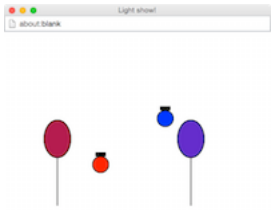
CS 051G Homework Laboratory # 3
You light up my life!

Objective: To gain experience implementing classes and methods.
Note that you must bring a program design to lab this week!

—

1 The Scenario.

For this lab, we would like you to write a program that simulates the way two lights would illuminate objects. In our case the objects to be illuminated will be balloons (to make you feel cooler!). The lights will be equipped with handles, and they can be dragged around the screen by grabbing the handles, while the balloons are just rooted in place. Of course you can set the light to whatever color you like, and the color of the balloons will depend on the distance of each light to the object. See the picture below to get a sense as to what the screen should look like when the program is executing.



As usual, copy the folder Lab3−Lights from the folder cs051G/labs. Copy this into your CSC51GWorkspace folder.

Open Chrome to http://www.cs.pomona.edu/~kim/minigrace. This week you will need to add the file lightStarter .grace from the Lab3-Lights folder you copied. Once it is loaded, click on the "Run" button. It will just pop up a blank window. As usual, you will edit the program in the browser window. Be sure to work in very small pieces so that you test what you have written very often.

2 Bring up the lights!

We hate to admit this to you, but people putting together video games often cheat and don't make the images and actions exactly true to life. Because most things happen quickly, and the player is paying attention to changes on the screen rather than measuring the conformance to the laws of physics, programmers just try to approximate reality. We will do the same with our lighting.

In theory, every pixel on the balloon should have a slightly different color because each is a different distance from the two lights. However, we will just determine a single color for the entire object. The second way in which we will cheat is that rather than having the illumination following an inverse-square rule (don't worry if you don't know what that means!), we will instead have the brightness decrease linearly with the distance.

To give you a step up on this project, we will even give you a method that will calculate the color that should appear on the object due to the two lights. The header of this method is:

method getColorFrom (first: Light, second: Light) to (pt: Point) −> Color
It takes two lights that you will place on the canvas and a point on the balloon, and returns the color that you should paint the entire balloon. You don't need to understand the details of how it works, but we will explain it in an appendix for those interested.

3 Design of the program.

We will help you design this program by identifying the classes and methods needed. In particular, you will need two classes named light, which generates objects of type Light, and balloon, which generates objects of type Balloon, as well as an object lightGame that inherits graphicApplication that sets everything up and responds to mouse actions.

3.1 Lighting the way

The light class creates objects of type Light:

```
type Light = {
    // move light by (dx, dy)
    moveBy(dx: Number, dy: Number) -> Done

    // does the light contain locn?
    contains(locn: Point) -> Boolean

    // the center of the light
    center -> Point

    // the color of the light
    lightColor -> Color

    // move the light to the top layer of the canvas to be above other objects
    sendToFront -> Done
}
```

As you can see from the image above, lights consist of a framed and a filled oval as well as a filled rectangle for the handle. Only the filled oval is the color of the light. The idea is that the light can only be dragged around by its handle (though the dragging is done in the object lightGame).

Methods moveBy and contains act like those for objectdraw graphic objects. Method moveBy moves the light (including its handle) by the given amount, while contains returns true iff locn is in the handle of the light.

The method center returns the location of the center of the ovals of the light. (Note that the value returned will change as the light is moved.) The method lightColor returns the color of the light, while sendToFront sends all the components of the light above all the other objects on the canvas.

3.2 The balloon: Rising to the challenge

The balloon class creates objects of type Balloon:

```
type Balloon = {
    // adjust the color of the balloon to reflect locations of the lights
    adjustColor -> Done
}
```

Balloons are much simpler than lights. Objects created by the balloon class must create the image of a balloon – something simple like our picture above is fine – with two filled ovals and two framed ovals on top of them to create the boundaries). It also needs to be able to change its color when requested, with the new color calculated using the locations of the lights.

As mentioned earlier, we provided you with a method getColorFrom that calculates the appropriate color based on the locations of the lights and the balloon's center. Thus your adjustColor method must obtain that color and then paint the filled ovals composing the balloon.

4 Putting together the pieces

4.1 Part 1

For the first part of this program, you should just worry about creating the lights and moving them around. We'll worry about their interactions with balloons later. Just like last week, we want you to come to lab with a written design. At the end of this section, we will be more specific about what you should bring with you to lab. To give you a better sense of what we mean by a written design, you can see a sample design for the first part of the laundry lab at http://www.cs.pomona.edu/classes/cs051G/labs/lights/LaundrySorterAppletDesignPart1Soln.grace.

