The Beauty and Joy of Computing[1]

Lab Exercise 8: Send in the clones

Note: This lab is not fully developed. The activities lead you through some exercises that demonstrate some powerful BYOB features, but are not explained as thoroughly as desired. Unfortunately, this is as complete as the writeup can be made prior to the lab - complete what you can, think about the concepts involved, and we'll talk more about these concepts in class.

Objectives

By completing this lab exercise, you should learn to

- Use the sprite cloning functionality of BYOB to create many sprites for complex animations;
- Access variables and scripts in other sprites; and
- Understand the basics of object-oriented programming, with object attributes and object scripts.

Background (Pre-Lab Reading)

You should read this section before coming to the lab. It describes how various things work in BYOB and provides pictures to show what they look like in BYOB. You only need to read this material to get familiar with it. There is no need to actually do the actions described in this section, but you certainly can do them if it would help you follow along with the examples.

In the previous labs, we have worked with a small number of sprites, and every sprite was specifically created and programmed through scripts that operated on that sprite. Scripts can create interesting and complex actions for individual sprites, but we often want to have many sprites. Think about an interesting game: there are often tens or hundreds or even thousands of characters that you must keep track of. In addition to characters, we might also have other game items such as weapons, clothing, buildings, that we also need to track. We obviously don't want to create a specific sprite for a game in which there are thousands of characters, and what saves us is the fact that many items share a similar structure and set of actions that they can perform. A style of program design that handles this very nicely is called **object-oriented programming**. While a full explanation of object-oriented design is beyond what we will do in this class, the basics are easy to understand.

We use the term object to refer to some program entity (in our examples, this could be a player, building, weapon, etc.) that has values or attributes associated with it and can perform certain actions (which are typically called **methods**). For example a player object might have attributes for the player's location, health, inventory, etc. A building might have a location, size, structure, construction material, etc. Notice that these attributes have generic names, and while all objects of a certain type (or class) might have the same set of attributes, it is the values of those attributes that makes each object distinct. BYOB supports a particular style of object-oriented

[1] *Lab Exercises for "The Beauty and Joy of Computing" [This is a preliminary version for the Fall 2012 class]*

programming in which new instances of objects are created by cloning an existing object, which carries over its full set of attributes and actions (scripts for that object), and then the program can adjust these attributes to make each new object distinct.

Activities (In-Lab Work)

In this lab we are going to create a simple memory game. You've probably played this kind of game before - for example, some people play with half a deck of cards, shuffle them up and deal out the cards face down. Players then take turns turning over pairs of cards - if you match (two queens, for instance) you take the pair and go again. In our game, we will have a bunch of Alonzos, but each one has a secret identity! There are two Alonzo's that are really dragons, two that are the the little blue character, etc. You will uncover secret identities by clicking on them.

Here's the basic idea: We'll start with a single Alonzo that can run scripts, but he will be hidden and invisible to the user. The only thing this Alonzo will do is create clones of himself, so this Alonzo is called the **parent sprite**. The parent Alonzo will have a total of 9 costumes, including the basic Alonzo costume and eight secret identity costumes. The basic outline of our game program is then as follows: The parent Alonzo will create two rows of Alonzo's, each row with eight Alonzos and a random assignment of secret identities. Since each of the new Alonzo's is a clone of the parent, each one will have the 9 costumes of the original Alonzo. How do we keep track of the secret identity for each Alonzo? That's precisely what we were describing above as an attribute, which is a value that can be set and tested or read separately for each clone. We will see how precisely how this is done in Activity 1 below.

Warning!!! In this lab you will be creating sprites and deleting sprites dynamically. It is easy (way too easy!) to accidentally delete all your sprites and the scripts that go along with them - this is super frustrating when it happens (I speak from experience!), and you have to start over if you haven't saved your work. Because of this, it is extremely important that you save regularly. Just get in the habit of clicking the save icon every time you modify scripts so that you can recover if they end up doing something disasterous.

