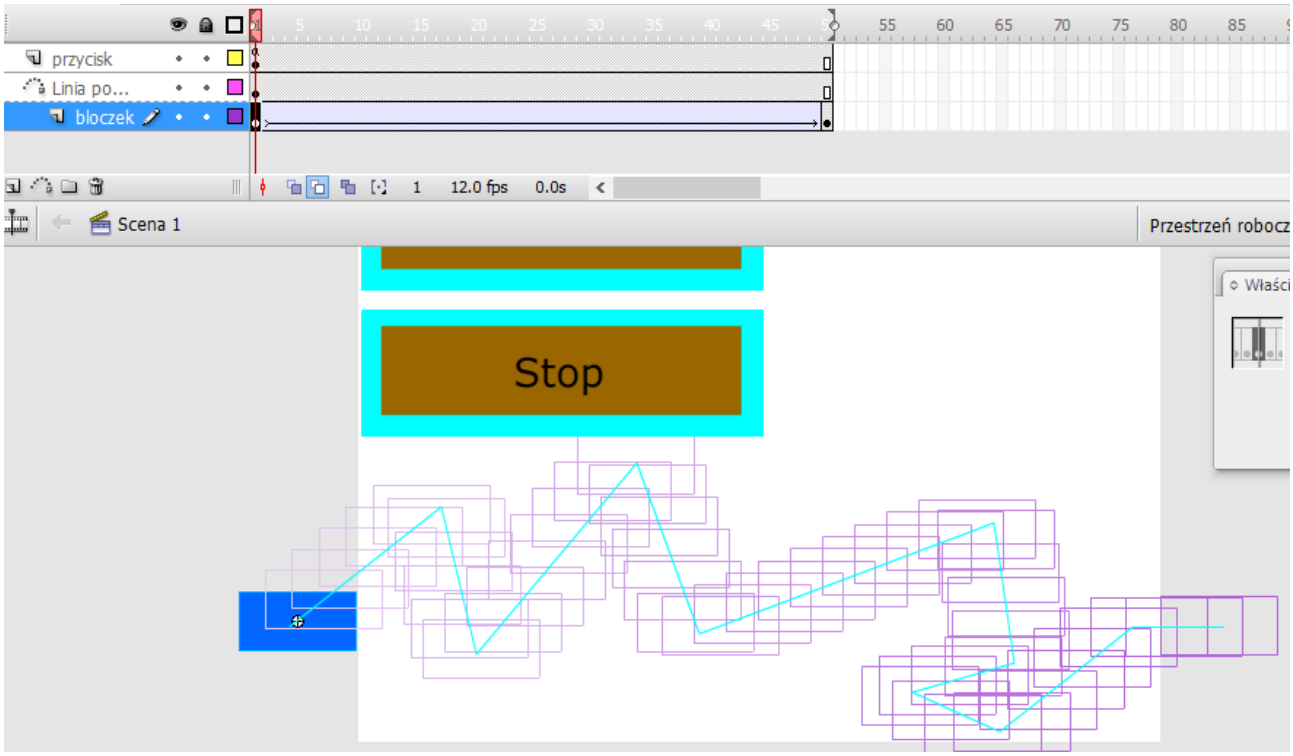
Jeżeli mamy zaimplementować animację obiektu poruszającego się w określony sposób lecz nie chcemy za dużo czasu spędzić nad klikaniem kolejnych klatek kluczowych, możemy spróbować dodać do animację linię ruchu. Linia ruchu to specjalna warstwa, na której kreśli się linie bądź tworzy odrys narzędziem ołówka. Następnie dodaje się do utworzonej linii ruchu warstwy z kształtami, które mają być za jego pomocą animowane. Najlepiej użycie tego zilustruje przykład: Dodajemy nową linię ruchu do naszej animacji i podpinamy pod nią warstwę z naszym kształtem:



Wybieramy narzędzie linii (bądź ołówka) i rysujemy linię (kształt odręczny). WAŻNE by rysowanie odbywało się na „warstwie” ruchu (nie po warstwie z kształtem do animacji!). Poza tym początek linii/kształtu musi zacząć się i zakończyć na środku danego obiektu/kształtu (dlatego najlepiej „pokazać”sobie wszystkie klatki kluczowe, które będą brały udział w animacji po linii). Przykład:



Proszę zauważyć, że na powyższym przykładzie bloczek nie posiada już swojej animacji. Został on skasowany (prawy przycisk myszy i opcja Usuń generowanie klatek pośrednich). Teraz by dokończyć ten rodzaj animacji dodajemy na warstwie z blokiem (prostokątem) ANIMACJĘ RUCHU (kształt nie może być animowany po linii). Animacja po wygenerowaniu klatek pośrednich może wyglądać np. tak (włączony obrys klatek):

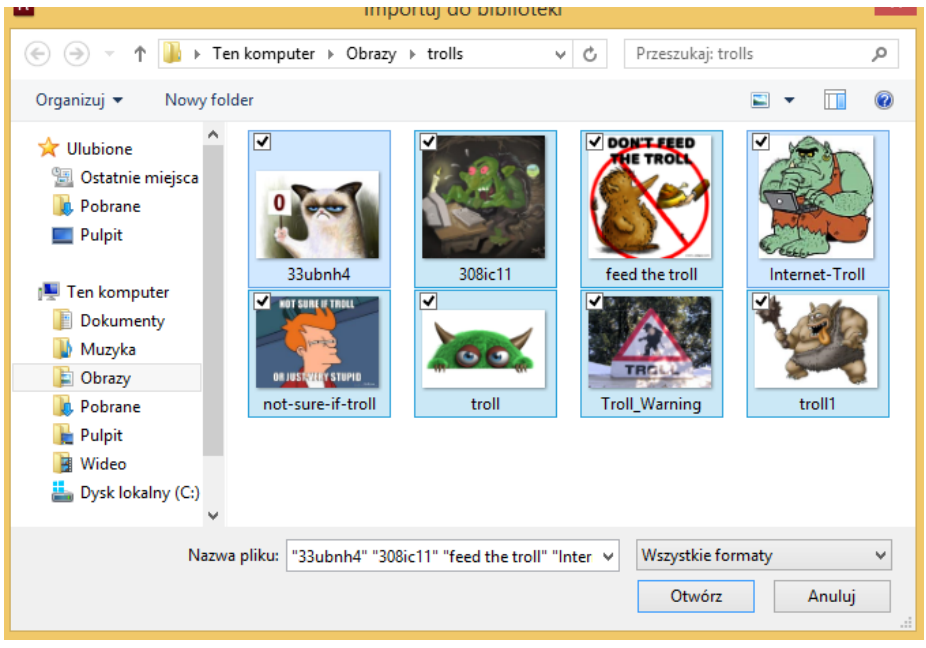


c) rysowanie poszczególnych klatek kluczowych samodzielnie

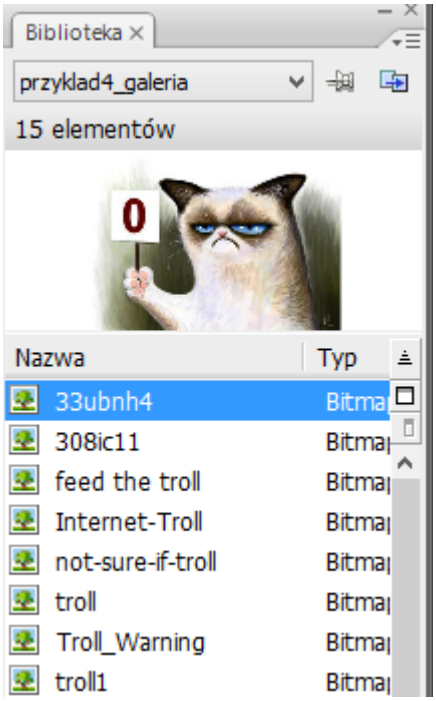
Niestety w przypadku bardzo złożonych animacji nie będziemy mogli polegać na żadnej z wyżej pokazanych metod animacji. Pozostanie przerysowywanie poszczególnych fragmentów sceny „ręcznie”, wstawiając klatki kluczowe co kolejną ramkę animacji. Trzeba jednak pamiętać, że każda kolejna klatka kluczowa odpowiednio zwiększa wynikowy plik animacji SWF – dzieje się tak dlatego, że narzędzie musi dla każdej klatki kluczowej przerysować i zachować całą scenę (w tym np. wszystkie kształty czy symbole).
Innym rozwiązaniem jest rozbijanie animacji na poszczególne symbole (pośrednie animacje) i włączanie ich do głównej sceny. To także spowoduje w pewnym stopniu zmniejszenie nakładu pracy przy całej scenie oraz uczyni ją bardziej „czytelną”.

