Kristina Frye, Konstantin Macarenco, Tyler Poole

CS 420/520

June 8, 2015

Final Project

1 Introduction

The CreateGraphics is a graphics package for Grace designed as a replacement for the existing ObjectDraw. It is based upon the Open Source Createjs Javascript project that can be found at `http://www.createjs.com/`. Createjs is a rich graphics library with a lot of functionality. This project only implements its most basic functionality and can be considered a proof of concept more than anything. However, we are confident that the framework that we have created could be easily expanded to include more of the Createjs functionality.

2 Architecture

There are two major pieces to the CreateGraphics library: createJsGraphicsWrapper and createGraphics.

createJsGraphicsWrapper provides short methods that wrap corresponding Createjs Javascript methods and objects into Grace objects. The objects in this file create an interface for the Javascript methods with as much one-to-one correspondence between Grace and Javascript as possible in order to maintain as much functionality and flexibility as possible.

createGraphics is a Grace-only file consisting of objects and methods that call the createJsGraphicsWrapper. The idea was the build a nice Grace interface that will work with the more technical wrapper functions.

See Figure 1 for a depiction of the different layers involved in the CreateGraphics library. The "user program" at the top refers to the code that the programmer will create in order to place graphics into the graphics window.

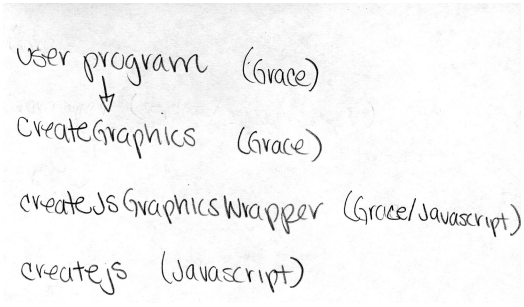

Figure 1: createGraphics layers

2.1 createJsGraphicsWrapper

The createJsGraphicsWrapper file (see Figure 2) consists of a number of related objects that contain javascript native code that calls the Createjs javascript library.

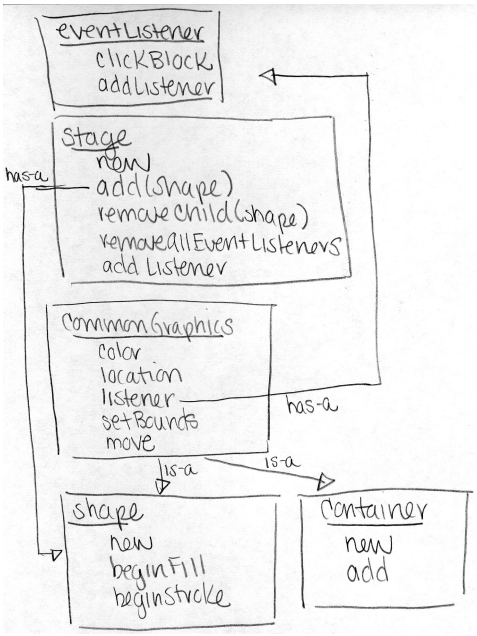

Figure 2: createJsGraphicsWrapper

Stage is the Createjs version of the HTML canvas with additional functionality. It holds all the the shape objects.

CommonGraphics is a base object that contains the functionality common to both the shape and container objects such as location, bounds, and moving.

Shape is the base shape object. It inherits CommonGraphics. It contains functionality that is common to all shape objects.

Container is a holding object that can be used to place multiple shape objects together. For instance, you could use a container to create a button holding rectangle and text objects.

The individual shape objects are also defined in this file including circle, rect, roundRect, ellipse, polyStar, line, and customObject. Each shape has its own draw routine.

2.2 createGraphics

createGraphics.grace (see Figure 4) consists of a nice interface for programmers to use in order to create graphics.

The parent object "shape" for all concrete shapes (circle, rectangle, ellipse, text, etc) is defined here as well as the main graphics object that is used to create the window, the canvas, and hold all reference to the created shapes.

In order to make the shapes easy to configure, most configuration variables (such as location, color, size, etc) are public. This allows the user to set them with easy calls such as color := "red" and location := 20@20. "Chaining" methods have also been defined in order to combine configurations onto one line such as: circle.colored("red").at(20@20).draw.

