

ActionScript Tutorial:

Jigsaw Puzzle When Complete Turns Into Video

This Flash application involves the player putting together a jigsaw puzzle by dragging pieces on the screen. Pieces can be rotated by keeping the mouse button down and pressing the left or right arrows. Rotations are quantized to be 15 degrees for each push of an arrow button. After a piece is released (the mouse button lifted), a check is made if the jigsaw puzzle is put together properly. When the puzzle is correctly assembled, a video plays. The jigsaw puzzle image is the first frame of the video so the effect is that the completed puzzle comes to life. This application is based on the jigsaw application and the single video playing application described in other tutorials and repeats much of the other material. However, if you have done my other jigsaw tutorial, I include some comments about the differences between the two via side notes.

Challenges:

- Obtaining and Editing Video: This can be done using simple or advanced tools. I used a Canon camera and the Windows Movie Maker.
- Obtaining the image file that starts the video clip: I set up Corel Paint Shop Pro to do Screen Capture. I opened the video using Windows Media Player, made sure it was at the start and captured the first frame. This will be the picture on the puzzle itself.
- Slicing up the image: in order to have a puzzle, we must have individual pieces that can be manipulated separately as well as be able to be mixed up in a random fashion on the screen. The challenge is how can we retain the information necessary to re-construct the puzzle in order to check if the puzzle is put together? The solution will involve the recording of each piece's coordinates in the completed image.
- Conversion of the Video to .flv: Flash playback requires that video clips be in the .flv format. The Flash Video Encoder application (part of the Adobe Creative suite) is used to produce a .flv file.
- Sizing and Dimensions of the Puzzle and Video: It is necessary to develop code to adjust the width and height of the video clip to match the image. We will take the height and width dimensions from the base image.
- Positioning the video playback object where the image was put together AND on top of all the pieces: This is accomplished by using a method called "addChild" to the last piece as well as making an adjustment based on the offsets calculated for that last piece.
- What if the player clicks "mix up" or "build" while the video is playing?: We correct this by making enhancements so that if a pesky player clicks the mix up or build buttons while the video was playing it will automatically stop and then re-start from the beginning upon the puzzle being completed again.

- Giving the player the ability to drag and rotate individual pieces: This involves using `addEventListener` to set up the appropriate event handlers for `MouseDown`, `MouseUp` and `KeyDown`.
- Providing a function to check if the puzzle is put together: This checking must not be precisely exact because this would make it painstakingly difficult for the player to complete the puzzle and trigger the video. The “checkit” function has a built-in tolerance factor that allows for some leeway. This function also allows for the puzzle to be constructed on any angle and checkit will perform the check, accept the puzzle and play the video on that angle.

Game Structure:

The basic structure of the application is the same as my earlier jigsaw applications: there is a .fla file and an .as file. A class method called “setup” is called by the .fla file to pass information to the “Piece” class including the name of the video .flv file. This video file must be present during the testing of the application and if the application is published and uploaded it must also be present on the server.

Once you have a working application, you can produce a new jigsaw puzzle using the same Piece.as file with a different .fla file. This is because the mechanics of the puzzle lie in the .as file while the visual specifics lie in the .fla file.

[Sidenote: In my previous versions of the jigsaw application, I used the mixup method to do the initial setting up of the pieces on the screen. I decided that a better approach is to have a separate static class method called “setup” to handle all this. This method calls mixup so the game starts with the pieces jumbled on the screen]

Game Implementation:

First, open up a new Flash .fla file (Flash file) and a new .as file (ActionScript file).

Next, if you do not already have one, obtain a video clip. My clip was a .wmv so opening it in Windows made it play in the Windows Player. You need to capture a still of the first frame of your movie in order to maintain the illusion of the puzzle turning into a video. You can use Corel Paint Shop Pro to capture the video opening frame. Once in Paint Shop Pro, I selected the picture part and did Copy to the Clipboard. If on a PC you have the option of using the ‘print screen’ key. This key takes a copy of the screen and puts it into your clipboard. You can paste this into Photoshop, Paint, or any kind of photo editor, edit and save accordingly.