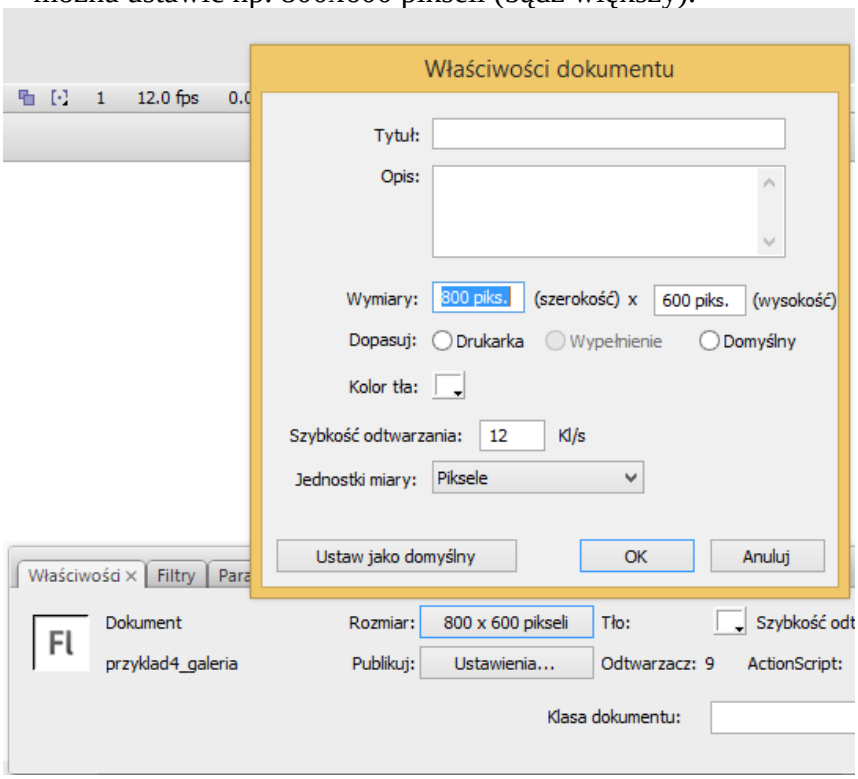
3. Tworzenie galerii zdjęć w programie Adobe Flash Professional.

Animacje Flash pozwalają także na tworzenie galerii zdjęć. Galerię można tworzyć poprzez dodanie zdjęć do naszej biblioteki, utworzenie z nich symboli, utworzenie ich miniaturek by następnie powiększać je przy kliknięciu. Bez zbędnych opisów spróbujmy utworzyć taką galerię krok po kroku:
a) klikamy menu Plik->Importuj->Importuj do biblioteki...
b) wybieramy obrazy, które chcemy mieć w bibliotece (im więcej tym lepiej)

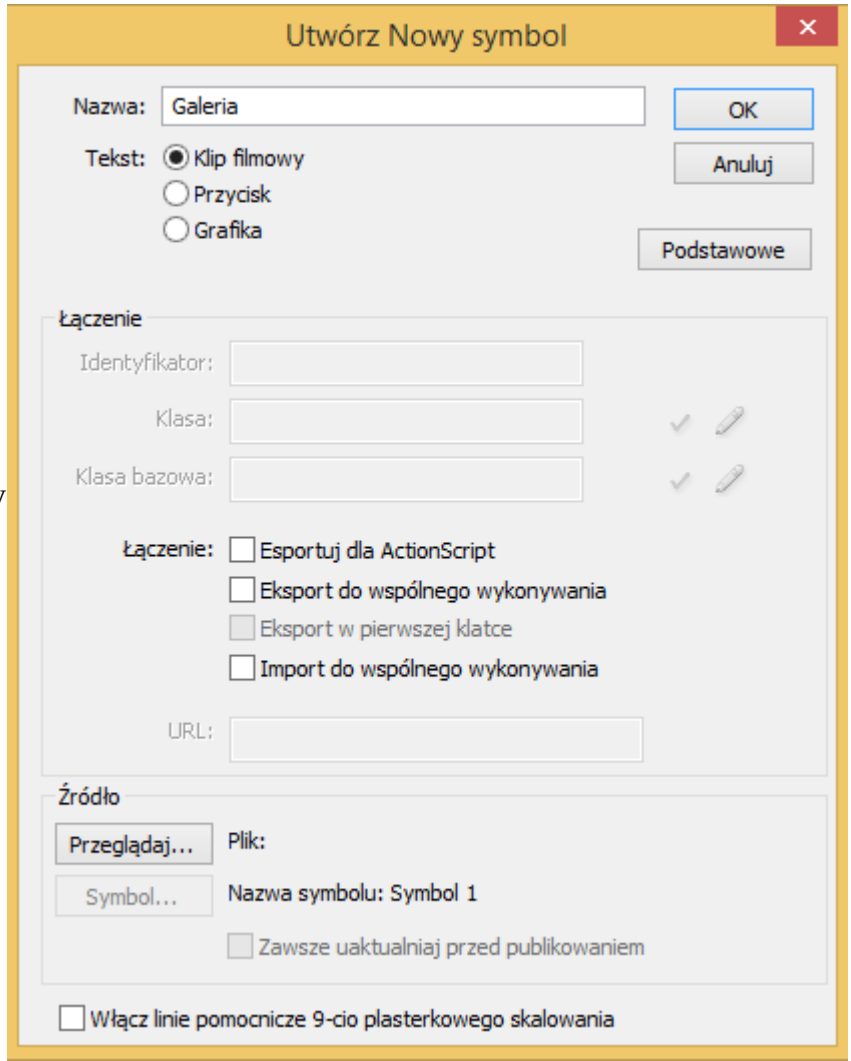


c) po dodaniu nasza biblioteka powinna wyglądać mniej więcej tak (wszystko zależy od załadowanych do niej plików)
d) ponieważ galerię tworzymy na nowym projekcie dobrze byłoby zadbać o jej odpowiednią wielkość; w tym celu klikamy dowolną część przestrzeni naszej sceny i wybieramy przycisk Rozmiar (bądź używamy skrótu CTRL+J). Rozmiar sceny można ustawić np. 800x600 pikseli (bądź większy).





d) tworzymy nowy symbol (CTRL+F8). Symbol ten będzie niezależną od sceny animacją zawierającą wszystkie zdjęcia (bitmapy) do wyświetlenia. Nazwijmy go jakkolwiek (dla celów przykładu nadana mu zostanie nazwa Galeria). Ma być klipem filmowym.
e) zaraz po utworzeniu symbolu program przeniesie nas do edycji go. To w nim poustawiamy wszystkie zdjęcia do wyświetlenia oraz utworzymy wymagane animacje. Teraz ustawmy obrazki, które chcemy wyświetlać w naszej animacji poprzez przeciągnięcie ich na obszar roboczy.