Activity 1: In this activity, you'll see how to clone Alonzo and place him in a particular location and set his secret identity. To start off, open BYOB with a new project, which should include an Alonzo sprite. An attribute is just a variable that is "For this sprite only," so the first thing to do is go to the variables category and click "Make a variable." Use "secret" for the name, and check "For this sprite only." We only briefly explored this kind of variable before, but knowing about the concept of cloning should allow you to make a little more sense out of this now. When you clone this sprite, it will also have a variable named "secret" with the same value as the parent Alonzo's "secret" variable, but after this point these are two separate and independent variables. Changing the "secret" variable in the parent sprite has no effect on the child sprite, and vice-versa. This means that if we have eight (or eight hundred) child sprites, each one will have its own secret. While we're setting up this first Alonzo, drag out the "set size" block and set the size to 40%, click on it, and you'll have an appropriately-sized Alonzo for this exercise. Finally, you should add 8 "secret identity" costumes to the Alonzo sprite, using the "Import" button in the Costumes pane. You can pick whatever costumes you want, as long as they are roughly the same size as Alonzo. Here are the costumes I used:

We are going to create two blocks for Alonzo, one of which will run in the parent sprite and create the clone, and the other of which will run in the newly created child sprite to initialize the new clone's properties (location, visibility, and secret identity). Let's start with the latter of those scripts, which initializes attributes of a newly-created sprite. In most object-oriented languages, this type of script is called a **constructor**, because it is used to construct a new object. We need to set the location for this sprite, set the secret identity, and since the parent sprite will be hidden we need to make sure this clone is visible. The only slightly tricky in this is the way we set the position. We'll use two parameters to denote a row and column number with row values from 1 to 2 and column values from 1 to 8. To set the actual x and y placement we'll use some formulas: we'll allocate a 60x60 block for each Alonzo, shifted by an amount so that we can fit two rows of eight that are roughly centered on the stage. Putting all these pieces together we get the following constructor, which you should build:

Next, we'll make the script that actually creates the clone. This script needs to do three things: actually create the clone, add this clone to a list of Alonzo's children so that we can keep track of them, and then have the clone execute its constructor. To create the clone there is a block in the "Operators" category called "clone" - note that this is actually a reporter block, because it provides a reference to the object (the child sprite) that is created. We'll keep these references on a list - so you should create a global list named "child alonzos", and every time we create a child we'll add it to the list. This makes it easier to "clean up" and delete all the child Alonzos when we are finished.

Finally, how do we execute the constructor that we made? This is a little tricky, but we'll go through the reasoning, and then show you the code. We want to run the constructor, which is the "place at..." block, but we want to run this *in the child sprite* - the one we just created. To refer to either an attribute or a script that is in another sprite, we use the block that looks like this (it's in the sensing category):

The first parameter gives what we want to access, and the second one specifies the object that we want to use. In this case we want to run the "place at..." script in the child that we just created - we use the special "the script" block in "Operators" to specify the script we want to run, so if we have a reference to the child object in a variable named "sClone" then we use the following construction to refer to this script in the child object:

Now that we can refer to the script, we need to say what we want to do with this script, and the options are “launch” or “run”, which are both in the “Control” category. The difference between these two options is whether we run the script and wait for it to complete (this is “run”), or whether we start the script in its own separate thread and let it run on its own, without waiting for it to finish (this is “launch”). The “launch” and “run” blocks allow you to specify inputs - just click on the arrow on the right of the “launch” block.

Let’s put all these pieces together: we want to clone the object and save a reference to that new object, we’ll add that reference to a list of Alonzo’s children, and finally launch the constructor in this child object to initialize it. Here’s the final block definition that you should construct:

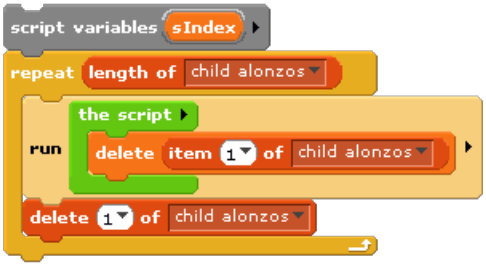
After you build this block, test it out to see if it works correctly. For example, drag out the “make clone” block, fill in the parameter and execute it, like this:

Click on this, and an Alonzo should appear on the screen, and an additional Alonzo sprite will appear in the sprites pane. Select this new sprite in the sprites pane, and check the “secret” variable in the variables category (you can either check “secret” so that it is a watch variable, or just click on the variable in the blocks palette to make it report its value). Does the secret have the value 2? (It should!) Next, select the original Alonzo sprite in the sprites pane, and try “make clone” again, but with different arguments (say x=2, y=2, and secret=4). You should see another Alonzo, at another position, and if you select that Alonzo in the sprites pane you should see the different secret value (4 in this case).

