Radosław Kamysz

FLASH I ACTIONSCRIPT

Aplikacje 3D od podstaw

Helion

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Redaktor prowadzący: Michał Mrowiec

Projekt okładki: Studio Gravite / Olsztyn
Obarek, Pokoński, Pazdrijowski, Zaprucki

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie?flacpo
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Kody źródłowe wybranych przykładów dostępne są pod adresem:
ftp://ftp.helion.pl/przyklady/flacpo.zip

ISBN: 978-83-246-3065-3

Copyright © Helion 2013

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Rozdział 8.
Interaktywność

Jedną z głównych zalet pisania aplikacji na platformę Flash jest to, że można tworzyć ciekawe interaktywne gry i strony internetowe. W tego typu projektach większość operacji wykonywanych jest za pomocą myszki, klawiatury i coraz częściej kamer. Jeżeli masz jakieś doświadczenie w realizacji tego typu projektów, z pewnością łatwo połączysz zdobytą wiedzę z informacjami zawartymi w tym rozdziale. Jeżeli są to Twoje początki z aplikacjami tworzonymi na platformę Adobe Flash Player, nie przejmuj się, na tym etapie wiesz już wystarczająco dużo.

Tym, co łączy zwykłe dwuwymiarowe aplikacje z projektami bazującymi na bibliotece Away3D, jest to, że w obu przypadkach można korzystać z urządzeń takich jak mysz czy klawiatura. Stosując szereg standardowych metod, zdarzeń z ActionScript 3.0 oraz Away3D, można zaprogramować odpowiednie operacje, które wywoływane będą przez ingerencję użytkownika.

Czytając ten rozdział, dowiesz się:

- Jak obsługuje się zdarzenia wciśnięcia i zwolnienia klawisza klawiatury.
- Czym są i jakie role odgrywają klasy `KeyboardEvent`, `Keyboard` oraz `KeyLocation`.
- Jak sterować trójwymiarowym obiektem za pomocą klawiatury.
- Czym jest klasa `MouseEvent3D`.
- W jakich sytuacjach i jak stosować `MouseEvent` oraz `MouseEvent3D`.
- Jak przemieszczać trójwymiarowe obiekty przy użyciu myszki.
- Jak malować po powierzchni trójwymiarowych obiektów.

Kup książkę Poleć książkę

Używanie klawiatury

W większości gier i innych aplikacji niekiedy obiekty sterowane są za pomocą klawiatury. W aplikacjach Flash do wykrywania czynności wykonywanych przez użytkownika na klawiaturze stosuje się obiekt zdarzenia `KeyboardEvent`.

Klasa KeyboardEvent

Klasa `KeyboardEvent`, ponieważ reprezentuje obiekt zdarzenia, zlokalizowana jest w pakiecie `flash.events`. Ma ona dwie stałe: `KeyboardEvent.KEY_DOWN` i `KeyboardEvent.KEY_UP`, które określają przypadki wciśnięcia i zwolnienia klawisza na klawiaturze. Aby móc korzystać z tych zdarzeń, należy metodę `addEventListener` wywołać na obiekcie `stage`, tak jak to pokazano poniżej:

```
stage.addEventListener(KeyboardEvent.KEY_DOWN,onKeyDown);
stage.addEventListener(KeyboardEvent.KEY_UP,onKeyUp);
```

Samo wywołanie detektora nie wystarczy do podjęcia konkretnych akcji. W ciele metody wywoływanej podczas wystąpienia zdarzenia należy rozpoznać, który klawisz zmienił swój stan. Do tego celu służą dwie właściwości: `keyCode` i `charCode`. Pierwsza zwraca wartość numeryczną odpowiadającą numerowi klawisza, a druga określa wartość liczbową kodu znaku wciśniętego klawisza.

Istnieje zasadnicza różnica między wartościami tych właściwości. Polega ona na tym, że kod klawisza jest przypisany do konkretnego klawisza fizycznego, natomiast kod znaku jest przypisany do konkretnego znaku. Czyli jeśli na przykład chcemy wykorzystać klawisz *5* na klawiaturze numerycznej i w górnym rzędzie, należy odwołać się do wartości właściwości `charCode`.

Domyślny zestaw kodów znaków UTF-8 zawiera w swoim zbiorze identyfikatory dla każdego ze znaków z osobna, odróżnia tym samym wielkość liter.

Na klawiaturze umieszczone są również klawisze specjalne, takie jak *Ctrl*, *Alt* czy *Shift*. Do określenia, czy któryś z tych klawiszy zmienił swój stan, służą specjalne właściwości: `ctrlKey`, `altKey` i `shiftKey`. Wartości tych właściwości są typu `Boolean`, gdzie `true` oznacza, że klawisz jest wciśnięty.

Tabele 8.1, 8.2 oraz 8.3 zawierają metody, właściwości i stałe klasy `KeyboardEvent`.

Kup książkę Poleć książkę

Tabela 8.1. *Właściwości klasy KeyboardEvent*

Nazwa	Rodzaj	Wartość domyślna	Opis
`altKey`	`Boolean`	`false`	Określa, czy klawisz *Alt* jest wciśnięty
`ctrlKey`	`Boolean`	`false`	Określa, czy klawisz *Ctrl* jest wciśnięty
`controlKey`	`Boolean`	`false`	Określa, czy klawisz *Control* jest wciśnięty
`commandKey`	`Boolean`	`false`	Określa, czy klawisz *Command* jest wciśnięty
`shiftKey`	`Boolean`	`false`	Określa, czy klawisz *Shift* jest wciśnięty
`charCode`	`uint`		Wartość liczbowa kodu znaku wciśniętego klawisza
`keyCode`	`uint`		Wartość liczbowa odpowiadająca numerowi klawisza na klawiaturze
`keyLocation`	`uint`		Położenie klawisza na klawiaturze

Tabela 8.2. *Metody klasy KeyboardEvent*

Nazwa	Opis
`KeyboardEvent(type:String, bubbles:Boolean, cancelable:Boolean, charCodeValue:uint, keyCodeValue:uint, keyLocationValue:uint, ctrlKeyValue:Boolean, altKeyValue:Boolean, shiftKeyValue:Boolean, controlKeyValue:Boolean, commandKeyValue:Boolean)`	Konstruktor
`updateAfterEvent`	Określa, czy po zakończeniu przetwarzania zdarzenia obraz powinien być zrenderowany, jeżeli lista wyświetlania została zmodyfikowana

Tabela 8.3. *Stałe klasy KeyboardEvent*

Nazwa	Opis
`KEY_DOWN`	Definiuje zdarzenie wciśnięcia klawisza
`KEY_UP`	Definiuje zdarzenie zwolnienia wciśniętego klawisza

Klasa Keyboard

`Keyboard` jest klasą typu `final`, umieszczoną w pakiecie `flash.ui`. Ma ona zdefiniowane kody większości znaków w postaci stałych publicznych. Do jej metod oraz właściwości można odwoływać się bez tworzenia obiektu. Dzięki tym cechom `Keyboard` stanowi swego rodzaju słownik do stosowania w instrukcjach warunkowych takich jak `if` czy `switch`.

Kup książkę Poleć książkę

```
switch(event.charCode)
{
   case Keyboard.UP: trace('Wciśnięto klawisz górnej strzałki'); break;
   case Keyboard.DOWN: trace('Wciśnięto klawisz dolnej strzałki'); break;
   case Keyboard.LEFT: trace('Wciśnięto klawisz lewej strzałki'); break;
   case Keyboard.RIGHT: trace('Wciśnięto klawisz prawej strzałki'); break;
}
```

Tabele 8.4 i 8.5 zawierają metody i właściwości klasy `Keyboard`. Ważniejsze od nich są jednak stałe umieszczone w tabeli 8.6. `Keyboard` ma dużą liczbę stałych, dlatego przedstawiono tylko niektóre z nich.

Tabela 8.4. *Właściwości klasy Keyboard*

Nazwa	Rodzaj	Wartość domyślna	Opis
`capsLock`	`Boolean`	`false`	Określa, czy klawisz *Caps Lock* jest wciśnięty
`hasVirtualKeyboard`	`Boolean`	`false`	Określa, czy udostępniona jest klawiatura wirtualna
`numLock`	`Boolean`	`false`	Określa, czy klawisz *Num Lock* jest wciśnięty
`physicalKeyboardType`	`String`		Określa rodzaj klawiatury

Tabela 8.5. *Metody klasy Keyboard*

Nazwa	Opis
`isAccessible():Boolean`	Określa, czy ostatnio wciśnięty klawisz jest dostępny dla innych plików SWF

Tabela 8.6. *Stałe klasy Keyboard*

Nazwa	Opis
`A … Z`	Klawisze alfabetyczne
`NUMBER_0 … NUMBER_9`	Klawisze numeryczne umieszczone w górnym rzędzie
`NUMPAD_0 … NUMPAD_9`	Klawisze numeryczne umieszczone na klawiaturze numerycznej
`F1 … F12`	Klawisze funkcyjne
`ENTER`	Klawisz *Enter*
`ESCAPE`	Klawisz *Esc*
`CONTROL`	Klawisz *Ctrl*
`BACKSPACE`	Klawisz *Backspace*
`SPACE`	Klawisz spacji
`LEFT`	Klawisz lewej strzałki

Kup książkę Poleć książkę

Tabela 8.6. *Stałe klasy Keyboard (ciąg dalszy)*

Nazwa	Opis
RIGHT	Klawisz prawej strzałki
UP	Klawisz górnej strzałki
DOWN	Klawisz dolnej strzałki

Klasa KeyLocation

W tabeli 8.1 punktu „Klasa KeyboardEvent" wspomniano o tym, że właściwość `keyLocation` określa położenie wciskanego lub zwalnianego klawisza. W sytuacji gdy dany klawisz ma swoje duplikaty w innych miejscach na klawiaturze, zastosowanie tej właściwości pozwala na odróżnienie na przykład lewego i prawego klawisza *Ctrl*, co umożliwia przypisanie każdemu innych działań. Do określenia wartości właściwości `keyLocation` klasy `KeyboardEvent` służą stałe klasy `KeyLocation`.

Zlokalizowana w pakiecie `flash.ui` klasa `KeyLocation` — tak samo jak `Keyboard` — jest klasą z przypisanym atrybutem `final`. `KeyLocation` można traktować jako mały słownik, który ma w swoich zasobach określenia dostępnych pozycji na klawiaturze.

```
if(event.charCode == Keyboard.CONTROL && event.keyLocation ==
↪KeyLocation.LEFT)
{
   trace('Wciśnięto lewy Ctrl');
}
else if(event.charCode == Keyboard.CONTROL && event.keyLocation ==
↪KeyLocation.RIGHT)
{
   trace('Wciśnięto prawy Ctrl');
}else
{
   trace('Wciśnięto inny klawisz');
}
```

Ponieważ klasa `KeyLocation` nie ma swoich metod i właściwości, tylko stałe określające położenie, w tym punkcie uwzględniono jedną tabelę 8.7.

Tabela 8.7. *Stałe klasy KeyLocation*

Nazwa	Opis
D_PAD	Określa, czy wciśnięty klawisz znajduje się na panelu kierunkowym
LEFT	Określa, czy wciśnięty klawisz znajduje się po lewej stronie klawiatury
RIGHT	Określa, czy wciśnięty klawisz znajduje się po prawej stronie klawiatury

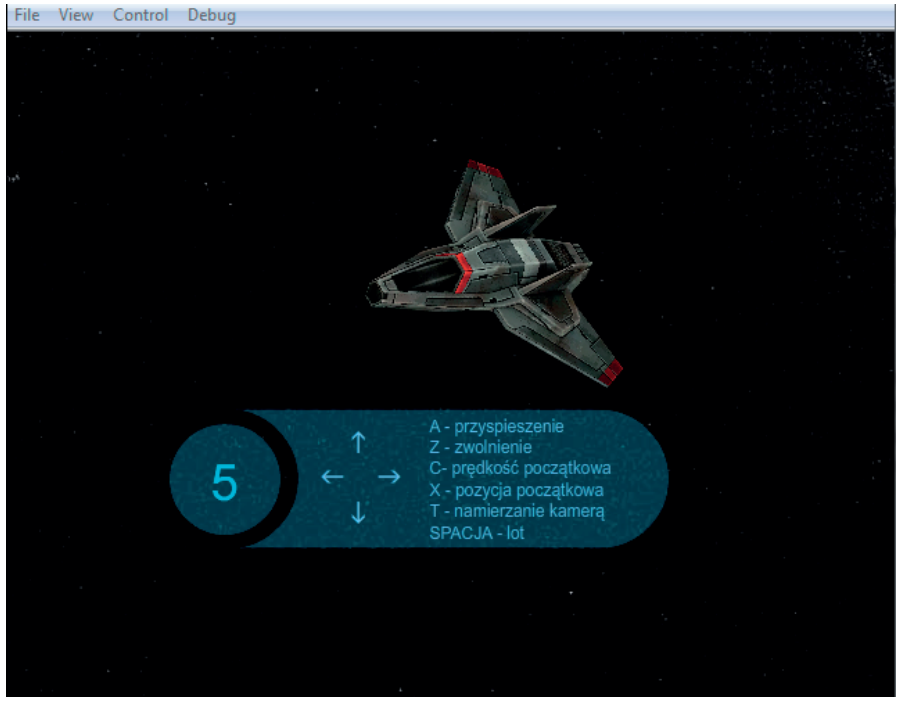
Kup książkę Poleć książkę

Tabela 8.7. *Stałe klasy KeyLocation (ciąg dalszy)*

Nazwa	Opis
NUM_PAD	Określa, czy wciśnięty klawisz znajduje się na klawiaturze numerycznej lub w jej wirtualnym odpowiedniku
STANDARD	Określa, czy pozycja klawisza nie została określona w konkretnych pozycjach

Sterowanie statkiem kosmicznym w przestrzeni

Skoro poznaliśmy klasy obsługujące interakcje użytkownika z użyciem klawiatury, przyszedł czas na zastosowanie zdobytej wiedzy w praktyce. Do tego celu wykorzystamy możliwość sterowania futurystycznym pojazdem w przestrzeni kosmicznej. Aplikacja ta składa się z dwóch warstw. Pierwsza to przestrzeń trójwymiarowa, w której porusza się pojazd, druga warstwa to element interfejsu użytkownika, w którym widoczne są aktualna prędkość pojazdu oraz spis klawiszy i akcji, jakie wywołują. Całość przedstawiono na rysunku 8.1.