We have provided you with the skeleton of a light class, copied below:

```
// Create light centered at pt with color color' on canvas
class light.at (pt: Point) color (color': Color) on (canvas: DrawingCanvas) -> Light {
    // radius of the light
    def radius: Number = 20

    // diameter of the light
    def diameter: Number = 2 * radius

    def handleHeight: Number = 5
    def handleWidth: Number = 15

    // more defs and/or vars as needed

    // move light by (dx, dy)
    method moveBy (dx: Number, dy: Number) -> Done {

    }

    // does the light contain locn
    // THIS CODE IS WRONG, FIX IT!!
    method contains (locn: Point) -> Boolean {
        true
    }

    // the center of the light
    // THIS CODE IS WRONG, FIX IT!
    method center -> Point {
        0 @ 0
    }

    // the color of the light
```

```
    // THIS CODE IS WRONG, FIX IT!
    method lightColor -> Color {
        red
    }

    // move the light to the top layer of the canvas to be above other objects
    method sendToFront -> Done {

    }
}
```

We have given you the header of the class, a few useful **def**s to help you with the size of components of the light, and the method headers. Methods that return values are provided with a default value to return. These are wrong – just included so your program will compile correctly. Where it says "THIS CODE IS WRONG, FIX IT!", you must indeed erase that code and insert the correct code instead.

At this point, we won't really be using the center and lightColor methods (though they should be easy to write). You may postpone them if you like. However you should fill in all the other details of the class and the other methods.

Begin by writing code to create the two circles (ovals) and the rectangular handle. The parameter pt represents the center of the circles. You will need to calculate the upper left hand corner of the smallest rectangle containing that circle. That is, its x-coordinate is the same as the leftmost point in the circle and its y component is the same as the top. I suggest calculating this location and naming it using a **def** declaration. Similarly, calculate the location of the upper left-hand corner of the rectangle forming the handle.

Write the code to create (and name!) the pieces of the light. Using those names you should be able to write the methods moveBy, contains, and sendToFront. Do remember that the contains method should return true only when the point is in the handle..

When you have this much of the light class written, you can test it by writing lightGame, an object that inherits graphicApplication class . It should include code to draw s light somewhere on the canvas and with whatever color you like. Then write methods onMousePress and onMouseDrag that will allow you to drag the light around. We suggest you set the window to be 500 by 500 pixels.

Once onMousePress and onMouseDrag work, the next step should be to add a second light and be able to drag either of the lights around.

We suggest declaring a variable (movingLight) that can be associated with the appropriate light and used to remember which magnet to move whenever the mouse is dragged. This variable will be useful in other parts of your assignment as well. We suggest associating the other light with another variable, say restingLight . You may also find it helpful to have an instance variable dragging to keep track of whether anything is being dragged. (See the Tshirt example from class.)

To finish this part of the lab, please write the code for the balloon class that draws the picture of the balloon. You do not have to worry about the method adjustColor. Test this out by adding a a balloon to the canvas in the lightGame object. Make sure that when the light is being dragged it always show up on top of the other light and any balloons on the screen.

As mentioned earlier, you should bring a design with you to lab. The design should show us how you plan to organize your light and balloon classes and lightGame object to accomplish the actions required of the first part of this lab only. We have told you what methods each class should have and the behavior that they should provide. You should write (in English, not Grace) your plan for how each method will provide the necessary behavior. You should start with the starter file http://www.cs.pomona.edu/classes/cs051G/labs/lights/lights.grace and modify it to contain the complete design.

Be sure to describe in your design (in English) what defs and variables you feel are necessary for

each class as well as filling in a detailed design for the code to initialize objects constructed from these classes. In particular, figure out exactly where each of the objects will be located with respect to the parameters given in the class definitions. Also include a design for the methods moveBy, contains, and sendToFront of class light and the methods onMousePress and onMouseDrag of lightGame. This level of preparation will allow you to progress much more quickly in lab so that you can take better advantage of the presence of the instructors and TAs. This week your design will be worth 10% of your lab grade.

4.2 Part 2: Interactions between the lights and balloons

When the lights are dragged, they should impact the colors of any balloons on the screen. Start by finishing the definition of the balloon class and the method adjustColor. The method adjustColor should be invoked when the balloons are created and every time a balloon is moved (e.g., in the onMouseDrag method.

To show how general your code is, add a second balloon to your canvas and make sure that both balloons change colors when the lights are moved. Notice that each balloon will adjust its color independently from the other. For example, if a red light is dragged directly over one balloon and a second balloon is more than 255 pixels away and has a blue light dragged on top of it then the first balloon will be red, while the second will be blue.

