Doing a *flash* animation for animB

Mathieu Clabaut

May 22, 2008

Introduction

This tutorial should provide the user with a tour through the most important functionalities allowing to build a flash animation connected with RODIN *animb* plugin, so that he gets a understanding of how the things work.

Contents

1 Context

This tutorial is based on the following assumption:

- the tool use is *Adobe Flash CS3 Professional* localized in English and install on *Windows* OS.
- the *animb* component version is `v0.04`,
- the animated B model contains one machine `mch_0`,
- this machine contains:
  - one variable `SELECTION` of type `MODE`
  - and one event `DRIVER`:

    ```
    DRIVER ≡
        SELECTION :∈ MODE
    ```
- The `MODE` set is valuated in the animator with the `"s1"`,`"s2"` and`"s3"` elements.

2 Overview

Figure 1: Overview of Flash and Rodin Interactions

3 Flash component installation

Get the flash component which is a zip file named `animbAS3`, unzip it in a `animbAS3` directory. [s3.1]

Close Adobe Flash CS3 to ensure a clean installation. [s3.2]

Launch the installation script by double clicking on `animbAS3/install`. After a few seconds, *Adobe Flash CS3* will be launched. Follow the installation directions given by the tool. [s3.3]

Close and restart *Adobe Flash CS3* in order to restart from a clean environment. [s3.4]

4 New Flash Animation

Create a new animation from a template by clicking on *File* > *New...* > *Templates*, choosing *AnimB* category and then *AnimB* template. Press OK. [s4.1]

A new empty animation is then created as shown in figure 2 on the following page. It is composed of three layers:

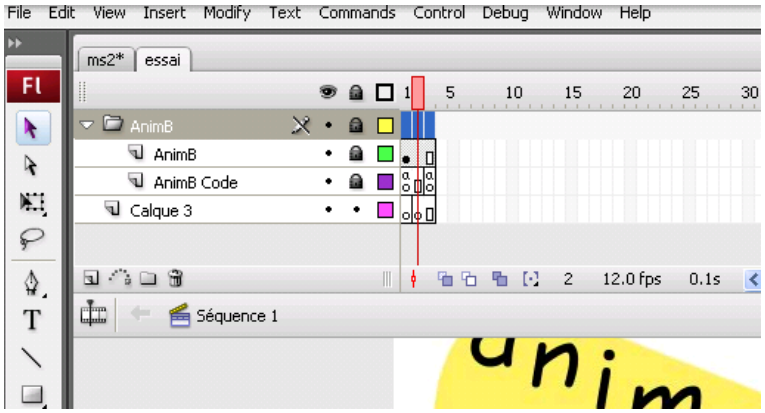


Figure 2: New animation created from *AnimB* template

1. `AnimB`, which contains the *animb* flash component implementing the connexion between flash and the eclipse animation,
2. `AnimB code`, which contains the associated action script code:
   - the code in the first frame is responsible for registering the launch of the animation upon the reception of the `animationInit` event and then stopping the natural flow on the animation,
   - the code in the third frame which is executed once the animation started, loads the state of the eclipse animation into the flash environment.
3. `Calque 3` which is an empty layer, ready to welcome your own work.

The organization of the timeline is explained in section A.3.2 on page 12.

Unlock the AnimB layer by clicking on the associated lock symbol. [s4.2]

Configure the *animb* component by selecting the *AnimB* layer, the first frame and then the *animb* component. [s4.3]

In the *parameter* panel, enter the *animation name*, *project name* and the *model name*[1] as given in the Rodin platform.

Lock the AnimB layer by clicking on the associated dot, below the lock symbol. [s4.4]

Save your animation by clicking on *File* > *Save*. [s4.5]

Test your settings by clicking on *Control* > *Test Movie* or by pressing ***Ctrl+Enter***. You should see after some time a window popping up and briefly showing the *animb* logo before going blank. [s4.6]

Don't forget to launch your animation under Rodin first!

[1]The "model name", is the name of the machine which is animated under Rodin.

5 Addition of Graphic Elements

We will add a new graphic element arbitrary named *selector* which has three states `s1`, `s2`, `s3` corresponding to three distinct frames (from 2 to 4) and whose current state can be changed by the user clicking on buttons.

The planned architecture of the animation is given in figure 3.