Proszę pamiętać, że dodanie zdjęcia do naszego symbolu może zakończyć się np. tak:

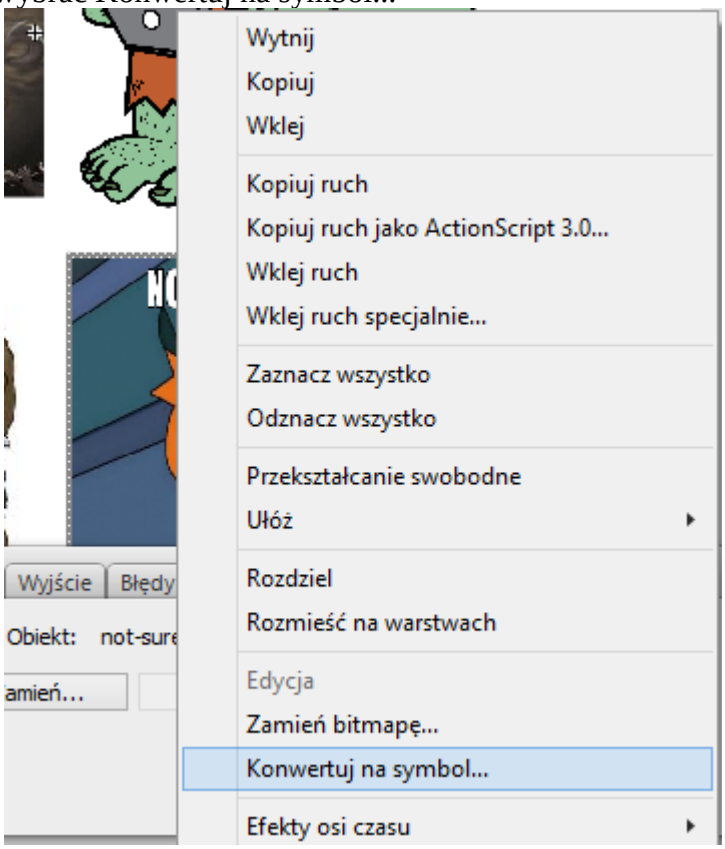
Dzieje się tak dlatego, że źródło posiada obraz w tak dużej rozdzielczości. Rozwiązaniem tego problemu jest:

- edytować obraz w zewnętrznym programie (Photoshop bądź w przypadku jego braku może być Paint .NET, GIMP); pozwoli to zoptymalizować ostateczny rozmiar naszej animacji (wszystkie pliki są dołączane do pliku swf; duże pliki graficzne dodane do niego drastycznie zwiększają ostateczny rozmiar całej animacji,a to z kolei wpływa na długie ładowanie się strony, do której została ona dołączona)
- jeżeli potrzebujemy obrazu o takiej rozdzielczości (bo gdzieś wykorzystujemy go w pełnym wymiarze) to możemy go tylko pomniejszyć w tym fragmencie animacji (zmienić jego szerokość i wysokość, a także punkt rysowania XY). Za każdym razem Flash i tak korzysta z tego właśnie źródła (dodajemy raz, a korzystamy wielokrotnie) więc dzięki temu nie trzeba będzie dodawać grafik w różnych rozmiarach (powiększanie mogłoby drastycznie mienić jakość grafiki na jej niekorzyść).

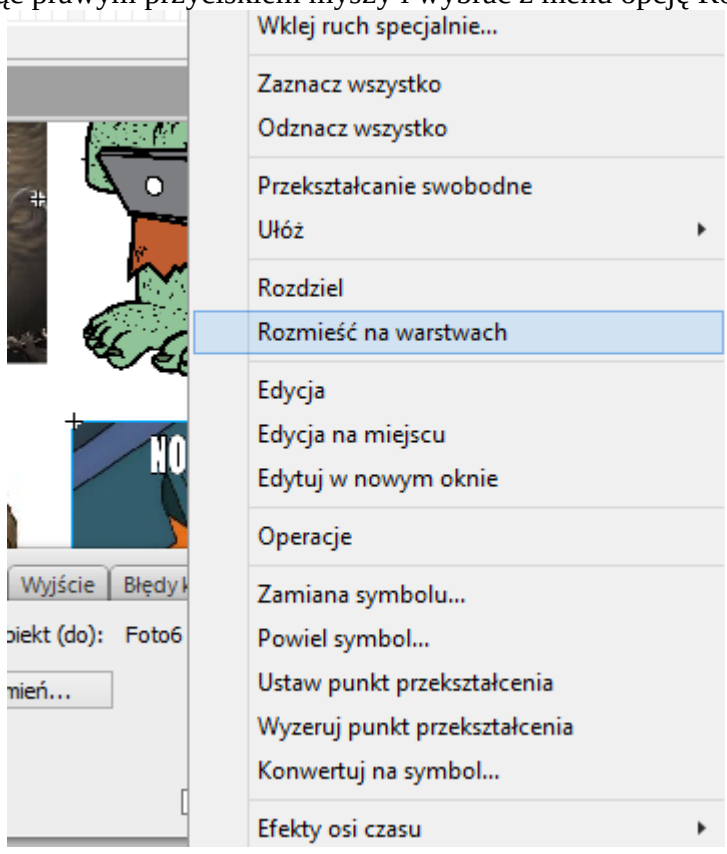
f) niestety tak utworzona galeria nie będzie działać! Dzieje się tak dlatego, że do obiektów graficznych (w tym wypadku obiekty typu Bitmap) nie można dodawać np. zdarzeń, nie wspominając o nadawaniu odpowiednich animacji. O wiele lepszym sposobem będzie zamienić nasze obrazki w nowe symbole – animacji (oczywiście można też operować na bitmapach; kwestia tego iż wymagałoby to napisania większej ilości kodu, chociaż w ogólnym rozrachunku byłaby to

metoda najefektywniejsza pod względem wydajnościowym!)

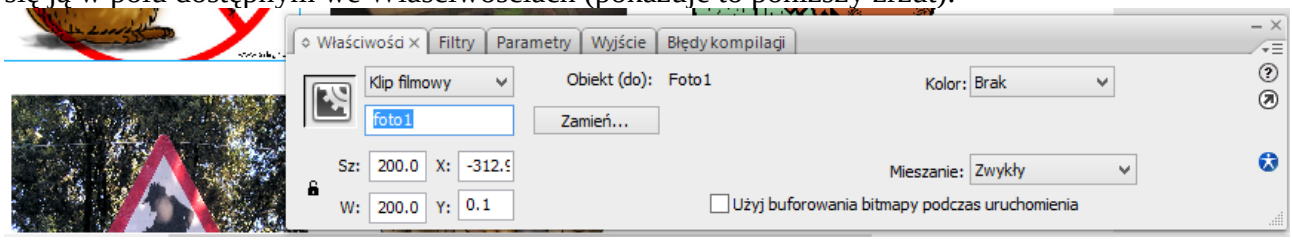
Aby zmienić aktualnie wstawione bitmapy w symbole wystarczy kliknąć na nich prawym przyciskiem myszy i wybrać Konwertuj na symbol...



Proszę tylko pamiętać, że nazwy poszczególnych klipów muszą być unikatowe.
g) wszystkie dodane symbole muszą być na osobnych warstwach. Jeżeli dotychczas nie rozmieściliśmy symboli po warstwach (wszystkie są na jednej) nic straconego. Wystarczy każdy kolejny symbol kliknąć prawym przyciskiem myszy i wybrać z menu opcję Rozmieść na warstwach

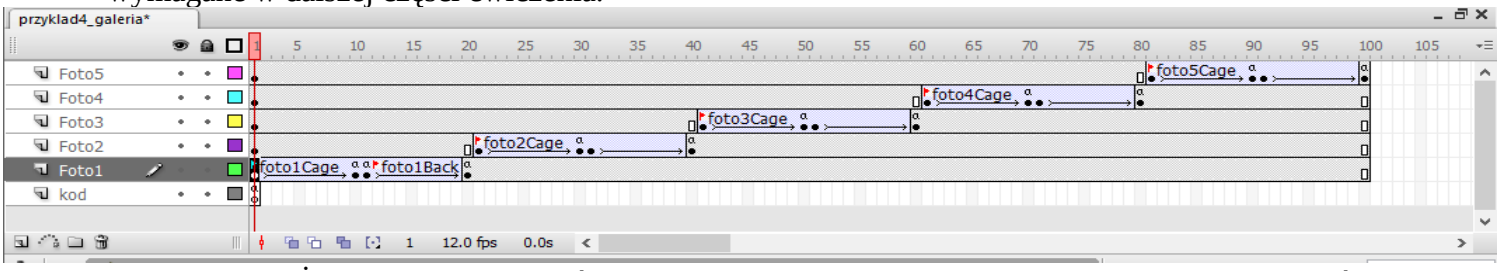


h) teraz dodajemy jeszcze jedną warstwę. Ma być pusta (bez żadnych obrazków czy innych elementów). Teraz BARDZO WAŻNE – każdemu elementowi na warstwie (każdemu symbolowi trzeba nadać odpowiednią nazwę. Nazwę dodaje się poprzez wybranie danego symbolu i wpisuje się ją w polu dostępnym we Właściwościach (pokazuje to poniższy zrzut):



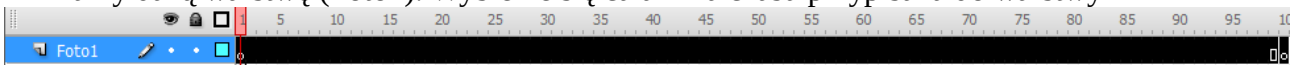
Każda nazwa musi być unikatowa.

i) kiedy każdy obrazek otrzyma już swoją nazwę możemy przystąpić do edycji linii czasu i warstw. Ostatecznie powinna ona wyglądać tak jak na poniższym obrazku. Proszę zwrócić uwagę na dwie klatki kluczowe umieszczone mniej więcej w połowie ruchu na danej warstwie (symbole animujemy przy pomocy ruchu, nie kształtu). Dodatkowo niektóre klatki mają symbol litery a (czyli znajduje się w nich ActionScript). Proszę nie sugerować się etykietami klatek kluczowych – nie są wymagane w dalszej części ćwiczenia.