Click handlers are added by simple defining a click variable with a block such as that shown in Figure 3.

```
circle.click := {
  print("clicked circle")
  circle.color := "blue"
  circle.draw
}
```

Figure 3: Click handler

3 Challenges

The biggest challenge of this project was to get the interface between the javascript Createjs library and the Grace compiler working. In order to get the graphics working, we examined the existing ObjectDraw object to see how it worked, figured out how the calls would work in straight Javascript, and then used the Javascript console in the Chrome browser to step through the code and figure everything worked together. One of the complications is that graphics are usually drawn in the main browser window. However, the goal in this case was to open a secondary "pop-up" window and draw the graphics in that window. Since this is not the usual way that Javascript is typically used, we ended up having to use a few workarounds.

One of these workarounds involves the event listener for the click handler. The Createjs library normally allows the developer to assign click handlers to the individual shape objects that are created. So, if you create a circle, you can assign a click handler directly to the circle. This handler only is activated when the circle is clicked. However, because we are using the pop-up window, this turned out not to be possible. This appears to either be a bug or a missing feature in Createjs, but the event listeners on individual shapes are not working when used with the pop-up window. However, the event handler for the "stage," which is the Createjs specialized version of the HTML5 canvas, does work. So we assign all event handlers to the stage and check the mouse position in order to determine which object was clicked. This is not as clean or efficient as adding the event handler directly to the shape, but it works. Another possibility in the future would be to designate part of the Grace compiler webpage for the graphics display. This would get rid of the "pop-up" issue and would allow a much greater variety in mouse handling.

Another challenge was adding sound capability. We were hoping that we could load the sound files when the graphics library was loaded. That way the user who is not working with the graphics library would not have to load the sound fields into the browser every time the Grace compiler was used. However, this didn't turn out to work very well. When loaded with the graphics library, the sound would play the first time the program loaded. However, if we tried to run the program again, the sounds would not play unless we reloaded the entire Grace browser webpage. Therefore we added the sound load into the main Grace index.html file, choosing only a small sampling of sounds with a relatively small file size.

4 Achievements

We are proud of the resulting user interface of the createGraphics library. Although it doesn't yet have a large number of features, we have provided a good framework that could easily be built upon, and the existing user interface should be easy for the beginning programmer to work with. We have made it as "Gracelike" as possible and provided default settings whenever they were needed in order to make it quick and simple to use. We think it's going to be a good improvement on the existing graphics library once some additional functionality is added.

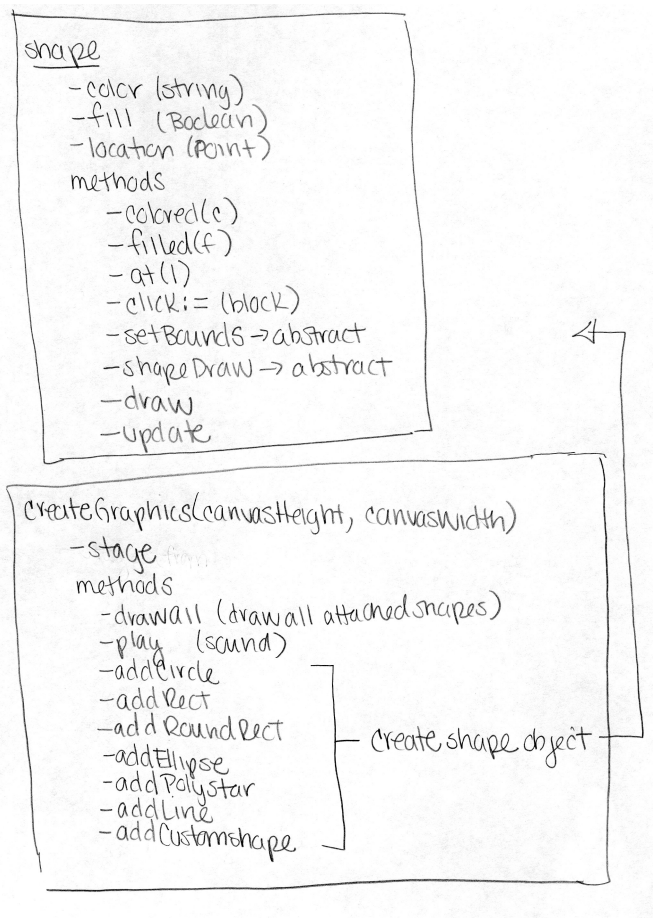

Figure 4: createGraphics