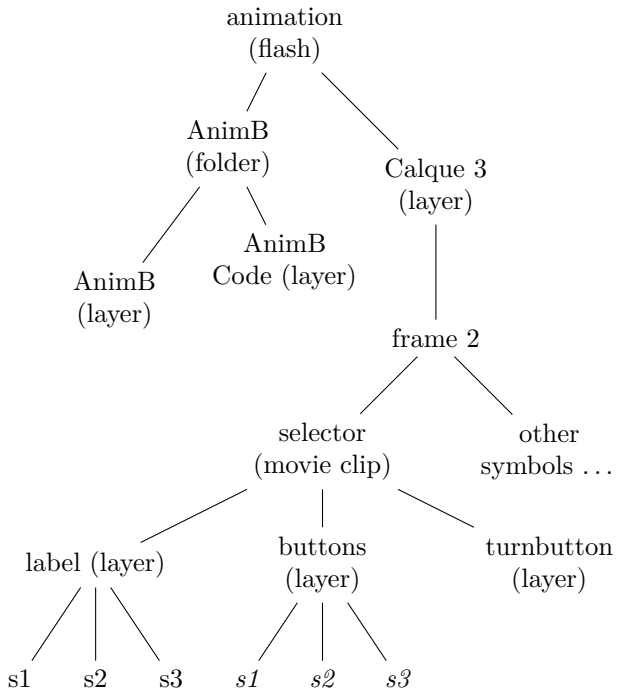


Figure 3: Planned animation architecture

5.1 Prepare the B Model

Add an observer to the event B model In order to let the animation be updated by the animator, we must define an expression or a predicate whose changes will triggered an event on the flash side. Under Rodin, go into the synthesis view of the machine and click on the *Create a new observed predicate* icon or *Create a new observed expression* icon . [s5.1]

5.2 Create a New Animated Object

Create a new symbol of type *Movie clip* in the second frame, by selecting the second frame and clicking on *Insert* > *New Symbol...* or ***Ctrl+F8***. Name it `selector` and press OK. [s5.2]

The selector will be composed of 3 layers (buttons, labels, turnbutton) each composed of4 frames. The first for the actionsScript initialisation, and the three others for the three different states of the selector.

Insert two more layers by clicking on *Insert* > *Timeline* > *Layer* (or on the *Insert Layer* icon). Name them respectively `labels`, `buttons` and `turnbutton`. [s5.3]

Create a button object by selecting the first frame in the `buttons` layer, pressing ***Ctrl+F8*** and choosing a *Button* type (see section A.3.1 on page 11. Name it `rectbut` and press OK. [s5.4]

Select the *Rectangle Tool* and draw a rectangle centered on the cross hair. Change its border and background colors to be mostly transparent (alpha of 30%). Right click on the *Hit* frame, and select *Insert Frame*.

Create three instances of the *rectbut* object by going back to the `selector` object, clicking on the associated button on the bar below the timeline. Then drag a *rectbut* from the *Library* panel on the right of the workspace, and drop it where appropriate on the current stage. Name it `s1`. [s5.5]

Redo the operation two times and name the new buttons `s2` and `s3`. You may have a look at the result in figure 4 on page 9 for inspiration.

Keep the button visible until frame four by right clicking on the fourth frame and selecting *Insert Frame*. [s5.6]

Create the action script listener associated to the button by selecting the first frame, pressing ***F9*** and pasting in code excerpt 1 on the next page. Note that *SELECTION* is the event being monitored via the *v_SELECTION* observer as specified in s5.1, *mch_0* is the name of the machine being animated and *DRIVER* is the event being triggered (see section 1 on page 3). [s5.7]

[NOTE] By putting the listener setup in the first frame and doing an animation between the frame two and four, we prevent the listener setup script from being run at each animation loop.

[NOTE] `this` in code excerpt 1 on the next page refers to the `selector` object. If the action script where to be located at the upper level, object `s1` would have been referred to by something like `root.selectorInstanceName.s1`.

[NOTE] See section B.5 on page 14 for how to pass the parameter for a B event composed of an `ANY` substitution instead of a $:\in$ or $:|$ substitution.

Put the three associated label by selecting the second frame of the *labels* layer, inserting a keyframe in the second frame. Select the *Text* tool and type the three labels *s1*, *s2* and *s3*. [s5.8]

[NOTE] You can lock the 2 other layer by clicking on the dot below the lock symbol. It will ease label placement relatively to the buttons.