When you have completed this activity, and have experimented with this enough to understand how things work, save it as “Lab8-Activity1”.

Activity 2: If you tested the “make clone” block from Activity 1 a few times, you should have several Alonzo sprites now. We need a way to “clean up” all these extra sprites, to make it easier to run several tests. Fortunately, we saved references to all these new sprites in the “child alonzos” list, so we’ll just go through that list, deleting all these cloned Alonzos. The easiest way to do this is to always work with the sprite at the front of the “child alonzos” list - we’ll delete the sprite using the reference at the front of the list, and then we’ll delete the first item in the list since we don’t need that reference any more. We repeat this for the entire list, giving the following script, which can be build in the regular scripts pane for the parent Alonzo. At the end, the child sprites will all be deleted, and the “child alonzos” list will be empty.
Important: *make sure you’ve got the original Alonzo selected in the sprites pane before making*

this script! If you make this in a child Alonzo, then this child will be deleted by its own script, and then you will lose all of the work you just did! So be careful to create scripts only in the "parent Alonzo", and save often! Here's the script you should build:



Once it is build, run it. Did all the child Alonzo's get deleted? Hopefully the children were deleted, and the original is still there. When you have this script build and tested, save you work as Lab8-Activity2.

Activity 3: We want to set up two lines of Alonzo clones, with a random ordering of secret identities in each row. What we want to do is create a list of secret identities (numbers 1 through 8) in random order. This is actually a good use of the "insert" block with the "Any" argument, which puts a value in a random position. To create a randomly ordered list, we simply insert values at "Any" position, and if we use the index iterator pattern to count through values to insert. We will create a reporter block that builds a list and reports it, so that we can save the list in a variable and use it for secret identities. Here's the definition:

So we can call this block, and iterate over the randomly-ordered identities, using those when we create a line of Alonzos. Examine the following script and understand what it does:

Notice the two loops - the outer loop iterates twice, once for each row of Alonzo's that we create. Within this first repeat loop, we create a random ordering of secret identities, and then create 8 clones of Alonzo with the secret identity taken from the random permutation (note that we add one, since the first costume is the basic Alonzo costume, not a secret identity). Build this script and test it out. When you've got it working correctly, save this as Lab8-Activity3.

Activity 4: Now you're on your own! This activity is entirely extra credit, but it will be good to see how far you can get in completing the memory game. The main task is to handle what happens when the player clicks on an Alonzo clone. We can uncover secret identities with a script that looks like this:

Of course, that's *not* what we want to do. We want to handle clicks in pairs, and hide or remove sprites when there is a match. This can be done with some fairly short scripts, but they're a little tricky. Experiment and see what you can make work! Save your final version as Lab8-Activity4.

Discussion (Post-Lab Follow-up)

No post-lab reading this time...

Terminology

The following new words and phrases were used in this lab:

- *attributes*: variables that are associated with a specific object or sprite
- *child sprite*: a sprite that is created by cloning another sprite - in cloning, the original sprite is known as the "parent sprite" and the new sprite is a "child sprite"
- *constructor*: a script that is run to initialize various attributes in a new sprite
- *methods*: scripts that are specific to a particular object or sprite, performing actions

- *object-oriented programming*: a style of program design that is oriented around defining programmatic objects in terms of the attributes and methods (note: object-oriented programming has many other aspects that are beyond the scope of this lab!)
- *parent sprite*: a sprite that creates another sprite by cloning - in cloning, the original sprite is known as the “parent sprite” and the new sprite is a “child sprite”

Submission

In this lab, you should have saved the following files: Lab8-Activity1, Lab8-Activity2, Lab8-Activity3, and Lab8-Activity4. Turn these in using whatever submission mechanism your school has set up for you.