1 Administrative Topics

- We take the quiz

2 Proj 6

Recall that a top-down design breaks down each task into smaller tasks. Fundamental to our project is breaking down complex objects into simpler component objects. Our tasks are broken down in a manner that is convenient for our object break-down (e.g. we break down the task of drawing a scene into the tasks of drawing each aggregate object, which are then broken down into the tasks of drawing each primitive object.

We have three levels in our hierarchy of drawable objects:

- primitives: Zelle’s graphics (shapes) objects. Each of these shapes has a unique creation function (the name of the shape type plus appropriate input) and some unique methods (mostly methods that allow you to learn about where the object is. All of these shapes share common draw, undraw, clone, and move methods.

- aggregates: lists of primitives. Each of these shapes has a unique create function (shapenameInit) and animation function (shapenameAnimate). All of these shapes share common draw, undraw, and move methods (in aggregate.py).
• scenes: lists of duples, where each duple contains a string describing the aggregate shape and an aggregate shape). Each scene has a unique creation function (scenenameCreate), draw function (scenenameDraw), and animate function (scenenameAnimate). Note that scene2.py (the file you are supposed to create in task 3 of Assignment 6), is not really a scene object – it is just a main program that draws two scenes. So it does not fit this description.

The design of this project is hierarchical (top-down). The main program is broken into 5 steps. Steps 2 through 4 involve hierarchies of function calls, reflecting the top-down design.

1. Create a window.

2. Create the objects. To do this we create one or more scenes. The functions that create the scenes call multiple functions that create aggregate objects. Each of those functions, in turns, calls functions that create multiple primitives. (Nice hierarchy).

3. Draw the objects. To do this we call the scene draw function, which calls the aggregate draw function on multiple aggregates, which calls the draw method on multiple primitives. Again, nice hierarchy.

4. Animate the objects. To do this we call the scene animate function in a loop. Each time the scene animate function is called, the appropriate aggregate animate functions are called. Those functions manipulate the primitives appropriately.

5. Close the window.

General Advice

• Don’t try to shrink or rotate objects. It is quite quite difficult.

• In the 11 class, we tried drawing and undrawing lightning bolts and it was a little complicated. Also, I think anything that is drawn during an animation call will be drawn on top. You probably don’t want this to happen if you animating a background.
• An easier approach to adding lightning bolts is to make them “disappear” by changing their color to match that of the background.

• You may add parameters to any of the functions, if you would like to make them more flexible (e.g. adding color parameters to fisher_price_personInit) or if they simply need more information (e.g. whether to shake or fall in fisher_price_personAnimate).

I have posted code for my aggregate objects on the web.