Insert two more identical frames by right clicking on the fourth frame and selecting *Insert Frame*. [s5.9]

Draw the selector device by selecting the first frame of the *turnbutton* layer, inserting a key frame. Choose the *Oval Tool* by clicking and holding the *Rectangle Tool* button on the toolbar (by default on the left of the workspace) or pressing ***O***. Draw a circle, filled with black. [s5.10]

```
import flash.events.MouseEvent;
import org.animb.XMLEvent;

  //add the listener for click event for each button
this.s1.addEventListener(MouseEvent.CLICK, clicks1);
this.s2.addEventListener(MouseEvent.CLICK, clicks2);
this.s3.addEventListener(MouseEvent.CLICK, clicks3);

  //Declare the listener for each button click
function clicks1(o:Object):void {
        var param:Array = new Array();
        param["SELECTION'"] = "s1"; // setup parameter
        //Execute the B model event on the Rodin platform
        root.animb.execEventWithParam("mch_0", "DRIVER", param);
}

function clicks2(o:Object):void {
        var param:Array = new Array();
        param["SELECTION'"] = "s2";
        root.animb.execEventWithParam("mch_0", "DRIVER", param);

}

function clicks3(o:Object):void {
        var param:Array = new Array();
        param["SELECTION'"] = "s3";
        root.animb.execEventWithParam("mch_0", "DRIVER", param);
}



  /* Add a listener for event v_SELECTION_change coming from the
  B model */
root.animb.addEventListener("v_SELECTION_change", v_SELECTION_change);

  /* Declare the behaviour of the Movie Clip depending on the
  parameter event value */
function v_SELECTION_change(e:XMLEvent):void {
        switch (e.currString) {
                case "s1" :
                        gotoAndStop(2); //Goto frame 2 and stop
                        break;
                case "s2" :
                        gotoAndStop(3);
                        break;
                case "s3" :
                        gotoAndStop(4);
                        break;

        }
}
```

Code excerpt 1: Buttons listener setup

Add a little rectangle on the upper side of your circle after having pressed the ***R***.

[NOTE] By default you're drawing shapes. As you've probably noticed, the circle and the rectangle were merge in a single shape as soon as the rectangle was drawn. If you want to edit the piece of the selector button afterward, you should draw them as objects. You can do so by pressing the ***J*** before drawing them.

Copy the first frame over the next three frames by right clicking on the first frame and selecting *Copy Frames* and the right clicking on the second frame and selecting *Paste Frames*. Repeat on the third and fourth frame. [s5.11]

[NOTE] Only the three last frames are used by the action script written in s5.7, but filing the first one is useful to understand the animation layout, as it is the only one that will be visible when inserting a *selector* instance on the main stage.

Rotate the selector button by selecting its rectangular area and moving it on the left or on the right, in both the third and the fourth frame. [s5.12]

As designed until now, the selector button animation will loop without interruption. We must stop the default flow of the animation to let it be controlled either by the user or by the B model animation.

Make the animation stop by adding `stop();` to the second frame of the *turnbutton* layer (hint: use ***F9***). Doing so, the buttons, labels and selector device will be drawn and the action script listener (step s5.7 on page 6) will be setup before the animation stopping. [s5.13]

Your *selector object* should (or may) look like the one given in figure 4 on the following page.

5.3 Add the New Object and Give a Try

Insert a selector object in the main stage by selecting the *Séquence 1* flash animation and dragging a *selector* object from the library panel to the main stage. [s5.14]

Run the animator on the Eclipse side . [s5.15]

MC: TODO

Test your animation by hitting ***Ctrl+Enter***. [s5.16]

A References

A.1 Eclipse Animator Messages and Flash AS API

The eclipse animator send messages to the flash component via XML messages. Each time such an XML message is received, the flash component raise a corresponding event.

The actions script class `org.animb.XMLEvent` provide the user with an interface to more easily fetch the content of those messages. Its public interface is given in code excerpt 2 on the next page.

