Birdopolis - ActionScript 3

Working with X.as, Birdopolis.fla., Birdopolis.swf

See next section if you are working with the art or UI.

1. Lock the .as file via SVN before opening it to edit it. Open the Birdopolis.fla file, too, as you will be testing game changes with that file, and then creating the Birdopolis.swf from it.
2. Edit the .as file.
3. When editing is done, click the Birdopolis.fla tab to give it the focus. Test the change/addition - Ctrl-Enter is preferred; using the Flash Pro UI to run Debug is not. Birdopolis.fla creates Birdopolis.swf.
4. When testing is done, commit to SVN the .as file(s) and Birdopolis.swf.
5. Test again, this time on dev rather than via the Flash IDE.

Birdopolis.fla doesn't get checked in, unless you do something that affects the .fla itself, like add music that the .fla handles or change the .fla's music settings.

Art Flow into AS3/Game Client

1. Artists create the art and place it a .fla file in **\holland\trunk\HollandProject\as3\External Art Content Sources**. GUIv2.fla contains GUI elements and is the file you will usually use.
2. Open the file in the Flash authoring tool. Open the Birdopolis.fla file, too, as you will be saving/test game changes with that file, too.
3. In the file containing the element that you want to work with, click the Timeline tab and find the element you want. It is best to work via the Timeline and not the Library, as the latter involves more steps that are not documented here.
4. Use the visibility control to show the element you want and hide all other elements. When you finally export to swf, only the visible element will be exported!
5. Highlight the element's name on the left of the Timeline to show it in the Scene window.
6. Double click on the element in the Scene window to open the element's own Timeline. This Timeline should have several layers, such as for ActionScript code, keyframes, animation code, and frame labels.
7. Typically, you will want to edit the ActionScript layer. Highlight the ActionScript layer, double click the "a" frame on the layer's Timeline, and look in the "Actions - Frame" window or tab for the actual ActionScript. If you're adding a button to an existing dialog, for example, you will need to set up things so that the game's ActionScript class(es) can find and control the button in the swf. This will likely mean setting things up to set a name for the button (passed in via the main ActionScript) and whatever else you need, like a click listener for the button.
8. When done, check the Library panel, find the overall element name there (e.g., the dialog you added the button to) and make sure it has a linkage name with the same name as the item name. This allows the main ActionScript to find the element. Also, right click the element in the Library and makes sure "Export for ActionScript" **is checked** and "Export in Frame 1" is **not** checked.
9. Now use **export movie** to export the swf to the project's swf folder (which will be in a different place than the .fla folder). Don't use publish. File -> Export -> Export movie. Make sure you give it the correct swf name! When you export from GUIv2.fla, you will only be exporting one gui element to a swf. Example: You changed the Bird Collections dialog (bird collections layer in GUIv2.fla). You export to move with all layers not visible except bird collections. You need to export it to guiBirdCollection.swf.
10. Click the Birdopolis.fla tab to give it the focus. Find the ActionScript class that handles the element. For example, AddEnergy.as handle the out-of-energy dialog. (Note that all graphics requests also involve Main.as.) Modify the controlling class as needed.
    - For a button, for example, you will need a requestGraphics(graphicsID, graphicsHandler, data, null) call. The null parameter is for a loader you don't have to worry about. The data can null or a string containing specific instructions for the condition or state of the graphics (e.g., which slot in the bird mall a graphic is being loaded in). Look at the code for examples. The graphicsID is the name of the swf file (without the .swf). The graphicsHandler is a function that tells the code when the graphics is done loading. requestGraphics() only requests the graphics; it does not control it.
    - For a button, you will also need a graphicsLoadedHandler(id, mc, data) which actually controls the graphic. id is the name of the swf file (without the .swf) and mc is the linkage name (which if everything is done right will be the same name as for id). data is the data per above.
11. You then can do things with the graphics, like myGraphics.<graphicsName> = <someCall> or myGraphics.<graphicsName>.<someProperty> = <whatever>, or myGraphics.<graphicsName>.<someting>, etc. myGraphics.<graphicsName> = <someCall> example: to use this, define a public function <someCall> that myGraphics.<graphicsName> = <someCall> will use.
12. Save files as needed. Click the Birdopolis.fla tab and test the change/addition.

> **SVN Note*

> *The .fla, .swf, and .as files are versioned using SVN. So, these files will need to locked before editing and then committed to SVN when local testing is done to get them onto dev.*

Birdopolis Display Organization

- GUI containers
    - positionGuiContainters() handles screen resizing.

For a handy chart, see https://picasaweb.google.com/107468940281608939345/DocNuuk?authkey=Gv1sRgCLKNkpuh0-KvtQE#5738378353865719090
(Security Note: Only people who know the link can access the chart.)

Notes

- Main class. Birdopolis.fla defines this as the main class, so Birdopolis cedes control to Main as it loads.
- LevelModel class. Contains lookup tables of elements in the level. Defines the model used by the level.
    - constructIsometric() displays the model.

    - deconstructIsometric() removes an object from the display tree but does not destroy the object.
- Resource scene container. Hold resources and purchase-expansion buttons.