At some point, you must use Adobe Flash Video Encoder (or a Flash Video Encoder) to produce an .flv file from the video clip. Save this file in the same folder as the .fla file.

In the .fla file, Insert/New Symbol, select movie clip symbol, and name it ‘base’. You can do an Edit/Paste into the symbol the image of the first frame of your video. It is necessary to select the image and invoke Modify/Break apart two times so that the image can be cut up into pieces. Make the registration point (the crosshairs on the stage) the upper left corner if it is not that already.

The next step is to use the line or pencil tool to draw lines on the image. This breaks it into the jigsaw pieces.

Next we shall follow a procedure for each piece.

First, select a piece. Despite what is shown in the screen shot, I recommend starting from the top left for simplicity's sake.

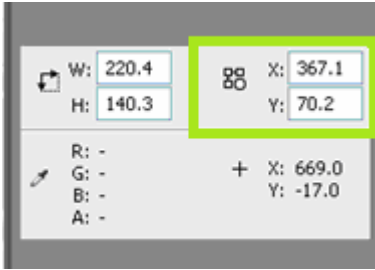


We shall follow a procedure for each piece.

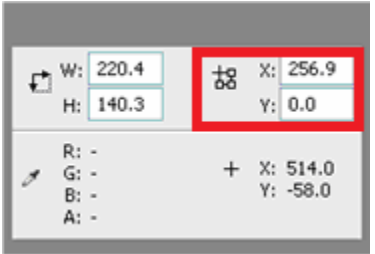
First, select a piece. Despite what is shown in the screen shot, I recommend starting from the top left for simplicity's sake.

Notice that the crosshair is the current registration point for this piece. It is necessary to record each piece's coordinates in relation to this crosshair. These coordinates will be what the computer uses to check if the puzzle is properly put together.

Make sure the Info panel is showing (Window/Info). Make sure also that the Info panel is showing three squares and a circle in the upper right (boxed in green).



If the following symbol is showing, this procedure will not work (boxed in red):



To switch it, click on the symbol itself.

Make sure one more time that the symbol in the upper right hand corner of the info panel in the one in the first picture, the one boxed in green.

Next, copy down the numbers in x and y of that section onto a text file or a piece of paper. In the case of this piece, I would write the following:

Piece3: X: 367.1 Y: 70.2

When assembling this information, I recommend recording and referring to your pieces in the following manner:



Next, Insert/New Symbol movie clip, give it a name (piece3 in the case of the piece selected in the above picture): and Paste into the new symbol the piece you just copied.

Ensure that the registration point is in the center of your piece. To do this. Set the X and Y in the upper right corner to 0:

The screenshot shows an animation software interface. At the top is a timeline with markers from 5 to 70. Below it is a workspace area containing a video clip of a person with a yellow balloon. To the right is a properties panel for the clip, named 'puzzletovideoclip'. The panel shows dimensions (W: 220.4, H: 140.3) and registration point coordinates (X: 0.0, Y: 0.0). Below these are rotation and scale settings. A list of 16 items is shown, including 'base' (Movie), 'Bitmap 2', 'Bitmap 3', 'bubble 2 blue glow' (Graphic), and 'bubble 2 green' (Button).

You must bring each piece to the Stage and give it an instance name. These names can be whatever you want, but I use the names piece1, piece2 etc to keep things easy. Remember the name of the last piece you bring to the Stage and named. In the case of my example puzzle, this was piece6.

Go to Window/Components and bring two buttons to the Stage. I made them different colors and edited the corresponding symbol in the Library to have the new label. Give each button an instance name: buildbtn and mixedbtn are fine.

Use the text tool to make a static text field on Stage. Put in the directions regarding the mouse and keys on how to move the pieces.

Go to Window/Components, and drag FLVPlayback to the Library. The code, shown below in Piece.as, creates a new FLVPlayback object, but the Component must be in the Library.