[NOTE] Those getter functions are used like an attribute (i.e. without any parenthesis) in action scripts. For example `var n:Number = e.currInt;`

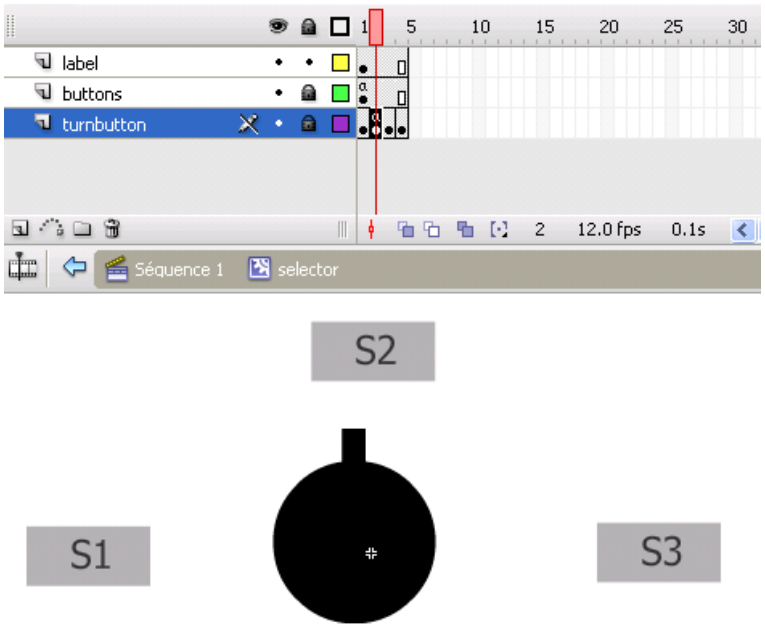


Figure 4: The *selector* object as designed in this tutorial

```
    public function get msg():XMLDocument;
    public function get currBool():Boolean;
    public function get prevBool():Boolean;
    public function get currString():String;
    public function get prevString():String;
    public function get currInt():String;
    public function get prevInt():String;
```

Code excerpt 2: org.animb.XMLEvent public interface

A.1.1 Errors and Warnings

On error or warning occurrences, the eclipse animator send the `error` or `warning` XML message which has the following forme:

```
<error text='...'/>
<warning text='...'/>
```

The flash component in turn raise the `error(e:XMLEvent)` or `warning(e:XMLEvent)` events.

A.1.2 Predicate Change

On *observername* predicate change, the eclipse animator send the `obervername_true` or `obervername_false` XML message which has one of the following form:

```
<observername_true />
<observername_false />
```

The flash component in turn raise the `obervername_true(e:XMLEvent)` or `obervername_false(e:XMLEvent)` events.

A.1.3 Expression Change

On *observername* expression change, the eclipse animator send the `observername_change` XML message which has one of the following form:

- For scalar expressions:

  ```
  <observername_change previous='...' current='...' />
  ```

- For set expressions:

  ```
  <observername_change
    <removed> setElements </removed>
    <added> setElements </added>
  />
  ```

  where `setElements` is a sequence of `setElement` defined as either:

  - `<mapplet> setElement setElement </mapplet>`
  - `<scalar  value='\ldots'/>`

The flash component in turn raise the `observername_change(e:XMLEvent)` event.

A.2 Flash Messages and Flash AS API

The actions script class `org.animb.AnimB` provides the user with the public interface given in code excerpt 3 on the following page.

A.3 Flash Concepts

A.3.1 Objects

Graphic Objects In Flash, graphic objects are items on the Stage. Flash lets you move, copy, delete, transform, stack, align, and group graphic objects.

```
  // connect the flash component to the eclispe B animator
public function connect();
  // request the state of the variables
public function loadInitState();

  // Execute an event
public function execEvent(modelName:String,
                          eventName:String):void ;
  // Execute an event with given parameter
public function execEventWithParam(modelName:String,
                                   eventName:String,
                                   parameters:Array)
  // Execute a random event
public function execRandomEvent();
  // Send a raw XML message to the eclipse B animator
public function sendXML(xmlString:String);
```

Code excerpt 3: org.animb.AnimB public interface

Shapes Shapes are one type of graphic object you can create in Flash. When you draw shapes that overlap each other in the same layer, the topmost shape cuts away the part of the shape underneath it that it overlaps. In this way, drawing shapes is a destructive drawing mode.

When a shape has both a stroke and a fill, these are considered separate graphic elements, which can be selected and moved independently.

Symbols A symbol is a graphic, button, or movie clip that you create once in the Flash authoring environment or SimpleButton and MovieClip classes. You can then reuse the symbol throughout your document or in other documents.

An action script class may alsobe attached to a Symbol. See Flash help section *Assigning a class to symbols in Flash.*

Movie Clip Symbols A *movie clip* is a flash object which embeds an animation. Several instances of one movie clip can be put in the stage, or in another movie clip (But note that a movie clip cannot be itself animated by a motion or shape tween).