Rysunek 8.1. *Zrzut ekranu z aplikacji sterowania statkiem kosmicznym*

Widoczny statek kosmiczny to jeden z darmowych modeli w formacie 3DS, które są dostępne w sieci. W pozycji początkowej pojazd ten skierowany jest dziobem w stronę osi Z. Gdy wciskamy klawisze lewej i prawej strzałki, obiekt obraca się

Kup książkę Poleć książkę

wokół osi Z. Natomiast klawiszami strzałki górnej i dolnej odpowiednio podnosi się i opuszcza dziób. Wprawienie w ruch statku kosmicznego następuje po wciśnięciu klawisza spacji. Zwroty i skręty uzależnione są od kąta nachylenia pojazdu i prędkości, z jaką się porusza. Standardowo prędkość jest równa 50, lecz można ją zwiększyć, wciskając klawisz *A*, lub zmniejszyć klawiszem *Z*. Aby szybko zredukować prędkość do jej standardowej wartości, wystarczy użyć klawisza *C*. W sytuacji gdy stracimy pojazd z pola widzenia, można przywrócić go do pozycji początkowej, wciskając klawisz *X*, lub namierzyć kamerą jego pozycję, używając klawisza *T*.

Ponieważ kamery zostaną omówione dopiero w rozdziale 9. „Kamery", skorzystamy ze standardowej, utworzonej wraz z widokiem.

Teraz przepisz i skompiluj kod źródłowy tego przykładu, a następnie zajmiemy się omawianiem jego poszczególnych fragmentów.

```
package
{
   import flash.display.StageAlign;
   import flash.display.StageScaleMode;
   import flash.display.MovieClip;
   import flash.display.Sprite;
   import flash.events.Event;
   import flash.events.KeyboardEvent;
   import flash.geom.Vector3D;
   import flash.ui.Keyboard;
   //
   import away3d.containers.ObjectContainer3D;
   import away3d.containers.View3D;
   import away3d.loaders.Loader3D;
   import away3d.loaders.Max3DS;
   import com.greensock.TweenMax;
   public class KeyboardEventsExample extends Sprite
   {
      private var view:View3D;
      private var speed:Number = 5;
      private var shipLoader:Loader3D;
      private var ship:ObjectContainer3D;
      private var leftArrowDown:Boolean = false;
      private var rightArrowDown:Boolean = false;
      private var upArrowDown:Boolean = false;
      private var downArrowDown:Boolean = false;
      private var spaceDown:Boolean = false;
      private var trackShip:Boolean = false;
      private var speedMonitor:Monitor;

      public function KeyboardEventsExample():void
      {
         if (stage) init();
         else addEventListener(Event.ADDED_TO_STAGE, init);
      }
```

```
private function init(e:Event = null):void
{
   stage.scaleMode = StageScaleMode.NO_SCALE;
   stage.align = StageAlign.TOP_LEFT;
   removeEventListener(Event.ADDED_TO_STAGE, init);
   addEventListener(Event.ENTER_FRAME, onEnterFrame);
   stage.addEventListener(Event.RESIZE, onResize);
   stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
   stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp);

   view = new View3D();
   view.x = stage.stageWidth * .5;
   view.y = stage.stageHeight * .5;
   addChild(view);
   view.camera.z = -100;

   speedMonitor = new Monitor();
   speedMonitor.txt.text = String(speed * 10);
   addChild(speedMonitor);

   loadShipModel();
   onResize();
}

private function onResize(e:Event = null):void
{
   universe.x = stage.stageWidth * .5;
   universe.y = stage.stageHeight * .5;
   view.x = stage.stageWidth * .5;
   view.y = stage.stageHeight * .5;
   speedMonitor.y = stage.stageHeight - 100;
   speedMonitor.x = 100;
}

private function loadShipModel():void
{
   ship = new ObjectContainer3D();
   shipLoader = Max3DS.load('../../resources/models/max3ds/spaceship/
   ↪spaceship.3DS');
   shipLoader.rotationY = -90;
   shipLoader.rotationX = 90;
   shipLoader.position = new Vector3D(0, 0, 0);
   ship.addChild(shipLoader);
   view.scene.addChild(ship);
}

private function onKeyDown(e:KeyboardEvent):void
{
   if (e.keyCode == Keyboard.C)
   {
      speed = 5;
      speedMonitor.txt.text = String(speed * 10);
```

Kup książkę Poleć książkę

```
            }
            if (e.keyCode == Keyboard.A)
            {
               speed += .5;
               speedMonitor.txt.text = String(speed * 10);
            }
            if (e.keyCode == Keyboard.Z)
            {
               speed -= .5;
               speedMonitor.txt.text = String(speed * 10);
            }
            if (e.keyCode == Keyboard.LEFT)  leftArrowDown = true;
            if (e.keyCode == Keyboard.RIGHT) rightArrowDown = true;
            if (e.keyCode == Keyboard.UP)    upArrowDown = true;
            if (e.keyCode == Keyboard.DOWN)  downArrowDown = true
            if (e.keyCode == Keyboard.SPACE) spaceDown = true;
         }

         private function onKeyUp(e:KeyboardEvent):void
         {
            if (e.keyCode == Keyboard.LEFT)  leftArrowDown = false;
            if (e.keyCode == Keyboard.RIGHT) rightArrowDown = false;
            if (e.keyCode == Keyboard.UP)    upArrowDown = false;
            if (e.keyCode == Keyboard.DOWN)  downArrowDown = false
            if (e.keyCode == Keyboard.SPACE) spaceDown = false;
            if (e.keyCode == Keyboard.X)
            {
               trackShip = false;
               view.camera.lookAt(new Vector3D(0, 0, 0));
               TweenMax.to(ship, 1, { x:0, y:0, z:0, rotationY:0, rotationX:0,
               ↪rotationZ:0 } );
            }

            if (e.keyCode == Keyboard.T)
            {
               trackShip = !trackShip;
            }
         }

         private function onEnterFrame(e:Event):void
         {
            if (spaceDown)      ship.moveForward(speed);
            if (leftArrowDown)  ship.roll(-5);
            if (rightArrowDown) ship.roll(5);
            if (upArrowDown)    ship.pitch(5);
            if (downArrowDown)  ship.pitch( -5);
if (trackShip) view.camera.lookAt(ship.scenePosition);
            view.render();
         }
      }
}
```

W tym przykładzie skorzystaliśmy z podstawowych klas języka ActionScript 3.0 między innymi do obsługi zdarzeń użycia klawiatury, słownika kodów klawiszy oraz wyświetlenia interfejsu.

```
import flash.display.MovieClip;
import flash.events.KeyboardEvent;
import flash.ui.Keyboard;
```

Z Away3D poza widokiem użyliśmy klas służących do ładowania i umieszczania na scenie zewnętrznych obiektów 3D. W tym konkretnym przypadku modeli w formacie 3DS.

```
import away3d.containers.ObjectContainer3D;
import away3d.containers.View3D;
import away3d.loaders.Loader3D;
import away3d.loaders.Max3DS;
```

Dodatkowo do ustawienia statku w pozycji zerowej sceny skorzystaliśmy z biblioteki `TweenMax`.

```
import com.greensock.TweenMax;
```

W klasie `KeyboardEventsExample` na początku zdefiniowaliśmy potrzebne obiekty i zmienne. Jako obiekt reprezentujący model 3D statku kosmicznego wykorzystaliśmy obiekt klasy `Loader3D` i nazwaliśmy go `shipLoader`. Wewnętrzne ustawienia pozycji modelu uzależnione są od tego, jak został on wyeksportowany. Dlatego właśnie zawartość obiektu `shipLoader` umieścimy w kontenerze o nazwie `ship`, aby w jego wnętrzu dokonywać zmian.

Obiekt o nazwie `speedMonitor` klasy `Monitor` to reprezentant obiektu interfejsu. Kolejne linijki to zmienne stosowane do obsługi pojazdu. Zmienna `speed` to stopień zmiany prędkości statku, z kolei `trackShip` określa, czy kamera ma śledzić obiekt statku, czy nie. Zmienne: `leftArrowDown`, `rightArrowDown`, `upArrowDown`, `downArrowDown` oraz `spaceDown` określają, czy wybrany klawisz został wciśnięty.

```
private var view:View3D;
private var shipLoader:Loader3D;
private var ship:ObjectContainer3D;
private var speedMonitor:Monitor;
private var speed:Number = 5;
private var trackShip:Boolean = false;
private var leftArrowDown:Boolean = false;
private var rightArrowDown:Boolean = false;
private var upArrowDown:Boolean = false;
private var downArrowDown:Boolean = false;
private var spaceDown:Boolean = false;
```

W konstruktorze uruchamiamy metodę `init()`, z chwilą gdy obiekt `stage` zostanie zainicjowany.

Kup książkę Poleć książkę

```
if (stage) init();
else addEventListener(Event.ADDED_TO_STAGE, init);
```

W metodzie `init()` w pierwszych dwóch linijkach usunęliśmy detektor zdarzenia `Event.ADDED_TO_STAGE` i utworzyliśmy nowy dla `Event.ENTER_FRAME`. W kolejnych linijkach ustawiliśmy właściwości dla okna aplikacji i dodaliśmy kolejne rejestratory zdarzeń dla obiektu `stage`. Pierwszy z nich uruchamia metodę `onResize()` z każdą zmianą rozmiarów okna. Kolejne dwa słuchają zdarzeń wciśnięcia i zwolnienia klawiszy klawiatury.

```
removeEventListener(Event.ADDED_TO_STAGE, init);
addEventListener(Event.ENTER_FRAME, onEnterFrame);
stage.scaleMode = StageScaleMode.NO_SCALE;
stage.align = StageAlign.TOP_LEFT;
stage.addEventListener(Event.RESIZE, onResize);
stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp);
```

W dalszej części metody `init()` stworzyliśmy widok Away3D i zmieniliśmy pozycję kamery na osi Z, tak aby przybliżyć ją do obiektu statku kosmicznego.

```
view = new View3D();
addChild(view);
view.camera.z = -100;
```

Po utworzeniu widoku dodaliśmy obiekt `speedMonitor`, który przedstawia listę dostępnych klawiszy i prędkość pojazdu. Wyświetloną w panelu użytkownika prędkość początkową uzyskaliśmy z iloczynu wartości zmiennej `speed` i liczby 10.

```
speedMonitor = new Monitor();
speedMonitor.txt.text = String(speed * 10);
addChild(speedMonitor);
```

Na końcu `init()` wywołujemy metodę `loadShipModel()` i `onResize()`, aby ustawić wszystkie obiekty w odpowiednim miejscu.

```
loadShipModel();
onResize();
```

W metodzie `onResize()` centrujemy widok Away3D oraz obiekt typu `movieClip` o nazwie `universe`. Obiekt `universe` to gwieździste tło, które zostało bezpośrednio umieszczone w głównym oknie aplikacji. Z kolei obiekt interfejsu użytkownika umieszczamy w lewym dolnym rogu okna.

```
universe.x = stage.stageWidth * .5;
universe.y = stage.stageHeight * .5;
view.x = stage.stageWidth * .5;
view.y = stage.stageHeight * .5;
speedMonitor.y = stage.stageHeight - 100;
speedMonitor.x = 100;
```

Kup książkę Poleć książkę

Metoda `loadShipModel()` służy do pobrania modelu z konkretnej lokalizacji i umieszczenia go na scenie. W pierwszej kolejności stworzyliśmy obiekt klasy `ObjectContainer3D` o nazwie `ship`, który będzie służył jako kontener dla pobranego modelu. Następnie pobierany jest model w formacie 3DS za pomocą klasy `Max3DS` i jej metody `load`.

Po załadowaniu zmieniliśmy kąt nachylenia statku na osi Y oraz X i ustawiliśmy jego pozycję w miejscu zerowym kontenera. Na końcu umieściliśmy obiekt na scenie.

```
ship = new ObjectContainer3D();
shipLoader = Max3DS.load('models/spaceship/spaceship.3ds');
shipLoader.rotationY = -90;
shipLoader.rotationX = 90;
shipLoader.position = new Vector3D(0, 0, 0);
ship.addChild(shipLoader);
view.scene.addChild(ship);
```

W sytuacji gdy model pochodzi z nieznanych źródeł, warto wcześniej sprawdzić w Away3D, jak prezentuje się on na scenie. Może dojść do tego, że jego skala i pozycja nie odpowiadają konkretnym potrzebom. W takiej sytuacji, jeżeli nie mamy możliwości edytowania modelu, można umieścić go w kontenerze `ObjectContainer3D` i w jego wnętrzu skorygować skalę bądź kąty nachylenia, na końcu ustawiając jego pozycję. Dzięki temu unikniemy konieczności przestawiania całej sceny i kamery, aby uzyskać oczekiwany efekt.

Metoda `onKeyDown()` w całości składa się z instrukcji warunkowych. W tym miejscu sprawdzane są wartości klawiszy i przypisywane odpowiednie akcje bądź wartości zmiennym określającym stan używanych klawiszy.

W sytuacji gdy wartość `keyCode` zdarzenia `KeyboardEvent` równa jest `Keyboard.C`, prędkość redukowana jest do wartości początkowej.

```
if (e.keyCode == Keyboard.C)
{
   speed = 5;
   speedMonitor.txt.text = String(speed * 10);
}
```

Z kolei gdy wartość `keyCode` zdarzenia `KeyboardEvent` równa jest `Keyboard.A`, prędkość statku zwiększa się o 0.5.

```
if (e.keyCode == Keyboard.A)
{
   speed += .5;
   speedMonitor.txt.text = String(speed * 10);
}
```

Zmniejszenie prędkości następuje po wciśnięciu klawisza *Z*, którego wartość `keyCode` równa jest `Keyboard.Z`.

```
if (e.keyCode == Keyboard.Z)
{
   speed -= .5;
   speedMonitor.txt.text = String(speed * 10);
}
```

Aby zachować ciągłość akcji sterowania statkiem, nie modyfikujemy w tym miejscu jego pozycji, lecz przypisujemy wartość typu `Boolean` właściwościom, które w metodzie `onEnterFrame()` będą wskazywały na wciskane przez użytkownika klawisze.