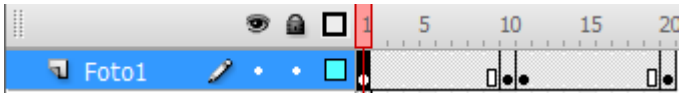
BARDZO WAŻNE! Proszę pamiętać o ustawieniu w klatkach kluczowych odpowiednich wartości dla każdego z symboli. Najlepiej byłoby utworzyć takie animacje, gdzie w połowie animacji danej warstwy symbol powiększa się na całą dostępną przestrzeń po czym ponownie się kurczy. Jak tego dokonać (zakładając, że tworzymy animację dla pierwszego obrazka - Foto1):

- klikamy daną warstwę (Foto1). Wybierze się cała linia czasu przypisana do warstwy



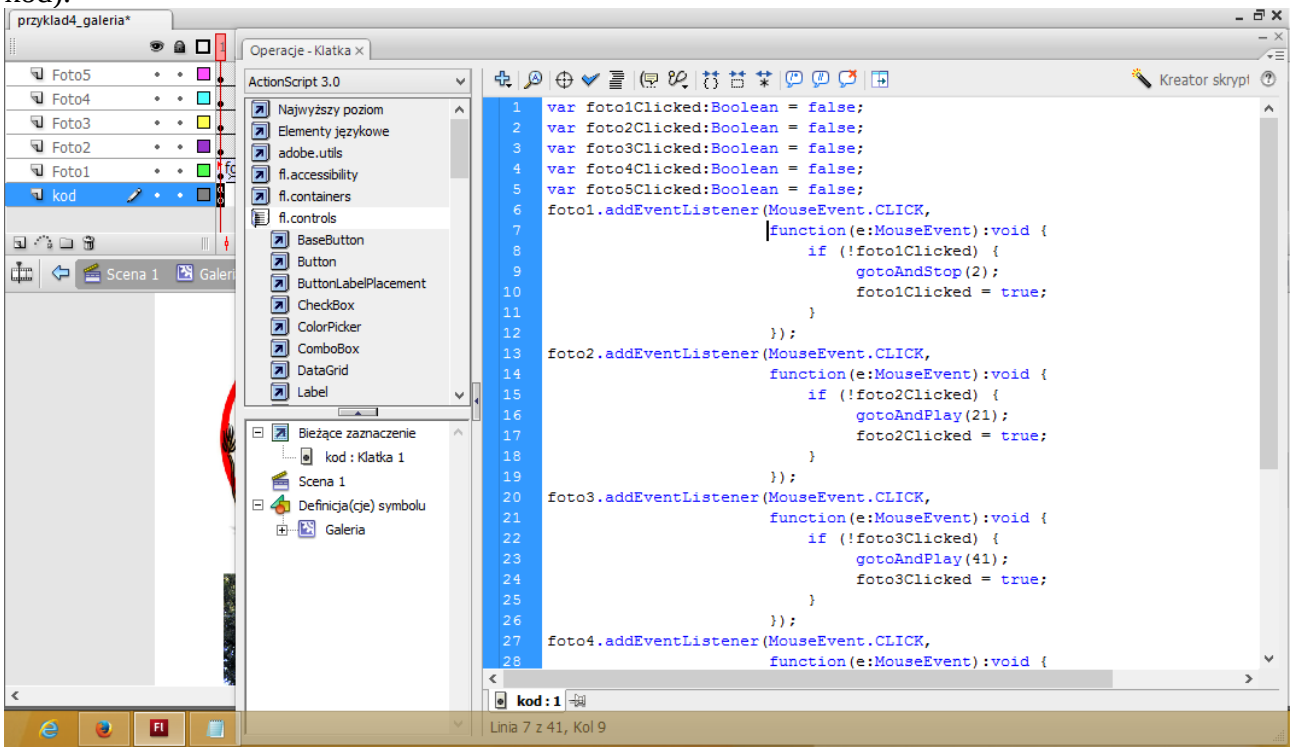
oczywiście możemy mieć w danej warstwie tylko jedną klatkę (kluczową). Wtedy zaznaczenie będzie wyglądać tak:

- jeżeli mamy więcej klatek wtedy wszystkie usuwamy (po prostu zaraz po zaznaczeniu warstwy klikamy prawym przyciskiem myszy w dowolne miejsce z zaznaczonymi klatkami i wybieramy Usuń klatkę). Klikamy na linii warstwy np. na klatkę 20 i wstawiamy klatkę kluczową. Dzięki temu zabiegowi klatka kluczowa pod numerem 20 będzie miała identyczną zawartość jak klatka spod numeru 1 (brak potrzeby kopiowania/ustawiania czegokolwiek.
- teraz klikamy w połowie naszej animacji (10) i ustawiamy klatkę kluczową. Najlepiej od razu kliknąć i ustawić klatkę kluczowa w ramce obok (11). Po naszej operacji powinniśmy mieć taką postać linii czasu:



- przechodzimy do klatki numer 10 (wybieramy ją). Ustawiamy w niej rozmiar i pozycję naszego symbolu tak by zajmował jak największą (a nawet całą) powierzchnię animacji. Teraz bardzo ważne: kopiujemy zawartość edytowanej klatki 10 do klatki numer 11 – klikamy w tym celu prawym przyciskiem myszy na klatce 10, wybieramy z menu Kopiuj klatki a następnie klikajmy na prawym przyciskiem myszy na klatkę 11 i wybieramy Wklej klatki (STANDARDOWE kopiuj/wklej nie zawsze działa!) Operację ponawiamy na kolejnych warstwach

Zaczynamy od edycji kodu dostępnego pod klatką kluczową numer 1 (pusta klatka na warstwie kod).