Remember that the balloons don't need to be draggable.

5 Getting it done!

Getting Started The starter folder contains the file lightStarter .grace. This file will initially contain skeletons of the code that you will need to complete. You should not change any of the type definitions in the

Due Date This assignment is due at 11 p.m. on Tuesday evening if you are in the Friday lab, and at 11 p.m. on Wednesday evening if you are in the Monday lab..

Extra credit There are a number of things you can do to get a total of 1 or 2 points of extra credit:

1. Make the lit object more complicated (a snowman?), and have the different pieces be illuminated differently from each other.

2. Add a third light. If you make them red, blue, and green you can get any color. Dragging it will be a bit tricky!

Submitting Your Work Before submitting your work, make sure that your program file includes your name in the comment heading up the code. Also, before turning in your work, be sure to double check both its logical organization and your style of presentation. Make your code as clear as possible and include appropriate comments describing major sections of code and declarations. Make sure your indentation is all consistent.

Turn in your project the same as in past weeks. Be sure to put your program in a folder whose name is of the form LastnameFirstname-Lab3. Then drag your folder to the dropoff folder.

Table 1: Grading Guidelines

Value	Feature
	Design (2 pts total)
1 pts.	Constants & Variables
1 pts.	Methods
	Readability (6 pts total)
2 pts.	Descriptive comments
1 pts.	Good names
2 pts.	Good use of constants
1 pts.	Appropriate formatting (including whitespace)
	Code Quality (4 pts total)
2 pts.	Good use of boolean expressions
1 pt.	Not doing more work than necessary
1 pt.	Using most appropriate methods
	Correctness (8 pts total)
1 pt.	Drawing lights correctly at startup
1 pt.	Dragging a single light
1 pt.	Ability to move either light
1 pt.	Drawing a balloon
1 pt.	Setting the color of the balloon correctly when created
1 pt.	Resetting the color of one balloon when dragging a light
1 pt.	Resetting the color of both balloons when dragging a light
1 pt.	No other problems

A Illumination revealed

For those who might be interested, here is the code used to calculate the color of an object determined by the two lights on the canvas:

```
// Calculate the color of the illuminated object based on the distance of the two
// lights, first and second, from pt.
method getColorFrom (firstLight: Light, secondLight: Light) to (pt: Point) -> Color {
    // calculate distance of lights to center of rectangle.
    // Don't let value get over 300
    def maxDistance: Number = 300
    def distance1: Number = min (pt.distanceTo (firstLight.center), maxDistance)
    def distance2: Number = min (pt.distanceTo (secondLight.center), maxDistance)
    def power1: Number = (1 - (distance1 / maxDistance))
    def power2: Number = (1 - (distance2 / maxDistance))

    // calculate the new color based on distances
    def r': Number = min (255, power1 * firstLight.lightColor.red +
                               power2 * secondLight.lightColor.red)
    def g': Number = min (255, power1 * firstLight.lightColor.green +
                               power2 * secondLight.lightColor.green)
    def b': Number = min (255, power1 * firstLight.lightColor.blue +
                               power2 * secondLight.lightColor.blue)

    color.r (r') g (g') b (b')
}
```

The color of the illuminated object is determined by the colors of the lights and their distance from the object. To simplify our formulas we will assume that points over maxDistance pixels from the object provide no illumination, and that the illumination is inversely proportional to the distance from the lights to the object.

The identifiers distance1 and distance2 are calculated as the distance from each of the lights to pt, however, each is restricted to be at most maxDistance. We use the min function to ensure they do not go over maxDistance

We assume that a light has full effect if it is directly on top of point and 0 effect if it is maxDistance or more away. Identifier power1 is set to the effect of light firstLight as a number between 0 and 1, where there is 0 effect if the point is maxDistance away, and has effect 1 if it is right on top of pt. Identifier power2 similarly is set to the effect of lamb secondLight.

Once we have calculated these effects, the only thing we need to do is to calculate the contribution of each lamp to each of the red, blue, and green components of the final lamps. Each is calculated using the powers of each lamp. For example,

```
power1 * first.lightColor.red + power2 * second.lightColor.red
```

calculates the impact of lights first and second on the red component of the illumination of the object. However, these is one last item that we must take into consideration: while each component is between 0 and 255, the sum might be a number greater than 255. As a result we use the min function to make sure the sum is no greater than 255.

Once we have calculated the red, blue, and green components of the illuminated object, we create the color from those components, and return it from the method.