To sum up, the Stage holds 2 buttons, all the pieces (make sure they have instance names!!), and a static text box. The pieces can be mixed up on the Stage. In addition, the Library holds the FLVPlayback symbol as well as the symbols for the instances on your stage.

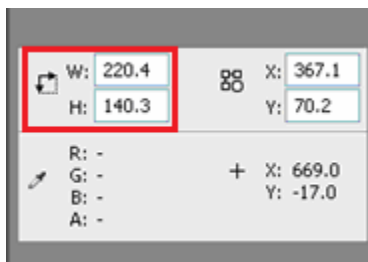
CODE

Making a new puzzle is all done in the .fla file. This code is to be written into the actions section of the first frame on your timeline. This is very similar to the last puzzle. The code for the puzzle is as follows:

<code>import jigsaw.*;</code>	get the Piece.as definition
	For each piece, set up the info.
<code>var p1:Piece = new Piece(96.4,124.5,piece1);</code>	(be sure to enter the coordinates of your pieces)
<code>var p2:Piece = new Piece(196.8,111.0,piece2);</code>	
<code>var p3:Piece = new Piece(367.1,70.2,piece3);</code>	
<code>var p4:Piece = new Piece(101.9,277.4,piece4);</code>	
<code>var p5:Piece = new Piece(268.4,248.3,piece5);</code>	
<code>var p6:Piece = new Piece(379.4,237.3,piece6);</code>	
<code>mixupbtn.addEventListener(MouseEvent.CLICK,</code>	set up event handling: it is a call to the class method mixup
<code>function(ev) {Piece.mixup();}</code>	
<code>);</code>	
<code>buildbtn.addEventListener(MouseEvent.CLICK,</code>	set up event handling: it is a call to the class method buildit
<code>function(ev) {Piece.buildit();}</code>	

);	
Piece.setup(piece6, "boyscoutsandtroll.flv",477.3,361.4,379.4,237.3);	Invoke the class method setup to pass information **make sure that the filename you put in the quotes is the title of your flv video

The 3rd and 4th parameters in the call to setup are calculated from measuring the width and height of the base image. This information is available in the info panel (where you got the X and Y coordinates) the location of these numbers is shown in the red box:



The 5th and 6th parameters are the offsets (X and Y coordinates) for the last piece (piece6 in this case). It needs to be the last piece brought to the stage so that the video is on top of all the pieces.

The Piece.as file, to be described next is independent of any specifications and can be used for any puzzle to video game you create. You create the Piece.as file in Flash by clicking on File/New and then select ActionScript File.

The Piece.as code in outline is the following:

import statements	
class variables	indicated by <i>static</i> , just one each for the whole program. Includes the pieces array
object variables	one for each Piece
Piece	the constructor method, called for each piece by the fla file
startdragging	event handler for mouse down (for each piece)
reportKeyDown	event handler for key down for each piece. Only does something for left and right arrows
stopdragging	event handler for mouse up for each piece. Calls checkit. If it returns true, plays video, using seek to start at start.

setup	called by fla file to set values for video. Also calls mixup.
mixup	event handler for mix up button and also called by setup
buildit	event handler for build button
checkit	called by stopdragging to check if puzzle is complete

Now alter your code according to your handout.

package header
needed to add to display list
needed for Mouse events
needed for key down event
needed for video playback
class header
just one of these: holds all the pieces. Needed to do checking and building
just one: fixed amount of rotation
just one: used by mixup
just one: determines if something is being dragged
just one: holds video
for building and checking
for building and checking
points to movie clip instance on the Stage
constructor method
store away
store away
store away
add to the array
set up event for starting dragging
set up event for stop dragging
set up event for key down
makes piece that is the movie clip accept the events
close method
method header for handler for MouseDown
starts dragging
sets variable to be used by reportKeyDown
close method
method header for key down