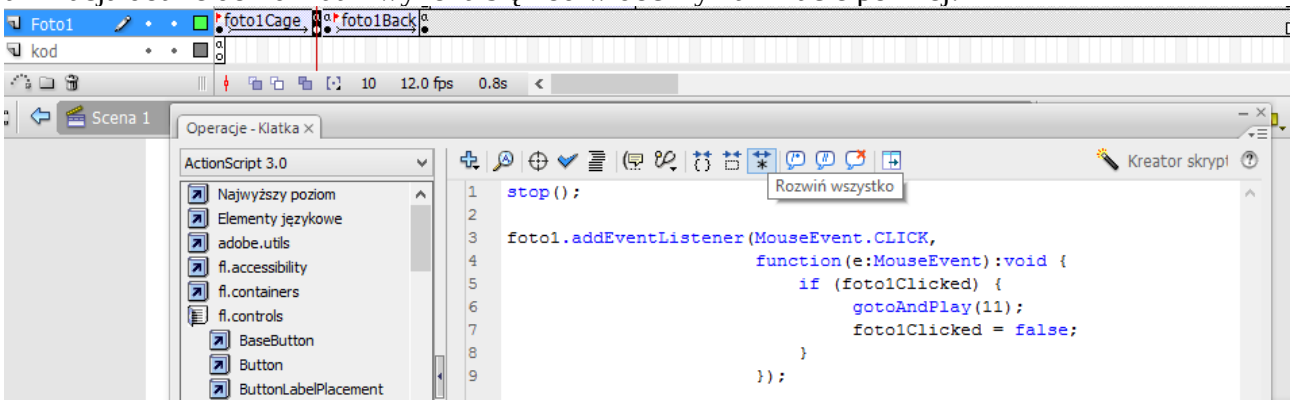
Na początek zainicjowane zostały zmienne typu prawda/fałsz z domyślnymi wartościami false (fałsz). Mają one za zadanie nie dopuścić do ponownego kliknięcia w zdjęcie, które zostało aktualnie wybrane przez użytkownika (bez tej ochrony każde kolejne kliknięcie na wybranym zdjęciu powodowałoby rozpoczęcie animacji na nowo!). Następnie definiowana jest obsługa zdarzeń dla każdego symbolu w animacji (proszę pamiętać iż kod definiowany dla jednej klatki animacji odnosi się do całej animacji). Proszę zauważyć iż zdarzenie działa dla każdego symbolu tylko wtedy, gdy odpowiednia zmienna Boolean ma wartość fałszu. W takim wypadku następuje przeskok do odpowiedniej klatki animacji i jej właściwy start. Sama zmienna ustawiana jest na wartość prawdy (true). Będzie zmieniana w skrypcie podpiętym pod inną klatkę animacji.
Pełny kod dostępny pod tą klatką:

```
var foto1Clicked:Boolean = false;
var foto2Clicked:Boolean = false;
var foto3Clicked:Boolean = false;
var foto4Clicked:Boolean = false;
var foto5Clicked:Boolean = false;
foto1.addEventListener(MouseEvent.CLICK,
                                        function(e:MouseEvent):void {

                                                if (!foto1Clicked) {
                                                        gotoAndPlay(2);

                                                        foto1Clicked = true;
                                                }
                                        });
foto2.addEventListener(MouseEvent.CLICK,
                                        function(e:MouseEvent):void {
                                                if (!foto2Clicked) {
                                                        gotoAndPlay(21);
```

```
                                        foto2Clicked = true;
                                }
                        });
foto3.addEventListener(MouseEvent.CLICK,
                        function(e:MouseEvent):void {
                                if (!foto3Clicked) {
                                        gotoAndPlay(41);
                                        foto3Clicked = true;
                                }
                        });
foto4.addEventListener(MouseEvent.CLICK,
                        function(e:MouseEvent):void {
                                if (!foto4Clicked) {
                                        gotoAndPlay(61);
                                        foto4Clicked = true;
                                }
                        });
foto5.addEventListener(MouseEvent.CLICK,
                        function(e:MouseEvent):void {
                                if (!foto5Clicked) {
                                        gotoAndPlay(81);
                                        foto5Clicked = true;
                                }
                        });
stop();
```

Jak widać na końcu kodu znajduje się odwołanie do funkcji stop(). Dlatego animacji nie przechodzi na następne klatki (domyślnie jest zatrzymana).

Przyjrzyjmy się jak działa galeria na podstawie jednego z symboli (elementu galerii). Gdy animacja dotrze do 10 klatki wykona się kod widoczny na zrzucie poniżej:



Po pierwsze widać, że na tej klatce animacja zatrzyma się (funkcja stop();). Po drugie dodajemy nasłuchiwanie kliknięcia myszą na właśnie powiększonym obrazie. Jeżeli animacja wykryje kliknięcie na tym elemencie (oraz wcześniej została ustawiona zmienne foto1Clicked na true) to przeniesie się do ramki numer 11 i rozpocznie animację. Ponadto zmieni wartość zmiennej foto1Clicked na false (zabezpieczy przed wielokrotnym kliknięciem na klatkę w tym momencie). Pełny kod spod tej animacji:

```
stop();

foto1.addEventListener(MouseEvent.CLICK,
                        function(e:MouseEvent):void {

                                if (foto1Clicked) {
```

```
                                        gotoAndPlay(11);
                                        foto1Clicked = false;
                                }
                        });
```

Ostatnią klatką zawierającą kod jest ramka 20. W niej znajduje się tylko jedna funkcja:

gotoAndStop(1);

mówiąca scenie by ustawiła głowicę odtwarzającą na pierwszą klatkę i się zatrzymała (punkt wyjścia).
Następne warstwy zawierają analogiczny kod (zmienia się nazwa symbolu nasłuchującego zdarzenia oraz numer klatek, do których następuje przeskok).

Po uruchomieniu aplikacji okaże się jednak, że coś jest nie tak. Przykładowo gdy wybierzemy pierwszy symbol animacja będzie wyglądać tak:

Niestety nie jest to błąd. Flash ustawia kolejne warstwy jedna nad drugą (tak samo jak np. PhotoShop czy Corel Draw). Dlatego warstwa będącą „niżej” niż pozostałe będzie przez nie przykryta – tak się dzieje w tym przypadku.

Adobe Flash Professional pozwala co prawda przesuwać symbole pomiędzy poszczególnymi warstwami jednak działa to jedynie w przypadku, gdy niczego nie animujemy. W przeciwnym przypadku cała animacja zaczyna zachowywać się w niekontrolowany sposób – niektóre elementy zanikają, niektóre w ogóle się nie animują, a jeszcze inne się klonują (dodają niejawnie do listy wyświetlania). Niestety to zachowanie obecne jest nawet w najnowszych wersjach narzędzia (i zapewne pozostanie nierozwiązane). W celu zobaczenia jak wygląda przesuwanie po warstwach poszczególnych elementów proszę wykonać następujący kod (dla pierwszego elementu):

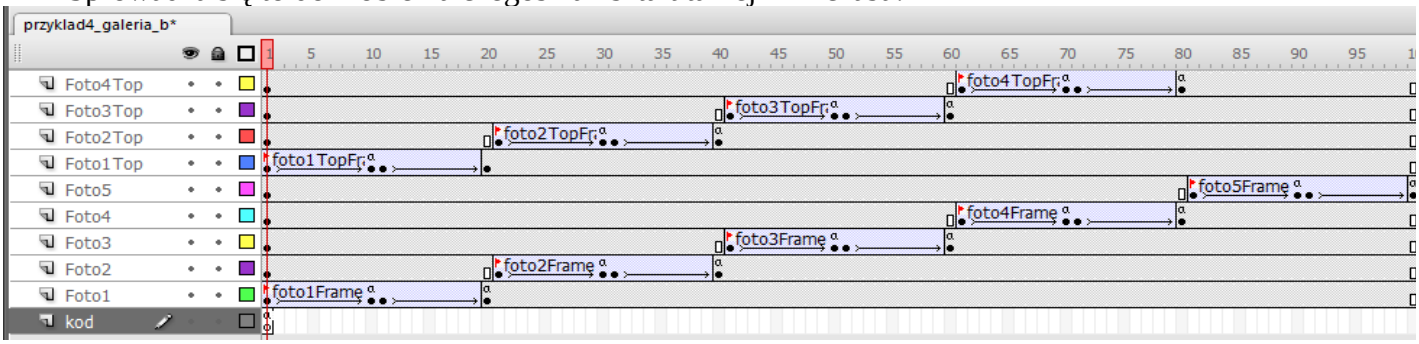
```
foto1.addEventListener(MouseEvent.CLICK,
```

```
function(e:MouseEvent):void {

        if (!foto1Clicked) {
            setChildIndex(foto1, numChildren - 1);
            foto1Clicked = true;
            gotoAndPlay(2);
        }
});
```

Wytłuszczona linia została dodana do głównego kodu nasłuchującego w animacji. W teorii zamienia ona pozycję symbolu foto1 na najwyższą z możliwych (zmienna numChildren przechowuje ilość wszystkich elementów dostępnych w danej scenie; ponieważ tablica indeksowa elementów jest zawsze numerowana od 0 należy od tej liczny odjąć 1 – bez odjęcia spowodowalibyśmy błąd wykonywania skryptu). Funkcja, według dokumentacji sama dba o przemieszczenie aktualnie najwyższego elementu na miejsce przenoszonego. W praktyce okazuje się, że funkcja DUPLIKUJE element przenoszony i dodaje go jako nowy, niezależny element. Niestety żadne zabiegi opisane jako walkaround (obejście) nie pomogły w używanej wersji – każdorazowo element zwiększa wielkość listy elementów (można to sprawdzić przy użyciu funkcji trace(numChildren);).