Zdarzenie `KeyboardEvent.KEY_DOWN` uruchamiane jest jedynie przy zmianie stanu, w jakim znajduje się klawisz, dlatego zapisanie zmiany pozycji w metodzie `onKeyDown()` spowodowałoby pojedyncze przejście dla każdego wciśniętego klawisza.

```
if (e.keyCode == Keyboard.LEFT)  leftArrowDown = true;
if (e.keyCode == Keyboard.RIGHT) rightArrowDown = true;
if (e.keyCode == Keyboard.UP)    upArrowDown = true;
if (e.keyCode == Keyboard.DOWN)  downArrowDown = true
if (e.keyCode == Keyboard.SPACE) spaceDown = true;
```

W metodzie `onKeyUp()` również zawarliśmy instrukcje warunkowe `if`, ale tutaj sprawdzaliśmy stan klawisza i przypisywaliśmy odpowiedniej zmiennej wartość `false`. Nie mogliśmy wszystkim naraz przypisać wartości `false`, ponieważ mogłoby to przerwać ciągłość innej akcji.

```
if (e.keyCode == Keyboard.LEFT)  leftArrowDown = false;
if (e.keyCode == Keyboard.RIGHT) rightArrowDown = false;
if (e.keyCode == Keyboard.UP)    upArrowDown = false;
if (e.keyCode == Keyboard.DOWN)  downArrowDown = false
if (e.keyCode == Keyboard.SPACE) spaceDown = false;
```

W przypadku wystąpienia warunku, w którym wciśnięty jest klawisz *X*, zapisaliśmy zmianę wartości zmiennej `trackShip` na `false` oraz skierowaliśmy kamerę w stronę środka sceny. Poza tym korzystając z klasy `TweenMax`, stworzyliśmy animację, w której statek kosmiczny powraca do swojej pierwotnej pozycji.

```
if (e.keyCode == Keyboard.X)
{
   trackShip = false;
   view.camera.lookAt(new Vector3D(0, 0, 0));
   TweenMax.to(ship, 1, { x:0, y:0, z:0, rotationY:0, rotationX:0,
rotationZ:0 } );
}
```

W warunku wciśnięcia klawisza *T* zapisaliśmy zmianę wartości `trackShip` na przeciwną do aktualnie ustawionej.

```
if (e.keyCode == Keyboard.T)
{
   trackShip = !trackShip;
}
```

Kup książkę Poleć książkę

W metodzie `onEnterFrame()`, która jest stale wywoływana, użyliśmy wszystkich zmiennych, których wartości zmieniają się w metodach `onKeyDown()` oraz `onKeyUp()`. W zależności od tego, która ze zmiennych równa jest wartości `true`, wykonywana jest akcja obrotu bądź przesunięcia. Dodatkowo jeżeli wartość `trackShip` równa jest `true`, kamera przy każdym odświeżeniu ustawia się w kierunku pozycji statku.

```
if (spaceDown) ship.moveForward(speed);
if (leftArrowDown)  ship.roll(-5);
if (rightArrowDown) ship.roll(5);
if (upArrowDown)    ship.pitch(5);
if (downArrowDown)  ship.pitch(-5);
if (trackShip) view.camera.lookAt(ship.scenePosition);
view.render();
```

Używanie myszy

W poprzednim podrozdziale pokazaliśmy, jak z użyciem klawiatury manipulować trójwymiarowymi obiektami umieszczonymi na scenie. Poparliśmy to przykładem sterowania pojazdem kosmicznym. Jak jednak wiadomo, w aplikacjach Flash częściej stosuje się mysz komputerową niż klawiaturę. Dlatego w tym podrozdziale przyjrzymy się możliwym sposobom zastosowania tego urządzenia wraz z biblioteką Away3D.

MouseEvent

Jak wiemy, w języku ActionScript 3.0 dostępna jest klasa `MouseEvent`, której obiekt obsługuje zdarzenia związane z myszą. W zwykłych aplikacjach dwuwymiarowych zdarzeń myszy można nasłuchiwać na obiektach potomnych względem klasy `InteractiveObject`, czyli prawie na każdym widocznym obiekcie. Niestety, na obiektach trójwymiarowych umieszczonych na scenie Away3D nie ma możliwości bezpośredniego stosowania zdarzeń klasy `MouseEvent`. Mimo to są sytuacje, w których można połączyć stosowanie klasy `MouseEvent` z obiektami trójwymiarowymi. Pierwszy sposób polega na zastosowaniu klasy `MovieClipSprite`, którą poznaliśmy w rozdziale 3. „Obiekty”. Jak pamiętamy, wyświetlana w obiekcie klasy `MovieClipSprite` tekstura ma postać obiektu klasy `DisplayObject`. Skutkiem tego na powierzchni tej tekstury można korzystać ze zdarzeń `MouseEvent`. Nie tylko na obiekcie klasy `MovieClipSprite` można stosować zdarzenia `MouseEvent`. Otóż w rozdziale 4. „Materiały” poznaliśmy klasy `MovieMaterial`, `PhongMovieMaterial` oraz `Dot3MovieMaterial`, dla których źródłem wyświetlanej grafiki są obiekty klas `Sprite` oraz `MovieClip`. Jak wiemy, bezpośrednio na obiekcie zarówno klasy `Sprite`, jak i `MovieClip` można wywołać zdarzenia `MouseEvent`, ale gdy używamy ich wewnątrz

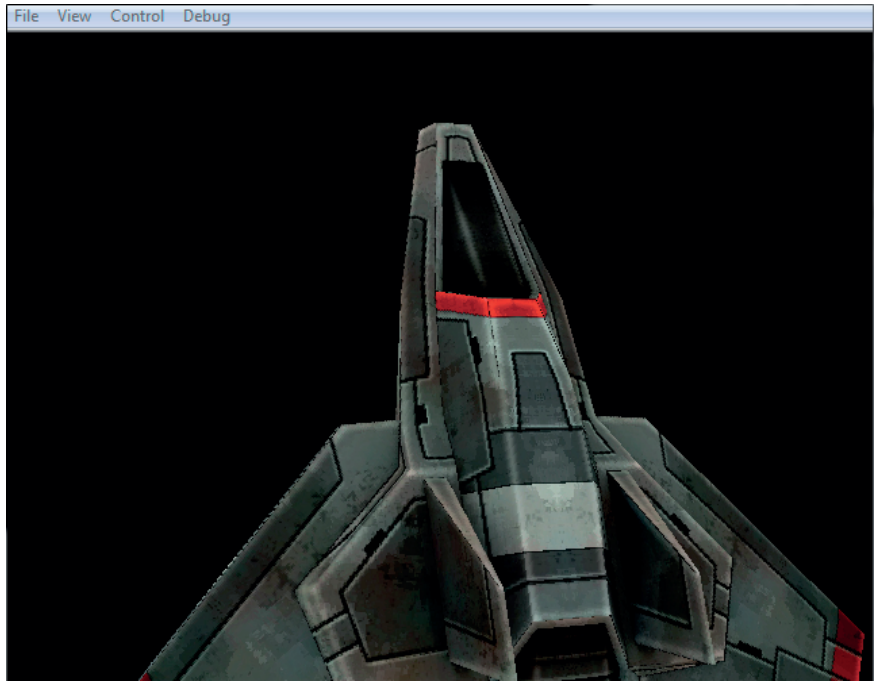
Kup książkę Poleć książkę

któregoś z wyżej wymienionych materiałów, są one niedostępne dla wskaźnika myszy. Jest to normalny stan, który można zmienić, nadając właściwości `interactive` wybranego materiału wartość `true`.

Inny sposób wykorzystania zdarzeń `MouseEvent` do modyfikowania obiektów 3D polega na pobieraniu współrzędnych wskaźnika myszy i przekazywaniu ich do sceny Away3D. Wykrywając zmianę pozycji kursora myszki w oknie aplikacji, można na obiekcie umieszczonym w przestrzeni trójwymiarowej wykonać czynności takie jak skalowanie, przesunięcie czy też obrót. W przykładzie umieszczonym w kolejnym punkcie tego podrozdziału odniesiemy się do tego sposobu interakcji z użyciem myszki komputerowej.

Obracanie i skalowanie statku kosmicznego

Widocznym elementem w tym przykładzie będzie jedynie model statku kosmicznego zastosowany wcześniej w punkcie „Sterowanie statkiem kosmicznym w przestrzeni". Pozycją początkową dla pojazdu będzie środek przestrzeni trójwymiarowej. Sam model zostanie skierowany dziobem wzdłuż osi Z. Poruszając kursorem, będzie można doprowadzić model do różnych pozycji i wymiarów. Przykładowe ustawienia zaprezentowano na rysunku 8.2.



Rysunek 8.2. *Przykładowe ustawienia pozycji i wymiarów modelu*

Aby zmienić pozycję modelu statku kosmicznego, należy wcisnąć lewy przycisk myszy i trzymając go, poruszać kursorem w różne strony okna aplikacji. Wciśnięcie i trzymanie przycisku powoduje zmianę wartości jednej ze zmiennych, która jest warunkiem podejmowania działań obrotu. Bez tego każde poruszenie kursorem, nawet nieplanowe, powodowałoby modyfikacje.

Aby dodatkowo przy poruszaniu kursorem kontrolować wymiary modelu, należy użyć klawisza *Ctrl*. Cały mechanizm wyjaśnimy przy omawianiu kodu źródłowego tego przykładu. Teraz przepisz i skompiluj następujący kod.

```
package
{
   import flash.display.StageAlign;
   import flash.display.StageQuality;
   import flash.display.StageScaleMode;
   import flash.display.Sprite;
   import flash.events.Event;
   import flash.events.MouseEvent;
   import flash.events.KeyboardEvent;
   import flash.ui.Keyboard;
   import flash.geom.Vector3D;
   //
   import away3d.containers.ObjectContainer3D;
   import away3d.containers.View3D;
   import away3d.loaders.Loader3D;
   import away3d.loaders.Max3DS;
   public class MouseEventExample extends Sprite
   {
      private var view:View3D;
      private var shipLoader:Loader3D;
      private var ship:ObjectContainer3D;
      private var mouseDown:Boolean = false;
      private var ctrlDown:Boolean = false;
      private var xPos:Number;
      private var yPos:Number;
      private var ease:Number = .3;

      public function MouseEventExample():void
      {
         if (stage) init();
         else addEventListener(Event.ADDED_TO_STAGE, init);
      }

      private function init(e:Event = null):void
      {
         stage.quality = StageQuality.LOW;
         stage.scaleMode = StageScaleMode.NO_SCALE;
         stage.align = StageAlign.TOP_LEFT;
         removeEventListener(Event.ADDED_TO_STAGE, init);
         addEventListener(Event.ENTER_FRAME, onEnterFrame);
         stage.addEventListener(Event.RESIZE, onResize);
```

Kup książkę Poleć książkę

```
        stage.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
        stage.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
        stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
        stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp);
        //
        view = new View3D();
        addChild(view);
        view.camera.z = -500;
        //
        ship = new ObjectContainer3D();
        shipLoader = Max3DS.load(' ../../resources/models/max3ds/spaceship/
        ↪spaceship.3DS');
        shipLoader.rotationY = -90;
        shipLoader.rotationX = 90;
        shipLoader.position = new Vector3D(0, 0, 0);
        ship.addChild(shipLoader);
        view.scene.addChild(ship);
        //
        onResize();
    }

    private function onMouseDown(e:MouseEvent):void
    {
        mouseDown = true;
    }

    private function onMouseUp(e:MouseEvent):void
    {
        mouseDown = false;
    }

    private function onKeyDown(e:KeyboardEvent):void
    {
        if (e.keyCode == Keyboard.CONTROL) ctrlDown = true;
    }

    private function onKeyUp(e:KeyboardEvent):void
    {
        if (e.keyCode == Keyboard.CONTROL) ctrlDown = false;
    }

    private function onResize(e:Event = null):void
    {
        view.x = stage.stageWidth * .5;
        view.y = stage.stageHeight * .5;
    }

    private function onEnterFrame(e:Event):void
    {
        xPos = stage.mouseX - stage.stageWidth * .5;
        yPos = stage.mouseY - stage.stageHeight * .5;
```

Kup książkę Poleć książkę

```
            if (mouseDown && !ctrlDown)
            {
               ship.rotationY += (xPos * .5 - ship.rotationY) * ease;
               ship.rotationX += (yPos * .5 - ship.rotationX) * ease;
            }
            else if (mouseDown && ctrlDown)
            {
               ship.scale(Math.sqrt(xPos * xPos + yPos * yPos)*.01);
            }

            view.render();
         }
      }
   }
```

W tym przykładzie najważniejszą dla nas klasą z pakietu `flash` jest oczywiście `MouseEvent`. Z Away3D, tak samo jak w poprzednich przykładach tego rozdziału, skorzystaliśmy z modelu 3DS. Dlatego potrzebowaliśmy klas `Loader3D`, `Max3DS` i `ObjectContainer3D` do umieszczenia modelu statku na scenie.