If any dragging going on
if the right arrow pressed
rotate this piece the set amount
end if
if the left arrow pressed
rotate this piece negative direction the set amount
end if
end if anything being dragged
end method
method header for mouse up event handler
set dragging to false
step dragging
do the check for the puzzle being complete. If it is...
... return video to its start
.... start playing video
... make video visible
end if puzzle complete
end method
header for method that handles build button clicked
used for iteration over pieces
used to hold each piece
used to offset puzzle from corner in x
... and y
make sure video is not visible
stop video (no problem if it wasn't playing)
for each piece
set np
position in x
position in y
set rotation to zero
end loop
end method
header for method that does setup. Called from fla file
make sure video doesn't start playing when source set
set the source
set the size
make visible by adding to Display list using the ahook value
position using values sent in
position using values sent in
Invoke mixup
end method

method header for handler for mixup button
used for iteration
used to be each piece
make sure video is not visible
make sure video is stopped
iterate over each piece
set np
position at random x
position at random y
rotate to random angle
close loop
end method
method header for checking. Called by stopdragging
set tolerance
for iteration
set to be each piece
the first (0 th) piece is what all are compared to
set 0 th piece rotation
Loop
set each piece
see if it is different from the first piece
... if so return false
end if
end loop
used for calculations
set to be used if there needs to be an adjustment
Is the first piece rotated?
convert to radians
calculate cosine
calculate sind
set so adjustment is done
close if
Now, iterate over all pieces
set np
set horizontal (x) distance from this piece to the 0 th piece
set vertical (y) distance from this piece to the 0 th piece
Does it need to be adjusted for the rotation
calculate new x
calculate new y

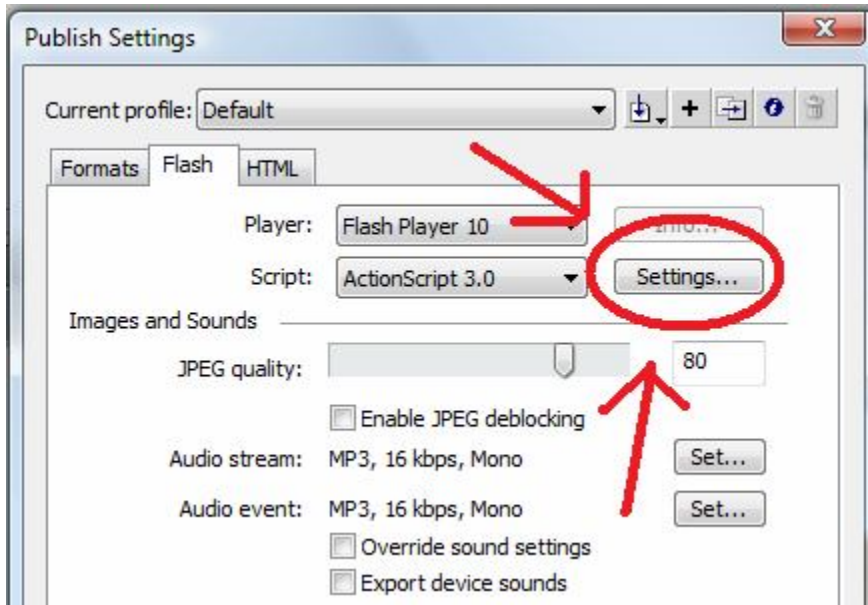
end adjustment
calculate current horizontal (x) distance from 0 th piece
calculate current vertical (y) distance from 0 th piece
calculate and check distance
if too much, return false
end if
end loop
If this point is reached, then all checks passed, so return true
close method
only one: method header for function to calculate distance between vectors
calculates distance
return value
close method
close class
close package

Before you publish the file, the .as file and the .fla file must be lined up accordingly. On a MAC, the .as file and .fla file can occupy the same folder but for some reason relating to how packages are set up, the .as file on a PC must be done in a particular way. In order for the .fla file to properly “find” the .as file on PCs, it needs to be in a folder at the top of the C: drive. Create a folder on the C: drive and name it something, I called mine ‘as3.’ In this folder, you will put a folder titled ‘jigsaw’ in which your Piece.as file will reside.