4. Naprawa utworzonej animacji.

Naprawy galerii można dokonać na kilka różnych sposobów. Najprostszym będzie po prostu sklonowanie warstw będących niżej i ustawienie ich klonów ponad elementami je przykrywającymi.
Sprowadza się to do zrobienia czegoś na kształt takiej linii czasu:



Proszę pamiętać iż nazwy warstw muszą być unikatowe. To samo tyczy się etykiet ramek na linii czasu. Ponadto KONIECZNIE trzeba zmieniać nazwy wszystkim symbolom użytym na wyższych warstwach (bez tego animacja zacznie się zachowywać w niekontrolowany sposób!).
Modyfikacjom uległ kod w głównej klatce warstwy o nazwie „kod”. Przykładowo nasłuch dla pierwszego obrazu wygląda teraz następująco:

```
foto1.addEventListener(MouseEvent.CLICK,
                function(e:MouseEvent):void {

        if (!foto1Clicked) {
            foto1.visible = false;
            foto1Top.visible = true;
            gotoAndPlay(2);
            foto1Clicked = true;
        }
});
```

Dodane zostały dwie wytłuszczone linie. Pierwsza z nich wyłącza wyświetlanie (widoczność) elementu, na który kliknęliśmy (żeby nie powielił się na animacji). Druga linia uwidacznia klon tegoż obrazu dostępny w sklonowanej warstwie (dla potrzeb animacji została mu nadana po prostu

nazwa foto1Top). Reszta kodu pozostała bez zmian. W ramce 10 warstwy Foto1 kod wygląda tak:
stop();

i to wszystko. Po prostu animacja zatrzymuje się. Lecz to nie koniec. W warstwie Foto1Top również znajduje się kod:

```
foto1Top.addEventListener(MouseEvent.CLICK,
                                        function(e:MouseEvent):void {
                                                if (foto1Clicked) {
                                                        gotoAndPlay(11);
                                                        foto1Clicked = false;
                                                }
                                        });
```

to on odpowiada za odwrotną animację, tj. zmniejszenie obrazu i powrotu na swoje miejsce.
Z kolei w ramce 20 warstwy Foto1 odnajdziemy taki kod:

```
gotoAndStop(1);
foto1.visible = true;
foto1Top.visible = false;
```

czyli wracamy do pierwszej ramki animacji i ukrywamy symbol z warstwy Foto1Top, a pokazujemy na powrót symbol z warstwy Foto1. Analogicznie sprawa wygląda w pozostałych warstwach za wyjątkiem warstwy Foto5. Jej kod w ogóle nie uległ zmianie gdyż nie zachodziła taka konieczność – warstwa ta zawsze była na wierchu i jako taka przykrywa wszystkie pozostałe toteż nie musi posiadać swojego klona, który przykryłby wszystkie pozostałe elementy.
W ramce warstwy „kod” pełen od skryptu wygląda następująco:

```
var foto1Clicked:Boolean = false;
var foto2Clicked:Boolean = false;
var foto3Clicked:Boolean = false;
var foto4Clicked:Boolean = false;
var foto5Clicked:Boolean = false;

foto1.addEventListener(MouseEvent.CLICK,
                                        function(e:MouseEvent):void {

                                                if (!foto1Clicked) {
                                                        foto1.visible = false;
                                                        foto1Top.visible = true;
                                                        gotoAndPlay(2);
                                                        foto1Clicked = true;
                                                }
                                        });

foto2.addEventListener(MouseEvent.CLICK,
                                        function(e:MouseEvent):void {
                                                if (!foto2Clicked) {
                                                        foto2.visible = false;
                                                        foto2Top.visible = true;
                                                        gotoAndPlay(21);
                                                        foto2Clicked = true;
                                                }
```

```
                                        });

foto3.addEventListener(MouseEvent.CLICK,
                                        function(e:MouseEvent):void {
                                                if (!foto3Clicked) {
                                                        foto3.visible = false;
                                                        foto3Top.visible = true;
                                                        gotoAndPlay(41);
                                                        foto3Clicked = true;
                                                }
                                        });

foto4.addEventListener(MouseEvent.CLICK,
                                        function(e:MouseEvent):void {
                                                if (!foto4Clicked) {
                                                        foto4.visible = false;
                                                        foto4Top.visible = true;
                                                        gotoAndPlay(61);
                                                        foto4Clicked = true;
                                                }
                                        });

foto5.addEventListener(MouseEvent.CLICK,
                                        function(e:MouseEvent):void {
                                                if (!foto5Clicked) {
                                                        gotoAndPlay(81);
                                                        foto5Clicked = true;
                                                }
                                        });
stop();
foto1Top.visible = false;
foto2Top.visible = false;
foto3Top.visible = false;
foto4Top.visible = false;

if (!foto1.visible)
        foto1.visible = true;
if (!foto2.visible)
        foto2.visible = true;
if (!foto3.visible)
        foto3.visible = true;
if (!foto4.visible)
        foto4.visible = true;
```

Wszystkie zmiany względem poprzedniej wersji zostały pogrubione. Bardzo ważne jest by w kodzie znalazły się linie dodane po funkcji stop();.

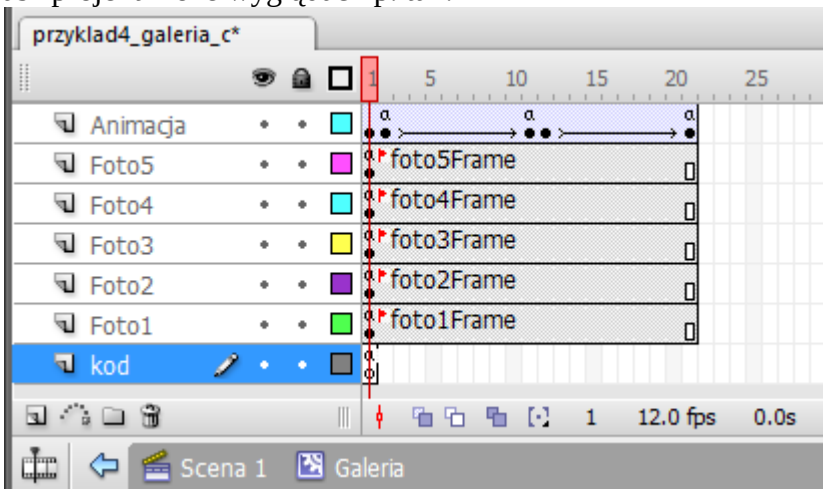
Pierwsze 4 odpowiadają za ukrycie elementów nadrzędnych (jeżeli jeszcze nie zostały ukryte). Mogą one się nie ukryć np. w przypadku, gdy użytkownik od razu kliknie następne zdjęcie. Instrukcje warunkowe mają natomiast wymusić pojawienie się wszystkich obrazów z warstw niższych (również bez nich mogą w losowych przypadkach zniknąć z animacji). Oczywiście najlepszym sposobem na przekonanie się o ważności tychże linii jest ich zakomentowanie i uruchomienie animacji – prędzej czy później natrafi się na „błędy”, którym zapobiegają.

5. Usprawnienie animacji galerii

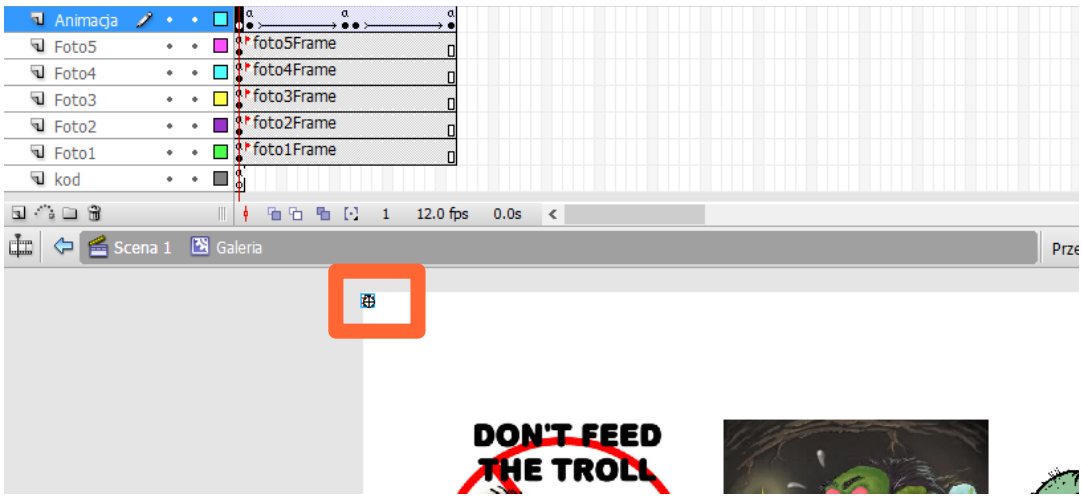
Przedstawiony powyżej pomysł sprawdzi się bez większych problemów. Niestety ma on kilka wad:
- powoduje potrzebę dodawania dwóch nowych warstw dla kolejnych obrazów w galerii
- wymaga klonowania istniejących elementów
- projekt staje się nieczytelny przy20 zdjęciach (nawet przy 10)
- metoda jest nieefektywna

Jak można zaradzić powyższym problemom? Zmienić graficzną prezentację na kod!