Na początku ciała klasy `MouseEventExample` zdefiniowaliśmy obiekt widoku Away3D, obiekty służące do pobrania i wyświetlenia modelu 3DS oraz kilka zmiennych. Pierwsza w kolejności `mouseDown` przyjmuje wartości typu `Boolean` i określa, czy lewy przycisk myszy jest wciśnięty. Jeżeli jest, to wartość tej właściwości równa się `true`. Kolejną zmienną, którą dodaliśmy, jest `ctrlDown`. Jej wartość również jest typu `Boolean` i określa, czy został wciśnięty klawisz *Ctrl*. Właściwości `xPos` oraz `yPos` to zmienne liczbowe, które określają odległość pozycji kursora od środka okna aplikacji. Ich wartości są modyfikowane przy każdym odświeżeniu stołu montażowego. Ostatnia zmienna `ease` odpowiada za złagodzenie przejścia między pozycjami bądź wymiarami modelu pojazdu. Jej wartość w trakcie uruchamiania oraz używania programu nie zmienia się, dlatego można zdefiniować ją jako wartość stałą.

```
private var view:View3D;
private var shipLoader:Loader3D;
private var ship:ObjectContainer3D;
private var mouseDown:Boolean = false;
private var ctrlDown:Boolean = false;
private var xPos:Number;
private var yPos:Number;
private var ease:Number = .3;
```

Podobnie jak w poprzednich konstruktorach dla klas przykładów, i tutaj, gdy obiekt `stage` jest gotowy, uruchamiamy metodę `init()`.

```
if (stage) init();
else addEventListener(Event.ADDED_TO_STAGE, init);
```

Kup książkę Poleć książkę

Na początku metody `init()` ustawiliśmy jakość generowanego obrazu na poziomie `LOW`. W następnej kolejności wyłączyliśmy skalowanie okna oraz wyrównaliśmy stół montażowy do lewego górnego rogu. Dzięki temu aplikacja będzie działała płynniej i przy powiększeniu okna zwiększymy obszar działania kursora.

```
stage.quality = StageQuality.LOW;
stage.scaleMode = StageScaleMode.NO_SCALE;
stage.align = StageAlign.TOP_LEFT;
```

W kolejnych linijkach kodu metody `init()` usunęliśmy detektor zdarzenia `Event.ADDED_TO_STAGE` i utworzyliśmy szereg nowych. Kolejność, z jaką je dodaliśmy, jest przypadkowa. Pierwszy detektor odpowiada za nasłuchiwanie zdarzenia `Event.ENTER_FRAME` i wywołanie metody `onEnterFrame()`. W drugiej kolejności dodaliśmy detektor dla zmiany rozmiarów okna aplikacji `Event.RESIZE`. Kolejne dwa detektory nasłuchują zdarzeń `MouseEvent.MOUSE_DOWN` oraz `MouseEvent.MOUSE_UP`, które odpowiadają za kliknięcie i puszczenie przycisku myszki. Podobne zadanie przydzieliliśmy detektorom zdarzeń `KeyboardEvent.KEY_DOWN` i `KeyboardEvent.KEY_UP`, z tą różnicą, że one wskazują na kliknięcie bądź zwolnienie klawisza na klawiaturze.

```
removeEventListener(Event.ADDED_TO_STAGE, init);
addEventListener(Event.ENTER_FRAME, onEnterFrame);
stage.addEventListener(Event.RESIZE, onResize);
stage.addEventListener(MouseEvent.MOUSE_DOWN, onMouseDown);
stage.addEventListener(MouseEvent.MOUSE_UP, onMouseUp);
stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp);
```

Po ustaleniu detektorów zdarzeń stworzyliśmy i dodaliśmy obiekt widoku biblioteki Away3D. Dodatkowo przybliżyliśmy kamerę do pozycji -50 na osi Z, tak aby statek w swojej początkowej pozycji był dobrze widoczny.

```
view = new View3D();
addChild(view);
view.camera.z = -50;
```

Gdy widok został dodany, stworzyliśmy obiekt statku kosmicznego, korzystając z ustawień z poprzednich przykładów tego rozdziału. Po pobraniu modelu za pomocą klasy `Max3DS` obróciliśmy go tak, aby dziobem, zwrócony był w stronę osi Z. Następnie umieściliśmy go w kontenerze o nazwie `ship` i dodaliśmy go do sceny.

```
ship = new ObjectContainer3D();
shipLoader = Max3DS.load('models/spaceship/spaceship.3ds');
shipLoader.rotationY = -90;
shipLoader.rotationX = 90;
shipLoader.position = new Vector3D(0, 0, 0);
ship.addChild(shipLoader);
view.scene.addChild(ship);
```

Na końcu bloku kodu metody `init()` odwołaliśmy się do metody `onResize()`.

Kup książkę Poleć książkę

Kolejnymi metodami, które zapisaliśmy w klasie `MouseEventExample`, są `onMouseDown()` oraz `onMouseUp()`. Gdy wciśnięty jest lewy przycisk myszki, wywoływana jest metoda `onMouseDown()`, w której właściwości `mouseDown` przypisywana jest wartość `true`.

Z kolei po puszczeniu przycisku myszy wartość `mouseDown()` zostaje zmieniona na `false` w metodzie `onMouseUp()`.

Idąc tym torem, zapisaliśmy metody `onKeyDown()` oraz `onKeyUp()`, które uruchamiane są z każdym wciśnięciem i zwolnieniem klawisza na klawiaturze. Jednak nie każde wywołanie tych metod powoduje zmiany. Wewnątrz obu metod zastosowaliśmy instrukcję warunkową `if`. Obie filtrują kody wciskanych i zwalnianych klawiszy w celu wychwycenia wartości numerycznej odpowiadającej numerowi klawisza *Ctrl*. Jeżeli klawisz ten zostanie wciśnięty, to w metodzie `onKeyDown()` zmienna `ctrlDown` zmieni swoją wartość na `true`. Z kolei po puszczeniu przycisku *Ctrl* w metodzie `onKeyUp()` zmienna `ctrlDown` zmieni swoją wartość na `false`.

Przy każdej zmianie rozmiaru okna uruchamiana jest metoda `onResize()`. W jej ciele zapisaliśmy wyśrodkowanie widoku `view` w oknie programu.

```
view.x = stage.stageWidth * .5;
view.y = stage.stageHeight * .5;
```

Na końcu klasy `MouseEventExample` zapisaliśmy metodę `onEnterFrame()`, wywoływaną przy każdym wystąpieniu zdarzenia `Event.ENTER_FRAME`. W pierwszych dwóch linijkach wpisaliśmy formuły wyliczające różnicę między współrzędnymi kursora i współrzędnymi środka okna aplikacji.

```
xPos = stage.mouseX - stage.stageWidth * .5;
yPos = stage.mouseY - stage.stageHeight * .5;
```

Następnie zapisaliśmy instrukcje `if`, które sprawdzają wartości zmiennych `mouseDown` oraz `ctrlDown`. W pierwszym przypadku sprawdzamy, czy wartość `mouseDown` jest równa `true`, a wartość `ctrlDown` równa `false`. Jeżeli warunek zachodzi, to wcześniej zdefiniowane odległości `xPos` i `yPos` zostaną użyte do zmiany kątów nachylenia modelu na osiach X i Y. Parametr `ease` służy jako mnożnik łagodzący przejście między pozycjami.

```
if (mouseDown && !ctrlDown)
{
   ship.rotationY += (xPos * .5 - ship.rotationY) * ease;
   ship.rotationX += (yPos * .5 - ship.rotationX) * ease;
}
```

Jeżeli wartości właściwości `mouseDown` i `ctrlDown` są równe `true`, to każda zmiana współrzędnych kursora spowoduje zmianę wymiarów obiektu. Im dalej od środka będzie skierowany kursor, tym większą skalę będzie miał model statku kosmicznego. Aby obliczyć tę odległość, musieliśmy użyć właściwości `xPos` i `yPos`. Bezpośrednio

Kup książkę Poleć książkę

w metodzie `scale()` obiektu `ship` obydwie te wartości podnieśliśmy do kwadratu i dodaliśmy. Następnie wyliczyliśmy pierwiastek z tej sumy i pomnożyliśmy go przez 0.01. W ten sposób, bazując na współrzędnych pozycji kursora, uzyskamy długość wektora, którą będziemy skalowali obiekt `ship`.

```
else if (mouseDown && ctrlDown)
{
   ship.scale(Math.sqrt(xPos * xPos + yPos * yPos)*.01);
}
```

Na końcu metody `onEnterFrame()` odświeżamy widok metodą `render()`.

MouseEvent3D

We wcześniejszych przykładach tego podrozdziału pokazaliśmy, że można za pomocą `MouseEvent` stworzyć interakcje użytkownika ze środowiskiem Away3D. Problem polega na tym, że przedstawione sposoby nie odnosiły się bezpośrednio do obiektów trójwymiarowych, tylko do obiektów klas pochodnych względem `InteractiveObject`. Na szczęście biblioteka Away3D ma swoją implementację pełnej obsługi zdarzeń myszki w postaci klasy `MouseEvent3D`.

Zlokalizowana w pakiecie `away3d.events` klasa `MouseEvent3D` jest odpowiednikiem standardowego `MouseEvent`, tylko że w tym przypadku ma ona zastosowanie bezpośrednio na obiektach umieszczonych w przestrzeni trójwymiarowej, niezależnie od ich rodzaju i materiału, którym są pokryte. W swoich zasobach klasa `MouseEvent3D` ma stałe, które definiują większość zdarzeń znanych z `MouseEvent`, między innymi `MouseEvent3D.MOUSE_DOWN`, `MouseEvent3D.MOUSE_UP` czy też `MouseEvent3D.MOUSE_MOVE`. Każde ze zdarzeń `MouseEvent3D` wywołuje się bezpośrednio na obiekcie, na którym ma ono być rejestrowane. Działa to na tych samych zasadach jak w przypadku zwykłych obiektów `DisplayObject`.

```
Obj3d.addEventListener(MouseEvent3D.MOUSE_DOWN, onMouseDown);
```

Oczywiście zamiast odwoływać się do obiektu zdarzenia, można wpisać samą postać `String`, którą reprezentuje stała obiektu `MouseEvent3D`. Powyższy przykład dodania detektora w tym przypadku wyglądałby następująco:

```
Obj3d.addEventListener("mouseDown3d", onMuseDown);
```

Tabele 8.8 i 8.9 zawierają metody i właściwości klasy `MouseEvent3D`. W tabeli 8.10 wypisane zostały stałe odpowiadające konkretnym zdarzeniom.

Kup książkę Poleć książkę

Tabela 8.8. *Właściwości klasy MouseEvent3D*

Nazwa	Rodzaj	Wartość domyślna	Opis
`ctrlKey`	`Boolean`	`false`	Określa, czy klawisz *Ctrl* jest wciśnięty
`elementVO`	`ElementVO`		Obiekt typu `ElementVO`, na którym wywołano zdarzenie
`material`	`Material`		Materiał obiektu 3D, na którym wywołano zdarzenie
`Object`	`Object3D`		Obiekt 3D, na którym wywołano zdarzenie
`sceneX`	`Number`		Współrzędna *x* na scenie
`sceneY`	`Number`		Współrzędna *y* na scenie
`sceneZ`	`Number`		Współrzędna *z* na scenie
`screenX`	`Number`		Współrzędna *x* w widoku
`screenY`	`Number`		Współrzędna y w widoku
`screenZ`	`Number`		Współrzędna *z* w widoku
`shiftKey`	`Number`	`false`	Określa, czy klawisz *Shift* jest wciśnięty
`uv`	`UV`		Współrzędne UV wewnątrz obiektu, na którym wywołano zdarzenie
`view`	`View3D`		Obiekt widoku, w którym wywołano zdarzenie

Tabela 8.9. *Metody klasy MouseEvent3D*

Nazwa	Opis
`MouseEvent3D(type:String)`	Konstruktor
`clone():Event`	Tworzy kopię obiektu `MouseEvent3D`

Tabela 8.10. *Stałe klasy MouseEvent3D*

Nazwa	Wartość domyślna	Opis
`MOUSE_DOWN`	`'mouseDown3d'`	Zdarzenie wciśnięcia przycisku myszy na trójwymiarowym obiekcie
`MOUSE_MOVE`	`'mouseMove3d'`	Zdarzenie poruszania kursorem na trójwymiarowym obiekcie
`MOUSE_OUT`	`'mouseOut3d'`	Zdarzenie opuszczenia kursora z trójwymiarowego obiektu lub któregokolwiek z obiektów w nim umieszczonych
`MOUSE_OVER`	`'mouseOver3d'`	Zdarzenie najechania kursorem na trójwymiarowy obiekt lub którykolwiek z obiektów w nim umieszczonych
`MOUSE_UP`	`'mouseUp3d'`	Zdarzenie puszczenia przycisku myszy na trójwymiarowym obiekcie

Kup książkę Poleć książkę

Tabela 8.10. *Stałe klasy MouseEvent3D (ciąg dalszy)*

Nazwa	Wartość domyślna	Opis
ROLL_OVER	'rollOut3d'	Zdarzenie najechania kursorem na trójwymiarowy obiekt
ROLL_OUT	'rollOver3d'	Zdarzenie opuszczenia kursora z trójwymiarowego obiektu

Metody dla zdarzeń MouseEvent3D

Obiekty klasy `Object3D` mają własne metody inicjujące detektory zdarzeń dla `MouseEvent3D`. Każda z tych metod ma inną nazwę, która wskazuje na rodzaj zdarzenia. Jako argument dla takiej metody należy podać jedynie nazwę metody, która ma być wywołana po zajściu zdarzenia.

Tabela 8.11 zawiera spis nazw metod oraz zdarzeń, którym one odpowiadają.

Tabela 8.11. *Metody obsługi zdarzeń MouseEvent3D*

Nazwa metody	Zdarzenie
addOnMouseDown	MouseEvent3D.MOUSE_DOWN
addOnMouseMove	MouseEvent3D.MOUSE_MOVE
addOnMouseOut	MouseEvent3D.MOUSE_OUT
addOnMouseOver	MouseEvent3D.MOUSE_OVER
addOnMouseUp	MouseEvent3D.MOUSE_UP
addOnRollOut	MouseEvent3D.ROLL_OUT
addOnRollOver	MouseEvent3D.ROLL_OVER

Malowanie na trójwymiarowych obiektach

W tym punkcie wyjaśnimy, jak napisać aplikację umożliwiającą malowanie po powierzchni obiektów 3D.

Aby wykonać to zadanie, w pierwszej kolejności stworzymy obiekt ściany i wypełnimy go teksturą z ceglaną powierzchnią. Następnie do tego obiektu dodamy detektory zdarzeń poruszania kursorem, wciśnięcia i puszczenia przycisku myszy. Dodatkową zmienną typu `Boolean` sprawimy, że malowanie będzie trwało od momentu wciśnięcia do zwolnienia przycisku myszy.

Sam proces rysowania umieścimy w metodzie obsługi zdarzenia `Event.ENTER_FRAME`.