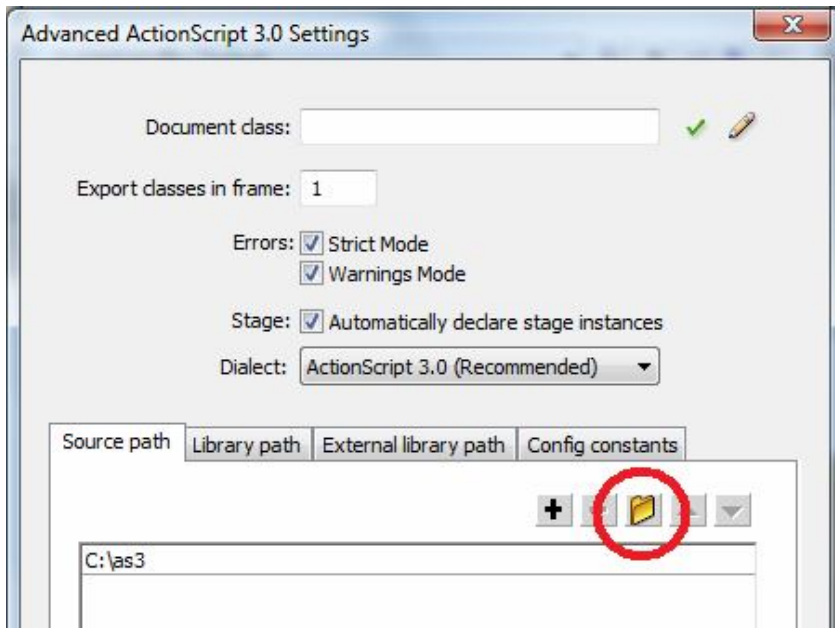
FINDING THE .AS FILE

The following series of pictures depicts how to do this on a **PC**:

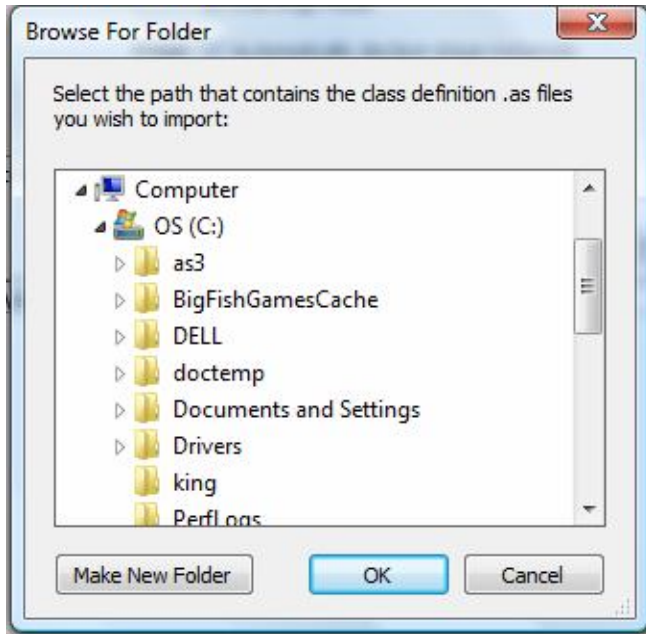
Go to File/Publish Settings and Click the settings button next to ActionScript 3.0:



Next, on the “Source Path” tab, click the little folder to browse directories:



From here you can navigate to the folder you set up on your C: drive:



Test the program in the usual way. The .flv file must be in the same folder as the .fla file. To be more formal, testing produces a .swf file in the same folder as the .fla file. This is the program that runs and it 'looks' for the .flv file in the same folder.

A quick way to check if the game works is to click on the Build button, drag a piece away. Then drag the piece back. Be patient, you'll need to give the video time to load.

Publish and Upload

When the application is finished and working, produce the finished version by clicking on File/Publish. This produces a .html file and a .swf file. It is up to you to make sure that the saved .fla file and the .as file correspond with what you Publish. You will need to upload the .html and the .swf and the .flv (the video file) to the server. You do not need to upload the .as or the .fla file or the image file, if it exists as a distinct file.