Po pierwsze należy zmienić sposób umieszczenia poszczególnych symboli w projekcie. Niech wszystkie elementy będą dostępne w pierwszej klatce kolejnych warstw. Nie powinny one więcej posiadać swoich animacji – mają po prostu stale wyświetlać się w jednej pozycji. Od animacji powinna być jedna warstwa, dostępna zawsze jako najwyższa. Oczywiście warstwa kodu może jak najbardziej pozostać (przyda się).
Po przebudowie nasz projekt może wyglądać np. tak:



O wiele schludniej niż poprzednio. Teraz nawet 50 obrazów nie stanowi większego problemu. Jak jednak animowany jest ruch zdjęć? Za wszystko niemal odpowiada kod oraz jeden, dodatkowy element dostępny w najnowszej warstwie.
Po pierwsze do biblioteki trzeba dodać nowy symbol (np. poprzez opcje Powiel... dostępną po wybraniu prawym przyciskiem myszy dowolnego, innego symbolu – dla potrzeb przykładu wybrany został symbol Foto1)
Następnie został on wstawiony do nowej warstwy „Animacja”. Jego wysokość i szerokość w pierwszej klatce została ustawiona na 10/10. Zawieszony został w lewym górnym rogu całej animacji (zaznaczony obramowaniem):



W zasadzie niemal niewidoczny. Z kolei w ramce numer 10 ma on mieć taki rozmiar jaki miały poszczególne symbole w poprzednich dwóch przykładach. W ostatniej, 20 ramce ma on być z powrotem mały (by animacja była płynna). Dodatkowo powinien posiadać ramkę kluczową w drugiej ramce animacji (prócz tej w ramce 11).
Kod w ramce 1 warstwy „kod” przedstawia się następująco:

```
var clickAvailable:Boolean = true;

function eventClick(e:MouseEvent):void {
	if (clickAvailable) {
		gotoAndPlay(2);
		clickAvailable = false;
	}
}

foto1.addEventListener(MouseEvent.CLICK, eventClick);

foto2.addEventListener(MouseEvent.CLICK, eventClick);

foto3.addEventListener(MouseEvent.CLICK, eventClick);

foto4.addEventListener(MouseEvent.CLICK, eventClick);

foto5.addEventListener(MouseEvent.CLICK, eventClick);
stop();
fotoFrame.visible = false;
fotoFrame.getChildAt(0).visible = false;
while (fotoFrame.numChildren > 1)
	fotoFrame.removeChildAt(fotoFrame.numChildren - 1);
```

Jak widać uległ znacznemu uproszczeniu. Posiada tylko jedną zmienną Boolean która mówi, czy użytkownik może klikać na miniaturach zdjęć, czy musi kliknąć na powiększonym zdjęciu. Ponieważ tylko jeden element jest animowany toteż kod funkcji zdarzenia znacznie się uprościł. Dlatego można było go zapisać w jednej funkcji i powielić na wszystkie zdarzenia (użyto funkcji nazwanej, której to nazwy użyto przy parametrze callback funkcji nasłuchującej). Element z warstwy „Animacja” (wyświetlający powiększenie obrazu) został nazwany fotoFrame. W ramce pierwszej jest on ukrywany (visible = false). Dodatkowo ukrywany jest pierwszy element-dziecko tegoż symbolu. Ponieważ element fotoFrame ma w późniejszych etapach kodu dodawane nowe

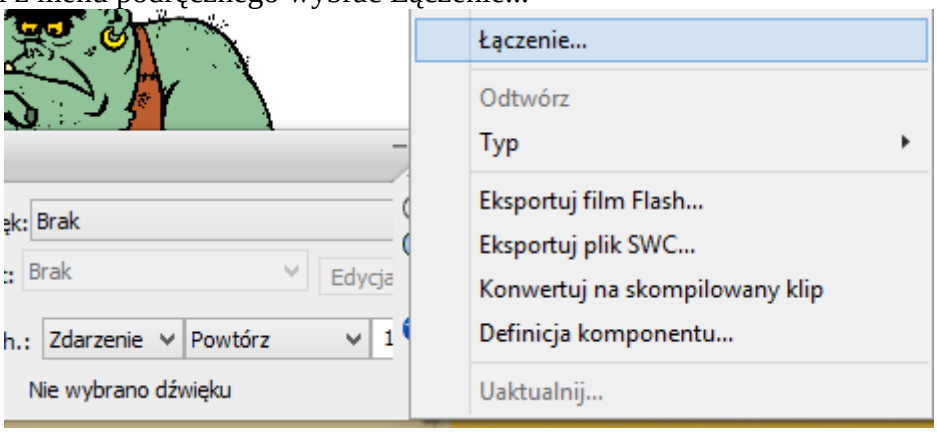
elementy do swojej animacji, pętla powtórzeniowa while dba by zostały w tym miejscu skasowane wszystkie elementy z listy poza pierwszym, domyślnym.

Każda kolejna ramka zawiera kod, dzięki któremu zawarte w nich poszczególne symbole nasłuchują kliknięcia. Oczywiście można było te operacje przenieść do warstwy z całym kodem jednak poprzez takie rozwiązanie zwiększa się przejrzystość kodu – w poprzedniej warstwie znajduje się kod ogólny dla całej animacji, a w poszczególnych warstwach kod dotyczący tylko ich elementów (w tym przypadku jednego, konkretnego). Przyjrzyjmy się zawartości kodu w pierwszej warstwie (Foto1):

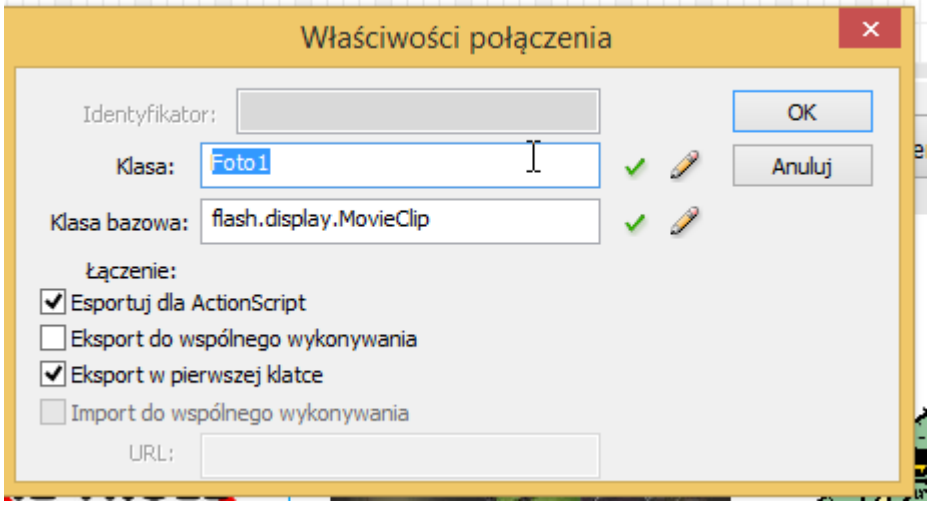
```
foto1.addEventListener(MouseEvent.CLICK,
                       function(e:MouseEvent):void {
                                  var newBitmap1:Foto1 = new Foto1();
                                  fotoFrame.addChild(newBitmap1);
                       });
```

Kod ten, w przypadku kliknięcia myszą na nasz obiekt, tworzy nowy element (obiekt) z elementu Foto1 i dodaje go do elementu fotoFrame (poprzez metodę addChild).