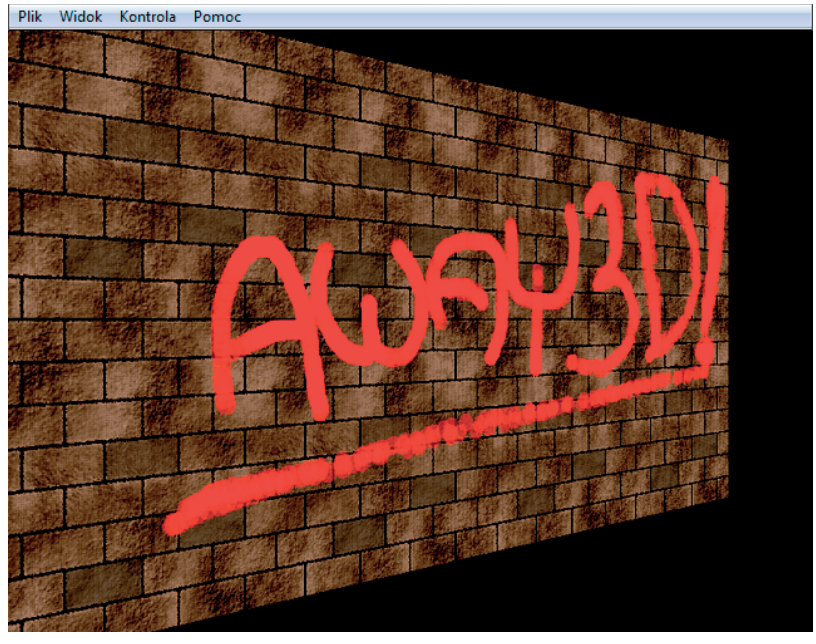
Aby na powierzchni ściany były widoczne zmiany, skorzystamy z materiału typu `MovieMaterial`. Jego pierwszą warstwę wypełnimy wcześniej wspomnianą ceglaną

Kup książkę Poleć książkę

teksturą. Kolejne rysunki będą skutkiem tworzenia nowych obiektów `Sprite`, w których metodą `drawCircle()` będziemy rysowali pojedyncze, półprzezroczyste czerwone koła.

Aby ustalić współrzędne kursora na obiekcie, skorzystamy ze wspomnianego w tabeli 8.8 obiektu `UV`.

Efekt, który uzyskamy po napisaniu i skompilowaniu przykładu, ilustruje rysunek 8.3.



Rysunek 8.3. *Malowanie graffiti na trójwymiarowej ścianie*

Teraz przepisz i skompiluj następujący kod. Następnie zajmiemy się omawianiem jego poszczególnych fragmentów.

```
package
{
   import flash.geom.Vector3D;
   import flash.events.Event;
   import flash.display.StageQuality;
   import flash.display.StageAlign;
   import flash.display.StageScaleMode;
   import flash.display.Sprite;
   //
   import away3d.containers.View3D;
   import away3d.primitives.Cube;
   import away3d.core.utils.Cast;
   import away3d.materials.MovieMaterial;
   import away3d.primitives.data.CubeMaterialsData;
   import away3d.events.MouseEvent3D;
```

```
public class GraffitiExample extends Sprite
{
   private var view:View3D;
   private var wall:Cube;
   private var cubeMaterials:CubeMaterialsData;
   private var drawingEnabled:Boolean = false;

   public function GraffitiExample()
   {
      if (stage) init();
      else addEventListener(Event.ADDED_TO_STAGE, init);
   }

   private function init(e:Event = null):void
   {
      stage.quality = StageQuality.LOW;
      stage.scaleMode = StageScaleMode.NO_SCALE;
      stage.align = StageAlign.TOP_LEFT;
      removeEventListener(Event.ADDED_TO_STAGE, init);
      stage.addEventListener(Event.RESIZE, onResize);
      stage.addEventListener(Event.ENTER_FRAME, onEnterFrame);
      //
      view = new View3D();
      addChild(view);
      //
      cubeMaterials = new CubeMaterialsData();
      cubeMaterials.front = new MovieMaterial(new brickTexture(),
      ↪{ interactive:true } );
      cubeMaterials.back = new MovieMaterial(new brickTexture(),
      ↪{ interactive:true } );
      cubeMaterials.top = new MovieMaterial(new brickTexture(),
      ↪{ interactive:true } );
      cubeMaterials.bottom = new MovieMaterial(new brickTexture(),
      ↪{ interactive:true } );
      cubeMaterials.left = new MovieMaterial(new brickTexture(),
      ↪{ interactive:true } );
      cubeMaterials.right = new MovieMaterial(new brickTexture(),
      ↪{ interactive:true } );
      wall = new Cube( { width:1024, height:512, depth:50,
      ↪cubeMaterials:cubeMaterials } );
      wall.addOnMouseDown(onMouseDown);
      wall.addOnMouseUp(onMouseUp);
      wall.addEventListener(MouseEvent3D.MOUSE_MOVE, onMouseMove);
      view.scene.addChild(wall);
      //
      onResize();
   }

   private function onMouseDown(e:MouseEvent3D):void
   {
      drawingEnabled = true;
   }
```

```
        private function onMouseUp(e:MouseEvent3D):void
        {
           drawingEnabled = false;
        }

        private function onMouseMove(e:MouseEvent3D):void
        {
           if (drawingEnabled)
           {
              var graff:Sprite = new Sprite();
              graff.graphics.beginFill(0xFF0000, .5);
              graff.graphics.drawCircle(e.uv.u * (e.material as MovieMaterial).
              ↪movie.width, (e.material as MovieMaterial).movie.height -
              ↪e.uv.v * (e.material as MovieMaterial).movie.height, 10);
              graff.graphics.endFill();
              (e.material as MovieMaterial).movie.addChild(graff);
           }
        }

        private function onResize(e:Event = null):void
        {
           view.x = stage.stageWidth * .5;
           view.y = stage.stageHeight * .5;
        }

        private function onEnterFrame(e:Event):void
        {
           wall.rotationY -= .1;
           view.render();
        }
     }

  }
```

Na początku bloku klasy `GraffitiExample` zdefiniowaliśmy obiekt widoku `View3D`, a następnie obiekt klasy `Cube` o nazwie `wall`. Jak wspomnieliśmy na początku tego punktu, materiałem pokrywającym ścianę jest obiekt klasy `MovieMaterial`. W naszym przykładzie nazwaliśmy go `wallTexture`. Jako źródło dla tego materiału posłużył obiekt klasy `brickTexture`. Jest to element `MovieClip` dodany do biblioteki pliku *fla*. Po zdefiniowaniu wszystkich potrzebnych obiektów utworzyliśmy jeszcze jedną zmienną o nazwie `drawingEnabled`, której wartość określa, czy można rysować po powierzchni ściany.

```
private var view:View3D;
private var wall:Cube;
private var wallTexture:MovieMaterial;
private var textureMC:brickTexture;
private var isDrawing:Boolean = false;
```

W konstruktorze uruchamiamy metodę `init()`, z chwilą gdy obiekt `Stage` zostanie zainicjowany.

Kup książkę Poleć książkę

```
if (stage) init();
else addEventListener(Event.ADDED_TO_STAGE, init);
```

Gdy metoda `init()` zostanie wywołana, obiekt nasłuchujący zdarzenia `Event.ADDED_TO_STAGE` będzie zbędny, dlatego można go usunąć, stosując metodę `removeEventListener()`.

Dla szybszego działania aplikacji ustawiliśmy niską jakość wyświetlanego obrazu, stosując w tym celu wartość `StageQuality.LOW`. W kolejnych linijkach usunęliśmy skalowanie obiektów i wyrównaliśmy stół montażowy do lewego górnego rogu. Ostatnimi operacjami, jakie wykonaliśmy na obiekcie `stage`, było przypisanie im detektorów zdarzeń reagujących na zmianę rozmiarów okna i odświeżanie wyświetlanego obrazu.

```
removeEventListener(Event.ADDED_TO_STAGE, init);
stage.quality = StageQuality.LOW;
stage.scaleMode = StageScaleMode.NO_SCALE;
stage.align = StageAlign.TOP_LEFT;
stage.addEventListener(Event.RESIZE, onResize);
stage.addEventListener(Event.ENTER_FRAME, onEnterFrame);
```

Po ustaleniu wszystkich opcji i dodaniu obiektów nasłuchujących zdarzeń `Event.RESIZE` i `Event.ENTER_FRAME` stworzyliśmy i dodaliśmy widok bez ustalania jakichkolwiek dodatkowych zmian wartości jego właściwości. Położenie zmieniamy w metodzie `onResize()`.

```
view = new View3D();
addChild(view);
```

W kolejnych linijkach metody `init()` stworzyliśmy i uzupełniliśmy zawartość obiektu klasy `CubeMaterialsData` o nazwie `cubeMaterials`. Każdemu z boków sześcianu przypisaliśmy nowy materiał typu `MovieMaterial`. W każdym z tych materiałów źródłem wyświetlanego obrazu jest obiekt klasy `brickTexture`. Dodatkowo aby umożliwić malowanie na powierzchni tego materiału, ustawiliśmy wartość właściwości `interactive` jako `true`.

```
cubeMaterials = new CubeMaterialsData();
cubeMaterials.front = new MovieMaterial(new brickTexture(),
↪{ interactive:true } );
cubeMaterials.back = new MovieMaterial(new brickTexture(),
↪{ interactive:true } );
cubeMaterials.top = new MovieMaterial(new brickTexture(),
↪{ interactive:true } );
cubeMaterials.bottom = new MovieMaterial(new brickTexture(),
↪{ interactive:true } );
cubeMaterials.left = new MovieMaterial(new brickTexture(),
↪{ interactive:true } );
cubeMaterials.right = new MovieMaterial(new brickTexture(),
↪{ interactive:true } );
```

Kup książkę Poleć książkę

Po wykonaniu tych czynności stworzyliśmy obiekt klasy `Cube` o nazwie `wall`, w którym właściwości `cubeMaterials` przypisaliśmy nasz obiekt `cubeMaterials`. Następnie korzystając z wcześniej poznanych metod `addOnMouseDown()`, `addOnMouseUp()` oraz zwykłego `addEventListener()` dodaliśmy obiekt nasłuchujący zdarzeń `MouseEvent3D`.

```
wall = new Cube( { width:1024, height:512, depth:50,
↪cubeMaterials:cubeMaterials } );
wall.addOnMouseDown(onMouseDown);
wall.addOnMouseUp(onMouseUp);
wall.addEventListener(MouseEvent3D.MOUSE_MOVE, onMouseMove);
```

Na końcu metody `init()` dodaliśmy obiekt ściany do sceny Away3D i wywołaliśmy metodę `onResize()` w celu ustawienia widoku na środku okna aplikacji.

```
view.scene.addChild(wall);
onResize();
```

Metody `onMouseDown()` oraz `onMouseUp()` wywoływane są w sytuacji zajścia zdarzenia `MouseEvent3D`. Po wciśnięciu przycisku myszki uruchamia się metoda `onMouseDown()`, w której właściwości `mouseDown` przypisywana jest wartość `true`. Z kolei przy puszczeniu przycisku myszy wywoływana jest metoda `onMouseUp()`, w której `mouseDown` zmienia swoją wartość na `false`.

Najważniejszą metodą w klasie `GraffitiExample` jest `onMouseMove()`. Kod zapisany w jej ciele odpowiada za rysowanie wzorów na powierzchni tekstury obiektu `wall`. Aby malowanie było możliwe jedynie przy wciśniętym przycisku myszy, musieliśmy w pierwszej kolejności zastosować instrukcję warunkową `if`, w której sprawdzana jest wartość właściwości `drawingEnabled`.

Jeżeli wartość tej właściwości równa jest `true`, to wewnątrz instrukcji tworzony jest nowy obiekt klasy `Sprite`.

```
var graff:Sprite = new Sprite();
```

Stosując odwołanie do obiektu `graphics` wewnątrz obiektu `graff`, można korzystać z jego metod rysowania.

Do narysowania półprzezroczystego czerwonego koła użyliśmy metod `beginFill()` oraz `drawCircle()`. W metodzie `beginFill()` jako argument podaliśmy szesnastkowy kod koloru czerwonego.

```
graff.graphics.beginFill(0xFF0000, .5);
```

Metoda `drawCircle()` przyjmuje trzy właściwości: `x`, `y`, oraz `radius`. Jako `x` podaliśmy wartość właściwości `u` z obiektu `UV` i pomnożyliśmy ją przez szerokość tekstury.

```
e.uv.u * (e.material as MovieMaterial).movie.width;
```

Kup książkę Poleć książkę

Z kolei aby określić wartość argumentu `y`, musieliśmy od wysokości tekstury odjąć iloczyn tej wysokości i wartości właściwości `v` z obiektu `UV`. Dzięki temu współrzędne tekstury będą zgodne z ruchem kursora.

```
(e.material as MovieMaterial).movie.height - e.uv.v * (e.material as
MovieMaterial).movie.height;
```

Etap rysowania zakończyliśmy metodą `endFill()`, po czym umieściliśmy nowo utworzony obiekt `graff` w wyświetlanym obiekcie klasy `brickTexture`.

```
graff.graphics.endFill();
(e.material as MovieMaterial).movie.addChild(graff);
```

Warto wspomnieć o tym, że zapis `(e.material as MovieMaterial).movie` odwołuje się do źródła materiału, na którym aktualnie znajduje się kursor.

Rozmieszczanie obiektów na planszy

Przyjmijmy, że Twoim zadaniem jest napisanie strategicznej gry czasu rzeczywistego. Jednymi z kluczowych elementów takiej gry są:

- możliwość zmiany pozycji obiektów strategicznych,
- tworzenie oraz przemieszczanie jednostek bojowych.

Tymi konkretnymi zagadnieniami zajmiemy się w tym punkcie, a efekt, który uzyskamy, kompilując kod źródłowy przykładu, przedstawia rysunek 8.4.

Rysunek 8.4. *Rozmieszczanie obiektów na planszy*

Na rysunku 8.4 widzimy kawałek planszy pokryty teksturą ze sztucznym ukształtowaniem terenu tworzącym pole podzielone wyschniętym korytem rzeki. Na tej planszy umieszczony jest jeden budynek, który w naszym scenariuszu pełni funkcję fabryki pojazdów latających. Poza nim w powietrzu wiszą cztery wyprodukowane statki kosmiczne.