In the context of the B event animator, it is generally convenient to only have movie clips and more generally other objects in the second frame of the *Calque 3* layer as defined by the *animb* template (instead of a full animation). It prevents any confusion between the things done in the frames of the *animB* folder and the frames of the *Calque 3* Layer.

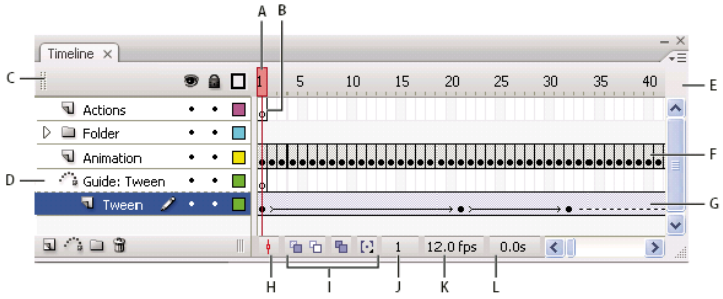
Button Symbols Buttons are actually four-frame interactive movie clips. Each frame in the timeline of a button symbol has a specific function:

- The first frame is the Up state, representing the button whenever the pointer is not over the button.
- The second frame is the Over state, representing the button's appearance when the pointer is over the button.

- The third frame is the Down state, representing the button's appearance as it is clicked.
- The fourth frame is the Hit state, defining the area that responds to the mouse click. This area is invisible in the final animation.

A.3.2 Workspace

Timeline A timeline description is given in figure 5



A Playhead
B Empty keyframe
C Timeline header
D Guide layer icon
E Frame View popup menu
F Frame-by-frame animation
G Tweened animation
H Scroll To Playhead button
I Onion-skinning buttons
J Current Frame indicator
K Frame Rate indicator
L Elapsed Time indicator

Figure 5: Timeline structure

Stage The stage is the panel where you can draw graphical objects. It shows the current selected frame according to the layer visibility and ordering.

B Useful Tips

B.1 Animations

Animations are done by playing frames consecutively. The *frame rate* could be set in the document properties panel.

B.1.1 Frame by Frame

Just draw frames content one after another.

The process may be eased by using the *onion skin* feature. After clicking the Onion Skin button , all frames between the Start Onion Skin and End Onion Skin markers (in the timeline header) are superimposed as one frame in the document window

B.1.2 Object Movement or Motion Tween

1. create one object in one layer,
2. create a key frame at the desired animation end,
3. modify the object on the last frame,
4. select a frame in between and right click on *Create Motion Tween*,
5. tweak parameters as desired.

B.1.3 Object Transformation or Shape Tween

1. create one object in one layer,
2. create a key frame at the desired animation end,
3. modify the object on the last frame,
4. select a frame in between and right click on *Create Shape, Tween*
5. tweak parameters as desired.

B.1.4 Collision Avoidance

MC: TODO

B.2 Synchronizing Flash and Rodin Animator

MC: rodin events are instantaneous. Flash animation may have duration. In case of controller/environment model, controller is supposed quicker than environment change...

B.3 Flash CS3 Usage

- When clicking, try to use modifier keys (***Ctrl***, ***Shift***, ***Alt***) alone or in combination to alter the tools behaviour.
- Use the Flash Help System often.

B.4 Debugging Your Animation

B.4.1 Traces from Animb Flash Component

You can display debug information provided by the *animb* flash component, by selecting it (in the *AnimB* layer; remove the lock and display the layer if necessary) and setting the *debug* parameter to `true`.

When running the animation, the *output panel* will display raw output from messages sent to and receive from the Rodin animator.

B.4.2 Traces from your Action Scripts

You may use the `trace` function to output specific traces onto the *ouptut panel*.

B.5 Trigger Event with Parameters

Suppose that instead of the following *DRIVER* event

```
SELECTION :∈ MODE
```

you have

```
ANY m
WHERE
  m : MODE
THEN
  SELECTION := m
```

The actions script listener associated to the button given in code excerpt 1 on page 7 is changed according to code excerpt 4:

```
function clicks1(o:Object):void {
    var param:Array = new Array();
    param["m"] = "s1";
    root.animb.execEventWithParam("mch_0", "DRIVER", param);
}
```

Code excerpt 4: Button action script revisited

C Animator plugin

C.1 Installation from Sourceforge

From CVS... MC: TODO

C.2 Basic Usage

In *Project Navigator* right-lick on the chosen machine and select *Animate*. MC: TODO

D Tables

Table of Code Excerpts

List of Figures