WAŻNE: nazwa Foto1 jako taka nie istniej w Actioncript. W opisywanym przypadku to nazwa klasy obiektu z biblioteki symboli. Żeby móc odwołać się do tej nazwy najpierw trzeba powiedzieć programowi Flash, że chcemy takowy obiekt na podstawie danego symbolu utworzyć. W tym celu trzeba kliknąć prawym przyciskiem myszy na symbolu, który ma zostać wyeksportowany do ActionScript i z menu podręcznego wybrać Łączenie...



Pojawi się okno w którym trzeba zaznaczyć opcję „Eksportuj dla ActionScript”. Automatycznie zaznaczy się „Eksportuj w pierwszej klatce”; dzięki temu z obiektu będzie można korzystać od początku animacji (a nie od czasu jego późniejszej deklaracji). Teraz tylko trzeba podać nazœ klasy oraz nazwę klasy bazowej (program Flash automatycznie je dla nas uzupełni – nie trzeba nic zmieniać). Można kliknąć na zielone odznaczenia – potwierdzi to narzędziu iż to właśnie takie nazwy chcemy wyeksportować do użytku w kodzie. Później klikamy przycisk OK, potwierdzamy wiadomość (o domyślnym renderowaniu klasy) i gotowe – możemy tworzyć nowe obiekty w oparciu o nasz symbol z biblioteki.

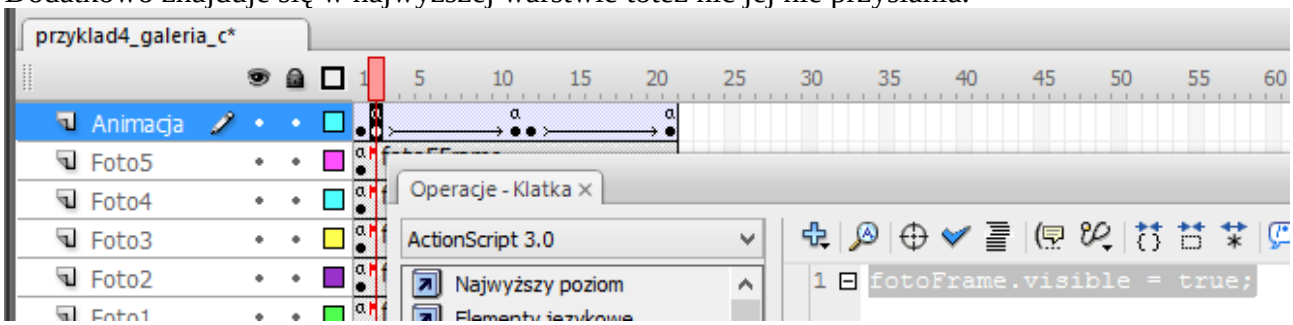


Analogicznie postępujemy z pozostałymi symbolami (MovieClip) w naszej bibliotece. Następnie ustawiamy analogicznie kod w kolejnych warstwach animacji (dla wszystkich obrazów).

W ramce 2 warstwy „Animacja” wklejamy taką oto linijkę:

```
fotoFrame.visible = true;
```

Od tego momentu nasza ramka z powiększającym się zdjęciem jest widoczna na animacji. Dodatkowo znajduje się w najwyższej warstwie toteż nic jej nie przysłania.



Ramka 11 zawiera następujący kod:

```
stop();

fotoFrame.addEventListener(MouseEvent.CLICK,
                                    function(e:MouseEvent):void {
                                                gotoAndPlay(12);
                                    });
```

Gdy użytkownik kliknie na fotoFrame to rozpocznie się animacja od klatki numer 12.

W ramce 21 natomiast znajduje się jedna linia kodu:

```
gotoAndStop(1);
```

Wszystko. Animacja powinna działać bez zarzutu. Jej projekt jest o wiele bardziej przejrzysty i ewentualne zmiany w nim nie będą powodować większych problemów z przegrupowaniem warstw czy kodu.

INFORMACJA: Może się zdarzyć, że niektóre obrazy dodawane w ten sposób do animacji mogą mieć nietypowy wymiar, przez co zepsuć wyświetlanie. Ponadto może zdarzyć się, że będą one miały przesunięty punkt początku rysowania (XY). Niedogodności te można naprawić poprzez przesunięcie tegoż punktu do wskazanego np. za pomocą kodu:

```
var newBitmap5:Foto5 = new Foto5();
newBitmap5.x = -100;
newBitmap5.y = -130;
fotoFrame.addChild(newBitmap5);
```

Należy tylko pamiętać, że zmiany tej trzeba dokonać przed dodaniem symbolu do fotoFrame!
Z kolei rozmiar maksymalnego powiększenia można rozwiązać poprzez dodanie kodu w ramce maksymalnego powiększenia:

```
var obj = fotoFrame.getChildAt(1);
obj.width = fotoFrame.getChildAt(0).width;
```

```
obj.height = fotoFrame.getChildAt(0).height;
```

Kod ma za zadanie pobrać element dodany do naszego fotoFrame, po czym ma rozciągnąć istniejący w num element do wymiarów elementu nadrzędnego.

Proszę zapoznać się kodem przykładu dołączonego do tego materiału.

6. Ładowanie grafik zewnętrznych

Adobe Flash pozwala także na ładowanie grafik nie należących do biblioteki danej animacji. Możemy bezpośrednio w kodzie dodać ścieżkę do zdjęcia, jakie ma zostać załadowane. W celu prezentacji skopiowany został obrazek png do katalogu animacji swf. Wykorzystany został kod poprzednio opisanej animacji (z punktu numer 5). Dla ramki w warstwie Foto1 został podmieniony kod na następujący:

```
foto1.addEventListener(MouseEvent.CLICK,
	function(e:MouseEvent):void {
		imgLoader = new Loader();
		imgLoader.contentLoaderInfo.addEventListener(Event.INIT,initImage);
		imgLoader.load(new URLRequest("not-sure-if-troll.png"));
	});

function initImage(evt:Event):void
{
	imgLoader.content.x = -80;
	imgLoader.content.y = -120;
	imgLoader.content.scaleX = 0.5;
	imgLoader.content.scaleY = 0.5;
	fotoFrame.addChild(imgLoader.content);
}
```

Tym razem funkcja nasłuchująca ma za zadanie utworzyć nowy obiekt w zmiennej imgLoader. Następnie dodajemy nasłuchiwanie nowego obiektu na zdarzenie początku wczytywania zdjęcia do tegoż elementu. Na koniec mówimy animacji jaki plik ma wczytać.
Funkcja initImage() ma za zadanie dodawać zdjęcie do naszej ramki. Proszę zauważyć, że zdjęcie to musiało mieć przesunięty punkt XY oraz musiało zostać przeskalowane (w tym wypadku zmniejszone) o 50% (0.5). W innym wypadku wyświetlało się w dolnym rogu animacji z wymiarami dwukrotnie przekraczającymi oczekiwany wymiar (wartości zostały dobrane eksperymentalnie!).
Użyta zmienna imgLoader została z kolei zainicjowana w pierwszej ramce warstwy „kod”:

```
var clickAvailable:Boolean = true;
var imgLoader:Loader;
```

Bez tego zabiegu nie byłaby ona widoczna w całym kodzie (we wszystkich warstwach) a tylko w pojedynczej funkcji (w tym wypadku nasłuchującej dla obrazka 1).

ZADANIA:

1. Proszę stworzyć własną galerię z ok. 8 zdjęciami
2. Proszę zoptymalizować ją tak jak podano w punkcie 5.
3. Proszę ładować obrazki spoza biblioteki (niech Flash ładuje zdjęcia z danego katalogu, gdzie

kolejne obrazy będą miały nazwy 1.png, 2.png itd...). Proszę zobaczyć co się stanie gdy obrazy zostana podmienione (będą mieć jednak te same nazwy).

Wykorzystano strony:
http://www.codingcolor.com/as3/how-do-you-load-an-image-into-an-empty-movieclip-in-as-3-0/
http://www.senocular.com/flash/tutorials/as3withflashcs3/?page=2
Przewijanie treści w Adobe Flash Professional (dodany nieopisany przykład e):
http://www.entheosweb.com/Flash/scrolling_content.asp