Tworzenie tych latających obiektów następuje po kliknięciu na obiekt fabryki — pojazd pojawia się na jej platformie, po czym przemieszcza na losowo wybraną pozycję.

Najechanie kursorem na którykolwiek z modeli powoduje pojawienie się czerwonej poświaty. Oznacza to, że na wybranym elemencie można wykonać akcję zmiany pozycji.

Żeby zmienić pozycję wybranego obiektu, należy kliknąć na niego, trzymając wciśnięty klawisz *Ctrl*.

Na podłożu planszy pojawi się niebieski celownik, tak jak to pokazano na rysunku 8.4. Celownik ten, podobnie jak w grach tego typu, wyznacza nowe miejsce docelowe.

Aby wybrany obiekt zmienił swoją pozycję na nową, należy kliknąć lewy przycisk myszy, trzymając wciśnięty klawisz *Ctrl*. W przypadku obiektu fabryki pozycja zmieniona zostanie natychmiastowo, ponieważ — w przeciwieństwie do obiektów statków — nie stworzyliśmy animacji płynnego przejścia wygenerowanej za pomocą bibliotek `TweenMax` i `TimelineMax`. Wszystkie procesy, które zachodzą w kodzie źródłowym tego przykładu, omówimy za chwilę. Teraz przepisz ten kod i załącz go jako główną klasę programu. Do poprawnego działania aplikacji niezbędne są zasoby umieszczone w pliku *fla*.

```
package
{
    import flash.display.StageAlign;
    import flash.display.StageQuality;
    import flash.display.StageScaleMode;
    import flash.display.Sprite;
    import flash.events.Event;
    import flash.events.KeyboardEvent;
    import flash.filters.GlowFilter;
    import flash.geom.Vector3D;
    import flash.ui.Keyboard;
    //
    import away3d.core.base.Object3D;
    import away3d.core.utils.Cast;
    import away3d.core.render.Renderer;
    import away3d.primitives.Plane;
    import away3d.materials.MovieMaterial;
    import away3d.materials.BitmapMaterial;
```

Kup książkę Poleć książkę

```
import away3d.events.MouseEvent3D;
import away3d.events.Loader3DEvent;
import away3d.containers.ObjectContainer3D;
import away3d.containers.View3D;
import away3d.loaders.Loader3D;
import away3d.loaders.Max3DS;
//
import com.greensock.TweenMax;
import com.greensock.TimelineMax;

public class MouseEvent3DExample extends Sprite
{
  private var view:View3D;
        private var pointer:Plane;
  private var pointerMC:Crosshair = new Crosshair();
  private var board:ObjectContainer3D;
  private var ground:Plane;
  private var texture:BitmapMaterial;
  private var modelLoader:Loader3D;
  private var landingDock:ObjectContainer3D;
  private var fighter:ObjectContainer3D;
  private var animations:TimelineMax;
  private var leftArrowDown:Boolean = false;
  private var rightArrowDown:Boolean = false;
  private var upArrowDown:Boolean = false;
  private var downArrowDown:Boolean = false;
  private var selectedBuilding:Object3D;
  private var selectedVehicle:Object3D;
  private var ctrlDown:Boolean = false;
  private var zoomInDown:Boolean = false;
  private var zoomOutDown:Boolean = false;
  private var fighters:Vector.<Object3D> = new Vector.<Object3D>;

  public function MouseEvent3DExample():void
  {
        if (stage) init();
        else addEventListener(Event.ADDED_TO_STAGE, init);
  }

  private function init(e:Event = null):void
  {
        stage.quality = StageQuality.LOW;
        stage.scaleMode = StageScaleMode.NO_SCALE;
        stage.align = StageAlign.TOP_LEFT;
        removeEventListener(Event.ADDED_TO_STAGE, init);
        addEventListener(Event.ENTER_FRAME, onEnterFrame);
        stage.addEventListener(Event.RESIZE, onResize);
        stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
        stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp);
        view = new View3D();
        view.renderer = Renderer.BASIC;
        view.x = stage.stageWidth * .5;
```

```
            view.y = stage.stageHeight * .5;
            addChild(view);
            view.camera.zoom = 15;
            view.camera.y = 800;
            view.camera.lookAt(new Vector3D(0, 0, 0));
            view.scene.addOnMouseMove(overScene);

            pointer = new Plane( { width:34, height:34,
            ↪material:new MovieMaterial(pointerMC),pushfront:true } );
            initBoard();
            initBuildings();
            onResize();
      }

      private function initBoard():void
      {
            board = new ObjectContainer3D();
            texture = new BitmapMaterial(Cast.bitmap('terrain'));
            ground = new Plane();
            ground.width = 512;
            ground.height = 512;
            ground.pushback = true;
            ground.material = texture;
            ground.addOnMouseDown(onGroundClick);
            board.addChild(ground);
            view.scene.addChild(board);
      }

      private function overScene(e:MouseEvent3D):void
      {
            pointer.x = e.sceneX;
            pointer.z = e.sceneZ;
      }

      private function onGroundClick(e:MouseEvent3D):void
      {
            trace(e.sceneX, e.sceneZ);
            if (ctrlDown && selectedBuilding)
            {
                  selectedBuilding.alpha = 1;
                  selectedBuilding.x = pointer.x;
                  selectedBuilding.z = pointer.z;
                  selectedBuilding = null;
                  view.scene.removeChild(pointer);
            }
            else if (ctrlDown && selectedVehicle)
            {
                  view.scene.removeChild(pointer);
                  TweenMax.to(selectedVehicle, 1, { x:pointer.x, z:pointer.z,
                  ↪onComplete:function() { selectedBuilding = null; } } );
            }
            else{}
```

```
}
private function initBuildings():void
{
      modelLoader = Max3DS.load('../../resources/models/max3ds/
      ↪landingDock/LandingDock.3DS');
      modelLoader.addOnSuccess(onLandingDockSuccess);
}

private function onLandingDockSuccess(e:Loader3DEvent):void
{
      landingDock = new ObjectContainer3D();
      landingDock.ownCanvas = true;
      landingDock.name = 'RED_landingDock';
      landingDock.addChild(modelLoader.handle);
      landingDock.scale(.2);
      landingDock.x = 71;
      landingDock.z = 196;
      landingDock.y = 15;
      landingDock.rotationX = 90;
      landingDock.filters = [ new GlowFilter(0xFF0000, 0, 7, 7, 1, 1)];
      view.scene.addChild(landingDock);
      landingDock.addOnMouseDown(landingDockOnMouseDown);
      landingDock.addOnRollOver(landingDockOnMouseRollOver);
      landingDock.addOnRollOut(landingDockOnMouseRollOut);
}

private function landingDockOnMouseDown(e:MouseEvent3D):void
{
      trace('landingDock:CLICK');
      if (ctrlDown)
      {
            selectedBuilding = landingDock;
            selectedBuilding.alpha = .5;
            view.scene.addChild(pointer);
            pointerMC.gotoAndStop('move');
      }else
      {
            modelLoader = Max3DS.load('../../resources/models/max3ds/
            ↪spaceFighter01/spaceFighter01.3ds');
            modelLoader.addOnSuccess(onModelLoaderSuccess);
      }
}

private function landingDockOnMouseRollOver(e:MouseEvent3D):void
{
      trace('landingDock:RollOver');
      TweenMax.to(landingDock.filters[0], .5, { alpha:1 } );
}

private function landingDockOnMouseRollOut(e:MouseEvent3D):void
{
      trace('landingDock:RollOut');
```

```
            TweenMax.to(landingDock.filters[0], .5, { alpha:0 } );
      }

      private function randomPosition(min:Number, max:Number):Number
      {
            return Math.floor(Math.random() * (1 + max - min)) + min;
      }

      private function onModelLoaderSuccess(e:Loader3DEvent):void
      {
            var randomX:Number = landingDock.x - randomPosition(-150, 150);
            var randomZ:Number = landingDock.z - randomPosition(-150, 150);
            fighter = new ObjectContainer3D();
            fighter.ownCanvas = true;
            fighter.addOnMouseDown(fighterOnMouseDown);
            fighter.addOnRollOver(fighterOnMouseRollOver);
            fighter.addOnRollOut(fighterOnMouseRollOut);
            fighter.name = fighter + fighters.length;
            fighter.filters = [new GlowFilter(0xFF0000, 0, 7, 7, 1, 1)];
            fighter.addChild(modelLoader.handle);
            fighter.scale(.001);
            fighter.rotationX = 90;
            fighter.x = landingDock.x;
            fighter.y = 20;
            fighter.z = landingDock.z;
            view.scene.addChild(fighter);
            fighters.push(fighter);
            animations = new TimelineMax( { onComplete:onFighterReady } );
            animations.append(TweenMax.to(fighter, 1, { scaleX:.1,
            ↪scaleY:.1, scaleZ:.1 } ));
            animations.append(TweenMax.to
            ↪(fighter, 2, { x:randomX, y: 60, z:randomZ } ));
            animations.play();
      }

      private function onFighterReady():void
      {
            trace('Fighter ready!');
      }

      private function fighterOnMouseDown(e:MouseEvent3D):void
      {
            trace('Fighter:CLICK');
            if (ctrlDown)
            {
                  selectedVehicle = e.target as ObjectContainer3D;
                  view.scene.addChild(pointer);
                  pointerMC.gotoAndStop('move');
            }
      }

      private function fighterOnMouseRollOver(e:MouseEvent3D):void
```

```
{
      trace('Fighter:RollOver');
      TweenMax.to(e.currentTarget.filters[0], .5, { alpha:1 } );
}

private function fighterOnMouseRollOut(e:MouseEvent3D):void
{
      trace('Fighter:RollOut');
      TweenMax.to(e.currentTarget.filters[0], .5, { alpha:0 } );
}

private function onResize(e:Event = null):void
{
      view.x = stage.stageWidth * .5;
      view.y = stage.stageHeight * .5;
}

private function onKeyDown(e:KeyboardEvent):void
{
      if (e.keyCode == Keyboard.LEFT)     leftArrowDown = true;
      if (e.keyCode == Keyboard.RIGHT)    rightArrowDown = true;
      if (e.keyCode == Keyboard.UP)       upArrowDown = true;
      if (e.keyCode == Keyboard.DOWN)     downArrowDown = true;
      if (e.keyCode == Keyboard.CONTROL)  ctrlDown = true;
      if (e.keyCode == 187)  zoomInDown = true;
      if (e.keyCode == 189)  zoomOutDown = true;
}
private function onKeyUp(e:KeyboardEvent):void
{
      if (e.keyCode == 187)  zoomInDown = false;
      if (e.keyCode == 189)  zoomOutDown = false;
      if (e.keyCode == Keyboard.LEFT)  leftArrowDown = false;
      if (e.keyCode == Keyboard.RIGHT) rightArrowDown = false;
      if (e.keyCode == Keyboard.UP)    upArrowDown = false;
      if (e.keyCode == Keyboard.DOWN)  downArrowDown = false;
      if (e.keyCode == Keyboard.CONTROL)
      {
            ctrlDown = false;
            if (selectedBuilding)
            {
                  selectedBuilding.alpha = 1;
                  selectedBuilding = null;
                  view.scene.removeChild(pointer);
            }
      }
}

private function onEnterFrame(e:Event):void
{
      if (zoomInDown)
      {
            if (ctrlDown) view.camera.y += 5;
```

```
                    else view.camera.zoom++;
                }
                if (zoomOutDown)
                {
                    if (ctrlDown) view.camera.y -= 5;
                    else view.camera.zoom--;
                }
                if (leftArrowDown)
                {
                    if (ctrlDown) view.camera.rotationY++;
                    else view.camera.x -= 10;
                }
                if (rightArrowDown)
                {
                    if (ctrlDown) view.camera.rotationY--;
                    else view.camera.x += 10;
                }
                if (upArrowDown)
                {
                    if (ctrlDown) view.camera.rotationX++;
                    else view.camera.z += 10;
                }
                if (downArrowDown)
                {
                    if (ctrlDown) view.camera.rotationX--;
                    else view.camera.z -= 10;
                }
                view.render();
            }
        }
    }
```

Jak w większości kodów źródłowych zawartych w tej książce, ustawiliśmy poziom jakości wyświetlania elementów, brak skalowania oraz wyrównanie stołu montażowego. Użyliśmy do tego celu klas `StageAlign`, `StageQuality` i `StageScaleMode`.

Jednym z istotnych elementów tego przykładu jest możliwość zmiany położenia obiektów znajdujących się na planszy. Aby móc wyznaczyć nową pozycję na trzech osiach równocześnie, musieliśmy skorzystać z klasy `Vector3D`. Do samego przemieszczenia użyliśmy zdarzeń `MouseEvent3D` i animacji generowanych klasami `TweenMax` i `TimelineMax`.

Ponieważ do zrealizowania tego przykładu skorzystaliśmy z modeli zapisanych w formacie 3DS, musieliśmy użyć kilku klas, aby wyświetlić je na scenie. W pierwszej kolejności użyliśmy ogólnej klasy `Loader3D`, przechowującej model. Aby pobrać obiekt w formacie 3DS, skorzystaliśmy z klasy `Max3DS`. Skala oraz położenie zawartości modelu uzależnione są od zastosowanych podczas eksportu ustawień. Pobrany z określonej lokalizacji obiekt modelu umieściliśmy w kontenerze `Object`
`↪Container3D`, aby w jego wnętrzu dostosować obiekt modelu do ustawień sceny.

Wspomniane otoczenie stworzyliśmy za pomocą klasy `Plane` i nadaliśmy odpowiedni charakter powierzchni, stosując klasę `Cast` oraz `BitmapMaterial`.

Przy wybieraniu nowej pozycji dla obiektów za śladem kursora podąża niebieski celownik. Podobnie jak w grach strategicznych, pokazuje on nowe docelowe położenie na planszy. Do jego wykonania zastosowaliśmy obiekt klasy `Plane` oraz teksturę `MovieMaterial`. Źródłem dla tego materiału jest osadzony w zasobach pliku *fla* element `MovieClip`.

Wewnątrz klasy `MouseEvent3DExample` zadeklarowaliśmy kilkanaście obiektów i zmiennych potrzebnych do prawidłowego funkcjonowania przykładu. Jak zwykle zaczęliśmy od zdefiniowania widoku Away3D.

Wskaźnikiem nowych pozycji jest obiekt klasy `Plane` o nazwie `pointer`. Dla źródła materiału wykorzystaliśmy obiekt klasy `Crosshair`. `Crosshair` nie zawiera w sobie żadnych dodatkowych metod poza konstruktorem. Służy jedynie jako łącznik z elementem `MovieClip` z biblioteki pliku *fla*.

Tego typu połączenia mogą wydawać się zbędne. Jednak gdyby projekt w swoich założeniach wymagał dodatkowych operacji wykonywanych na wskaźniku, dobrze byłoby umieścić je w osobnej klasie, takiej jak `Crosshair`, i odwoływać się do nich poprzez obiekt tej klasy.

W następnej kolejności zadeklarowaliśmy obiekty potrzebne do wyświetlenia planszy. Do stworzenia powierzchni zastosowaliśmy obiekt klasy `Plane` o nazwie `ground`, a do pokrycia jej teksturą użyliśmy obiektu `texture` klasy `BitmapMaterial`. Przyjmując, że do samego otoczenia moglibyśmy dodać jeszcze inne elementy, wygenerowaną powierzchnię umieściliśmy w kontenerze `board`. W razie konieczności modyfikowania środowiska wystarczyłoby odwołanie się do tego obiektu.

W tym przykładzie za pobieranie modeli odpowiada obiekt o nazwie `modelLoader` klasy `Loader3D`. W przypadku budynku fabryki pobrana zawartość umieszczana jest w kontenerze o nazwie `landingDock`. Z kolei każdy utworzony statek kosmiczny przypisywany jest do obiektu `fighter` i dodawany do wektora `Vector.<Object3D>` o nazwie `fighters`.

Aby wyselekcjonować konkretny element z całej grupy dodanych modeli, zdefiniowaliśmy specjalne obiekty klasy `Object3D` o nazwach `selectedBuilding` i `selected`
`↪Vehicle`.

Do opanowania animacji złożonych z więcej niż jednej sekwencji posłużyliśmy się obiektem `animations` klasy `TimelineMax`.

Poza samymi obiektami zadeklarowaliśmy również szereg zmiennych potrzebnych do wykonywania operacji w konkretnych przypadkach. Wszystkie te właściwości

Kup książkę Poleć książkę

przyjmują wartości typu `Boolean` i odpowiadają za kliknięcie poszczególnych klawiszy klawiatury.

Do określania stanu klawiszy strzałek zastosowaliśmy zmienne: `leftArrowDown`, `rightArrowDown`, `upArrowDown` oraz `downArrowDown`, których wartości domyślnie równe są `false`. W zależności od tego, który klawisz został wciśnięty, odpowiadającej zmiennej przypisujemy nową wartość równą `true`.

Za zmianę wartości właściwości `ctrlDown` odpowiada wciśnięcie lub zwolnienie klawisza *Ctrl*. Z kolei klawisze + i – umieszczone w bloku klawiszy alfanumerycznych regulują wartości właściwości `zoomInDown` oraz `zoomOutDown`.

W konstruktorze uruchamiamy metody `init()`, z chwilą gdy obiekt `Stage` zostanie zainicjowany.

```
if (stage) init();
else addEventListener(Event.ADDED_TO_STAGE, init);
```

W metodzie `init()` w pierwszej kolejności wykonaliśmy wszystkie potrzebne operacje na obiekcie stołu montażowego. Ustawiliśmy jakość generowanego obrazu, wyrównanie oraz wyłączyliśmy skalowanie. Następnie dodaliśmy detektory dla zdarzeń zmiany rozmiaru, odświeżenia obrazu i użycia klawiatury.

```
stage.quality = StageQuality.LOW;
stage.scaleMode = StageScaleMode.NO_SCALE;
stage.align = StageAlign.TOP_LEFT;
removeEventListener(Event.ADDED_TO_STAGE, init);
addEventListener(Event.ENTER_FRAME, onEnterFrame);
stage.addEventListener(Event.RESIZE, onResize);
stage.addEventListener(KeyboardEvent.KEY_DOWN, onKeyDown);
stage.addEventListener(KeyboardEvent.KEY_UP, onKeyUp);
```

W dalszej części metody `init()` stworzyliśmy widok Away3D i zmieniliśmy ustawienia kamery tak, aby obejmowała całą powierzchnię planszy.

Do renderowania widoku skorzystaliśmy z klasy `BasicRenderer`, której obiekt utworzyliśmy, stosując odwołanie `Renderer.BASIC`. Przed umieszczeniem widoku na liście wyświetlanych obiektów wywołaliśmy metodę `addOnMouseMove()`, która przy zdarzeniu poruszania kursorem ma wywołać metodę `overScene()`.

```
view = new View3D();
view.renderer = Renderer.BASIC;
view.scene.addOnMouseMove(overScene);
addChild(view);
view.camera.zoom = 15;
view.camera.y = 800;
view.camera.lookAt(new Vector3D(0, 0, 0));
```

Kup książkę Poleć książkę

Po dodaniu widoku w celach testowych umieściliśmy obiekt klasy `Trident`. Długość jego ramion ustawiliśmy na poziomie 100 pikseli i zmieniliśmy wartość właściwości `showLetters` na `true`.

Linijkę kodu dodającą oś `Trident` do sceny Away3D umieściliśmy w komentarzu tak, aby nie wyświetlać obiektu `trident` bez potrzeby.

```
trident = new Trident(100, true);
//view.scene.addChild(trident);
```

Dalej w metodzie `init()` stworzyliśmy wskaźnik pozycji, który wcześniej zdefiniowaliśmy jako obiekt `pointer`. Jego wysokość oraz szerokość ustawiliśmy tak, aby zgadzały się z wymiarami źródła grafiki `pointerMC`. Ze względu na swoją rolę wskaźnik jest zawsze widoczny na powierzchni planszy. Aby uzyskać taki efekt, nie podnieśliśmy obiektu na osi Y, tylko zastosowaliśmy właściwość `pushfront`. Przypisanie jej wartości `true` powoduje wyświetlenie siatki wybranego obiektu nad pozostałymi znajdującymi się w tej samej pozycji Y.

```
pointer = new Plane( { width:34, height:34, material:new
MovieMaterial(pointerMC),pushfront:true } );
```

Na końcu metody `init()` odwołaliśmy się do dwóch metod tworzących potrzebne modele oraz metody ustawiającej pozycję widoku.

```
initBoard();
initBuildings();
onResize();
```

W metodzie `initBoard()` poza stworzeniem planszy i nadaniem jej odpowiednich ustawień ważne było zastosowanie metody uruchamiającej metodę `onGroundClick()` przy każdym kliknięciu na powierzchni planszy.

```
ground.addOnMouseDown(onGroundClick);
```

Metoda `overScene()` wywoływana jest w sytuacji poruszania kursorem w przestrzeni sceny. W tym przypadku interesowały nas jedynie współrzędne osi X oraz Z, które przypisaliśmy do pozycji obiektu `pointer`.

```
pointer.x = e.sceneX;
pointer.z = e.sceneZ;
```

W metodzie `onGroundClick()` zapisaliśmy proces przemieszczania wybranego obiektu. Wewnątrz ciała metody w pierwszej kolejności znajdują się instrukcje warunkowe, które sprawdzają, czy został wciśnięty klawisz *Ctrl* oraz jakiego rodzaju jest kliknięty obiekt. Zdefiniowaliśmy dwie możliwe sytuacje. Pierwsza dotyczy kliknięcia obiektu `landingDock` reprezentującego budynek fabryki. W tym przypadku przejście między pozycjami jest natychmiastowe, ponieważ nie zastosowaliśmy tutaj klasy `TweenMax`, tylko od razu przypisaliśmy właściwościom `x` i `z` nowe

Kup książkę Poleć książkę

wartości pozycji wskaźnika `pointer`. Poza tym przywróciliśmy pierwotny stan przezroczystości obiektu `landingDock`, wyczyściliśmy zawartość obiektu `selected↪Building` i usunęliśmy ze sceny wskaźnik.

```
if (ctrlDown && selectedBuilding)
{
   selectedBuilding.alpha = 1;
   selectedBuilding.x = pointer.x;
   selectedBuilding.z = pointer.z;
   selectedBuilding = null;
   view.scene.removeChild(pointer);
}
```

Druga sytuacja określona w metodzie `onGroundClick()` dotyczy pojazdów latających, czyli obiektów `fighter`. W ich przypadku aby zmienić pozycję, stosuje się klasę `TweenMax`, w której właściwościom `x` i `z` przypisaliśmy nowe wartości położenia wskaźnika. Dodatkowo wewnątrz metody `to()` klasy `TweenMax` zapisaliśmy metodę, która zostanie wywołana na koniec animacji. Celem tej metody jest usunięcie ze sceny obiektu `pointer` oraz wyczyszczenie odwołania do klikniętego pojazdu w obiekcie `selectedVehicle`.

```
else if (ctrlDown && selectedVehicle)
{
   TweenMax.to(selectedVehicle, 1, { x:pointer.x, z:pointer.z,
   onComplete:function()
   {
      view.scene.removeChild(pointer);
      selectedVehicle = null;
   }
   } );
}
```

Aby pobrać z wybranej lokalizacji model fabryki w metodzie `initBuildings()`, skorzystaliśmy z obiektu `modelLoader` oraz klasy `Max3DS`. Dodatkowo użyliśmy detektora zdarzeń `Loader3DEvent`, aby przy zakończeniu pobierania wywołać metodę `onLandingDockSuccess()`.

```
modelLoader =
Max3DS.load('../../resources/models/max3ds/landingDock/LandingDock.3DS');
modelLoader.addOnSuccess(onLandingDockSuccess);
```

Dopiero w metodzie `onLandingDockSuccess()` zapisaliśmy proces dodawania modelu do sceny Away3D. W pierwszej kolejności nadaliśmy mu nazwę oraz umieściliśmy zawartość obiektu `modelLoader` w kontenerze `landingDock`. Ponieważ w swoich pierwotnych ustawieniach wymiary modelu są znacznie większe od wymiarów dodanej planszy, musieliśmy zastosować metodę `scale()` do zmniejszenia skali obiektu `landingDock`.

```
landingDock = new ObjectContainer3D();
landingDock.name = 'RED_landingDock';
```

Kup książkę Poleć książkę

```
landingDock.addChild(modelLoader.handle);
landingDock.scale(.2);
```

W kolejnych linijkach ustawiliśmy model w wybranym miejscu i przypisaliśmy mu filtr czerwonej poświaty, który będzie się pojawiał po każdym najechaniu kursora na obiekt.

```
landingDock.x = 71;
landingDock.z = 196;
landingDock.y = 15;
landingDock.filters = [ new GlowFilter(0xFF0000, 0, 7, 7, 1, 1)];
```

Aby poświata była widoczna na obiekcie 3D, należy przypisać jego właściwości `ownCanvas` wartość `true`.

Do rozpoznawania akcji myszy na powierzchni obiektu `landingDock` zastosowaliśmy trzy metody zdarzeń `MouseEvent3D`.

```
landingDock.addOnMouseDown(landingDockOnMouseDown);
landingDock.addOnRollOver(landingDockOnMouseRollOver);
landingDock.addOnRollOut(landingDockOnMouseRollOut);
```

Kliknięcie na obiekt `landingDock` powoduje uruchomienie metody `landingDockOn`
`↪MouseDown()`. W jej ciele sprawdzamy, czy wartość właściwości `ctrlDown` jest równa `true`. Jeżeli warunek jest spełniony, to metoda ta spowoduje zmianę pozycji obiektu fabryki i wyświetlenie wskaźnika.

```
if (ctrlDown)
{
   selectedBuilding = landingDock;
   selectedBuilding.alpha = .5;
   view.scene.addChild(pointer);
   pointerMC.gotoAndStop('move');
}
```

W sytuacji gdy wartość właściwości `ctrlDown` jest równa `false`, celem metody `landingDockOnMouseDown()` jest stworzenie nowego obiektu statku kosmicznego. Proces ten zaczyna się podobnie jak tworzenie modelu fabryki. W pierwszej kolejności za pomocą klasy `Max3DS` pobierany jest z określonej lokalizacji model *SpaceFighter01.3DS*. Następnie z chwilą zakończenia pobierania wywołujemy metodę `onModelLoaderSuccess()`.

```
else
{
   modelLoader = Max3DS.load('./models/sf/spaceships/SpaceFighter01.3DS');
   modelLoader.addOnSuccess(onModelLoaderSuccess);
}
```

W metodzie `onModelLoaderSuccess()` zapisaliśmy cały proces tworzenia i pokazania modelu statku kosmicznego. Według scenariusza obiekt ma się pojawić na powierzchni budynku, zwiększyć swoją pozycję na osi Y i przenieść w przypadkowe miejsce wokół fabryki.

Żeby uzyskać taki efekt, w pierwszej kolejności zdefiniowaliśmy dwie zmienne numeryczne: `randomX` i `randomY`, których wartości są różnicą pozycji modelu fabryki i losowej liczby z przedziału. Do wyznaczenia przypadkowej liczby napisaliśmy osobną metodę `randomPosition()`, którą omówimy później.

```
var randomX:Number = landingDock.x - randomPosition(-150, 150);
var randomZ:Number = landingDock.z - randomPosition(-150, 150);
```

W dalszej części metody `onModelLoaderSuccess()` stworzyliśmy obiekt `fighter`, który przechowuje pobraną zawartość obiektu `modelLoader`.

```
fighter = new ObjectContainer3D();
fighter.addChild(modelLoader.handle);
```

Kolejność definiowania właściwości nie ma tutaj większego znaczenia, dlatego zaczęliśmy od dodania detektorów zdarzeń `MouseEvent3D`. Tak samo jak w przypadku obiektu `landingDock`, na każdy ze statków kosmicznych będzie można kliknąć, najechać i opuścić kursorem powierzchnię modelu.

```
fighter.addOnMouseDown(fighterOnMouseDown);
fighter.addOnRollOver(fighterOnMouseRollOver);
fighter.addOnRollOut(fighterOnMouseRollOut);
```

Następnie dodaliśmy filtr czerwonej poświaty, który będzie się pojawiał, gdy kursor znajdzie się na powierzchni modelu.

```
fighter.filters = [new GlowFilter(0xFF0000, 0, 7, 7, 1, 1)];
```

Ponieważ pojazd ma się wyłaniać z hangaru, dla animacji ustawiliśmy jego pozycję zgodną z położeniem fabryki oraz skalę na poziomie jednej tysięcznej.

```
fighter.scale(.001);
fighter.x = landingDock.x;
fighter.y = 20;
fighter.z = landingDock.z;
```

Po utworzeniu statku i przypisaniu jego właściwościom nowych współrzędnych dodaliśmy go do sceny oraz obiektu `fighters`.

```
view.scene.addChild(fighter);
fighters.push(fighter);
```

Na końcu metody `onModelLoaderSuccess()` stworzyliśmy animację złożoną z dwóch etapów. W pierwszym powiększyliśmy skalę obiektu, co ma symbolizować jego powstawanie, w drugiej zaś przenieśliśmy go do losowych pozycji `x` i `z` na wysokości 100 pikseli.

Kup książkę Poleć książkę

```
animations = new TimelineMax( { onComplete:onFighterReady } );
animations.append(TweenMax.to(fighter, 1, { scaleX:.4, scaleY:.4,
↪scaleZ:.4 } ));
animations.append(TweenMax.to(fighter, 2, { x:randomX, y: 100,
↪z:randomZ } ));
animations.play();
```

Do tworzenia animacji złożonej z kilku etapów najlepiej stosować klasę `TimelineMax`. Poszczególne fazy animacji dodaje się metodą `append()`, w której argumentem jest odwołanie do klasy `TweenMax`. Gotową sekwencję animacji można uruchomić, stosując metody `play()` lub `restart()`.

Szczegóły dotyczące klasy `TweenMax` znajdują się na stronie: *http://www.greensock.com/timelinemax/*.

Metoda `randomPosition()`, jak wspomnieliśmy wcześniej, służy do wygenerowania losowej liczby. Do wyliczenia tej wartości posłużyliśmy się metodami klasy `Math` oraz przedziału liczbowego podanego w argumentach `min` i `max`.

```
return Math.floor(Math.random() * (1 + max - min)) + min;
```

W metodach `landingDockOnMouseRollOver()` i `landingDockOnMouseRollOut()` zapisaliśmy animację pokazywania i chowania poświaty. Aby płynnie przejść od jednego stanu do drugiego, skorzystaliśmy z klasy `TweenMax`, w której odwołujemy się do pierwszej pozycji z tablicy `filters` obiektu `landingDock` i zmieniamy wartość właściwości `alpha`.

```
//landingDockOnMouseRollOver
TweenMax.to(landingDock.filters[0], .5, { alpha:1 } );

//landingDockOnMouseRollOut
TweenMax.to(landingDock.filters[0], .5, { alpha:0 } );
```

Na tej samej zasadzie zakodowaliśmy pojawianie się poświaty wokół obiektów `fighter` w metodach `fighterOnMouseRollOver()` i `fighterOnMouseRollOut()`. Jedyną różnicą jest odwołanie do wybranego obiektu. Ponieważ w tym przypadku liczba pojazdów latających może być większa niż jeden, aby animacja dotyczyła konkretnego obiektu, musieliśmy w klasie `TweenMax` użyć zapisu `e.currentTarget.filters[0]`.

Metoda `fighterOnMouseDown()` odpowiada za pojawienie się wskaźnika i rozpoczęcie procesu przemieszczania obiektu latającego. Żeby sprawdzić, czy użytkownik, klikając na obiekt, chce go przenieść w inne miejsce, zastosowaliśmy instrukcję `if` sprawdzającą, czy zmienna `ctrlDown` jest równa `true`. Jeżeli ma ona wartość `true`, na scenie w pozycji kursora pojawia się wskaźnik. Żeby przejście dotyczyło konkretnego pojazdu, przypisaliśmy obiektowi `selectedVehicle` odwołanie do kliknięty pojazdu.

```
if (ctrlDown)
{
```

```
        selectedVehicle = e.target as ObjectContainer3D;
        view.scene.addChild(pointer);
        pointerMC.gotoAndStop('move');
    }
```

Przy każdej zmianie rozmiaru okna uruchamiana jest metoda `onResize()`. W jej ciele zapisaliśmy wyśrodkowanie widoku `view` w oknie programu.

```
view.x = stage.stageWidth * .5;
view.y = stage.stageHeight * .5;
```

Znane nam z wcześniejszych przykładów metody `onKeyDown()` oraz `onKeyUp()` służą do sprawdzania, czy zaszły warunki zapisane w instrukcjach `if`, i przypisania odpowiedniej wartości poszczególnym właściwościom. W obu metodach tego przykładu sprawdzaliśmy akcje wciśnięcia i zwolnienia klawiszy strzałek, *Ctrl* oraz znaków plusa i minusa znajdujących się w górnym rzędzie klawiatury. W metodzie `onKeyUp()` przy zwolnieniu klawisza *Ctrl* dodatkowo sprawdziliśmy, czy kursor znajdował się na obiekcie `landingDock`. Jeżeli tak, to musieliśmy zapisać powrót do pierwotnego stanu poziomu jego przezroczystości, wyczyścić zawartość obiektu `selectedBuilding` i usunąć wskaźnik ze sceny.

```
if (e.keyCode == Keyboard.CONTROL)
{
    ctrlDown = false;
    if (selectedBuilding)
    {
        selectedBuilding.alpha = 1;
        selectedBuilding = null;
        view.scene.removeChild(pointer);
    }
}
```

Na końcu zapisaliśmy metodę `onEnterFrame()`, która wywoływana jest przy każdym wystąpieniu zdarzenia `Event.ENTER_FRAME`. Zawiera ona w sobie kilka instrukcji warunkowych. W każdej z nich dodatkowo sprawdzamy, czy został wciśnięty klawisz *Ctrl*.

Zastosowanie kombinacji klawiszy pozwoliło nam uniknąć korzystania z większej ich liczby.

Pierwsza instrukcja `if` sprawdza, czy wartość zmiennej `zoomInDown` równa jest `true`. Jeżeli warunek jest spełniony, to w zależności od stanu zmiennej `ctrlDown` wykonywana jest odpowiednia operacja. Gdy klawisz *Ctrl* jest wciśnięty, to kamera zmienia swoją pozycję na osi Y. W przeciwnym razie zwiększana jest wartość właściwości `zoom`.

```
if (zoomInDown)
{
    if (ctrlDown) view.camera.y += 5;
```

Kup książkę Poleć książkę

```
    else view.camera.zoom++;
}
```

W przypadku drugiej instrukcji `if` sprawdzana jest wartość zmiennej `zoomOutDown`. Gdy wartość tej właściwości jest równa `true`, to kamera może zmieniać swoją pozycję na osi Y lub zmniejszać wartość właściwości `zoom`.

```
if (zoomOutDown)
{
    if (ctrlDown) view.camera.y -= 5;
    else view.camera.zoom--;
}
```

Trzecia instrukcja `if` dotyczy zdarzenia wciśnięcia klawisza lewej strzałki. Jeżeli warunek jest spełniony, to kamera może wykonywać obrót w lewą stronę lub zmniejszyć pozycję na osi X.

```
if (leftArrowDown)
{
    if (ctrlDown) view.camera.rotationY--;
    else view.camera.x -= 10;
}
```

Działania czwartej instrukcji warunkowej `if` są odwrotne do poprzedniej. Dotyczy ona bowiem wciśnięcia klawisza prawej strzałki. Gdy wartość właściwości `right ↪ArrowDown` jest równa `true`, to kamera może obracać się w prawą stronę bądź zwiększać pozycję na osi X.

```
if (rightArrowDown)
{
    if (ctrlDown) view.camera.rotationY++;
    else view.camera.x += 10;
}
```

W piątej instrukcji `if` sprawdzana jest wartość zmiennej `upArrowDown`. Jeżeli warunek jest spełniony, to kamera może wykonywać obrót w górę lub zwiększyć pozycję na osi Z.

```
if (upArrowDown)
{
    if (ctrlDown) view.camera.rotationX++;
    else view.camera.z += 10;
}
```

W ostatniej instrukcji `if` warunkiem jest wciśnięcie klawisza dolnej strzałki. Gdy wartość właściwości `downArrowDown` jest równa `true`, to kamera może wykonać obrót w dół bądź zmniejszyć pozycję na osi Z.

```
if (downArrowDown)
{
```

Kup książkę Poleć książkę

```
        if (ctrlDown) view.camera.rotationX--;
        else view.camera.z -= 10;
    }
```

Na końcu metody `onEnterFrame()` wywołaliśmy odświeżenie wygenerowanego obrazu metodą `render()`.

Podsumowanie

- Do kontrolowania obiektów można korzystać zarówno z myszki, jak i klawiatury.
- Klasa `KeyboardEvent` reprezentuje zdarzenia związane z używaniem klawiatury. Rozpoznaje procesy wciśnięcia i zwolnienia klawisza.
- Aby wykrywać akcje klawiatury, należy dodać detektor bezpośrednio do obiektu klasy `Stage`.
- Do rozpoznawania wciśniętego klawisza służą właściwości `keyCode` i `charCode`. Pierwsza właściwość zwraca wartość numeryczną położenia klawisza na klawiaturze, a druga określa kod znaku wciśniętego klawisza.
- Przy kontrolowaniu zdarzeń użycia klawiatury przydatna jest klasa `Keyboard`, która między innymi ma zdefiniowane stałe określające kody standardowych znaków klawiszy.
- Klasa `KeyLocation` służy do określenia lokalizacji klawisza, ma stałe określające strefy standardowej klawiatury.
- W metodach wywołanych zdarzeniami klawiatury warto stosować dodatkowe zmienne o wartościach typu `Boolean`, określające status poszczególnego klawisza. Jeśli stosujemy te zmienne w metodach wywołujących animację, będzie możliwe tworzenie konkretnych efektów dla różnych kombinacji wciśniętych klawiszy.
- Zdarzenia `MouseEvent` nie działają bezpośrednio na obiekcie trójwymiarowym.
- Na materiałach interaktywnych można stosować zdarzenia klasy `MouseEvent`.
- Właściwość `interactive` w materiałach służy do określenia, czy mają być stosowane zdarzenia myszki.
- Zdarzenia `MouseEvent` wywoływane na obiekcie klasy `Stage` można wykorzystać do zmiany kątów nachylenia poszczególnych obiektów na scenie.

Kup książkę Poleć książkę

- Biblioteka Away3D ma własną klasę do obsługi zdarzeń myszy o nazwie `MouseEvent3D`.
- Stosując współrzędne UV, można umieszczać elementy w konkretnym miejscu tekstury typu `MovieMaterial`.
- Aby filtry typu `Blur` lub `Glow` były widoczne na elementach, trzeba zastosować właściwość `ownCanvas` na wybranym obiekcie i przypisać mu wartość `true`.

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Skrowidz

A

B

Kup książkę Poleć książkę

C

D

E

F

G

Kup książkę Poleć książkę

H

I

J

K

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

L

Ł

M

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

N

O

Kup książkę Poleć książkę

P

Kup książkę Poleć książkę

Kup książkę Poleć książkę

R

S

Kup książkę Poleć książkę

Ś

T

Kup książkę Poleć książkę

U

V

W

Kup książkę Poleć książkę

X

Z

Kup książkę Poleć książkę

Kup książkę Poleć książkę

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION

1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!
http://program-partnerski.helion.pl

GRUPA WYDAWNICZA
Helion SA

WKROCZ Z FLASHEM
W ŚWIAT INTERAKTYWNYCH APLIKACJI 3D!

Technologia Flash na dobre zagościła w świecie interaktywnych animacji i gier komputerowych 2D. Stała się również (obok HTML5) standardem prezentacji tego rodzaju treści na stronach WWW. Taki stan rzeczy to efekt połączenia samego Flasha z Actionscriptem, językiem programowania ułatwiającym sterowanie pracą aplikacji i reagowanie na działania podejmowane przez użytkownika. Jednak użytkownicy internetu oczekują obecnie czegoś więcej niż tylko zwykłe animacje i gry 2D – oczekują interaktywnych treści 3D, ponieważ możliwości odtwarzania tego typu materiałów są coraz większe. Do tej pory brakowało narzędzi do tworzenia takich aplikacji, a także technologii, które je wspomagały. Na szczęście z odsieczą przychodzi Flash, który śmiało wkracza w świat 3D, oferuje ciekawe biblioteki i nie ma przy tym praktycznie żadnej konkurencji.

Jeśli znasz już trochę Flasha, a teraz chcesz poszerzyć swoją wiedzę o znajomość szybko rozwijającej się technologii 3D, sięgnij po książkę *Flash i Actionscript. Aplikacje 3D od podstaw*. Prostym, lecz precyzyjnym językiem przedstawiono w niej najważniejsze kwestie związane z projektowaniem i tworzeniem aplikacji 3D z wykorzystaniem technologii Flash i języka Actionscript, zaprezentowano sposoby używania bezpłatnej biblioteki Away3D oraz pokazano, jak za jej pomocą budować i optymalizować własne rozwiązania.

- Wybór, pobieranie i instalacja właściwej biblioteki 3D dla technologii Flash
- Podstawowe komponenty biblioteki Away3D i budowanie aplikacji za jej pomocą
- Używanie materiałów i korzystanie z różnych rodzajów świateł
- Tworzenie, importowanie i stosowanie modeli oraz animacji 3D w aplikacjach Flash
- Zapewnianie interakcji aplikacji z użytkownikiem i obsługa urządzeń wejściowych
- Definiowanie kamer z perspektywy pierwszej i trzeciej osoby oraz ich używanie
- Obsługa dźwięku i elementów tekstowych

Interaktywne aplikacje 3D to przyszłość.
Zacznij ją tworzyć już dziś!

sięgnij po WIĘCEJ

KOD KORZYŚCI

helion.pl
księgarnia internetowa

Nr katalogowy: 7864

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

ISBN 978-83-246-3065-3

9 788324 630653

Cena: 99,00 zł

Informatyka w najlepszym wydaniu