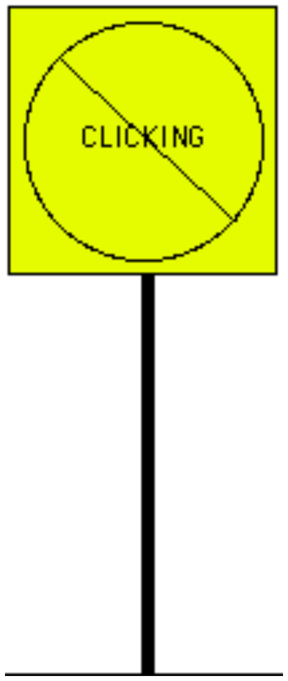
CSCI 051G Practice Laboratory # 1
An Introduction to Grace and the ObjectDraw Library

Objective: To demonstrate the use of the lab computers, Grace, and graphics primitives

—

This lab will introduce you to the tools with which you will be working this semester. Your task is fairly simple. You will construct a Grace program that draws an image resembling a roadside warning sign, but with a message more appropriate for a computer screen, as shown below.



Then you will modify your Grace program so that it modifies the picture in simple ways in response to actions the user performs with the mouse.

Part 1. Logging in & setting up your account

To get started, you must log in to one of the iMac computers in our lab.

Begin by entering the user name and password that you were provided for your computer science account in the login window and click “Log in”. Once you log in, the screen should display a window containing icons for some of the files you can access on the machine.

Hopefully you have already changed the password we assigned to your account to something that only you will know and that will be easier for you to remember. Don’t, of course, make your new password too easy to guess. In particular, at least use something that is a mix of letters and numbers. However, do make sure you memorize it or write it down in some secure spot. You will need to use it every time you use the CS lab. If you haven’t changed it, please do it soon as the password initially assigned to you was not transmitted securely and your account will be hackable if you continue to use that password. To change, see Corey in the office (218 Edmunds) off the glass-walled lab upstairs.

The files you create for our course laboratory projects will not actually be stored in the computer on which you are working. Instead, they will be stored on our department's file server. By entering the account information for your account, you have instructed the computer you are using to access your files on that server. You can access your files from any computer in our lab by simply logging in to that computer.

Never just sit down and start working on a computer in the lab without logging in! If you do, you will lose the work you do as you will be in someone else's account.

Next observe that there is already a folder labeled "CS51GWorkspace" on the desktop for your CS51G work. This is where you will do all of your work for this class.

To work on the lab, you will need to move some files from the course folder (named "cs051G", with a small red arrow in the upper corner) to your CS51GWorkspace folder.

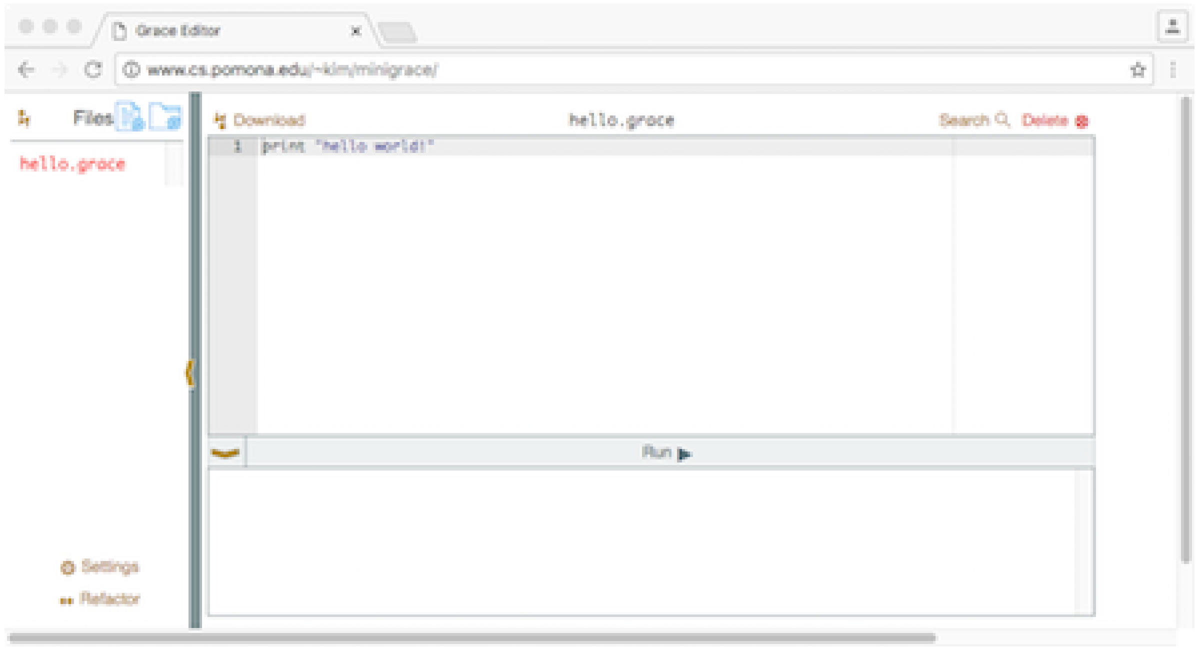
1. Double click on the cs051G folder to open it. Double click on the "labs" folder.
2. Drag the "Lab1-Sign" folder to the CS51GWorkspace folder on your desktop
3. Close the "labs" folder by clicking in the red circle in the upper left corner of that window.

Using the MiniGrace editor & execution environment

We are going to be using a compiler for the Grace language called minigrace (because it doesn't yet do all the things we want in Grace). A compiler is a piece of software that takes a program written by a programmer and converts it to a form that can be executed by the computer. In our case, the program is going to be executed in a web browser, so the compiler will convert your Grace program to a much more primitive program in the language Javascript, the language that web browsers understand.

To get started, go to the minigrace web page at http://www.cs.pomona.edu/~kim/minigrace/. It might take a few seconds to load. Be patient!

The web browser will display a window on your screen that should look like this:



```
print "hello world!"
```

The components in this window are divided into three main units.

The File Browser The area on the left side of the screen lists all of the files that the editor is currently handling. Every file used in your program should be listed there. Click on a file to have its contents appear in the Program Inspector.

The upload button (with upward pointing arrow) above this area can be used to load existing files into the editor. Click on the button, navigate to the desired file and then select open. You can also start a new file by clicking on the icon that looks like a page with a "+" to the right of the "upload" button. It will be important for later programs to organize your code into folders for each assignment. You can create a new folder by clicking on the icon that looks like a file folder (again with a "+" on it) to the right of the new file icon.

Please click on the new folder icon now. When the dialog box comes up, name the new folder, "Lab 1" (without the quotes). When you create or load files into the file browser, you can then drag them into this folder (or just select the folder before creating or loading the file and it will appear inside the folder.

The Program Inspector/Editor Panel The large rectangle appearing on the upper right part of the window is the area in which the your program will appear when you load it. Above this area there are two buttons: "download", with a downward pointing arrow, which can be used to save your file, and "delete", which is used to remove a file from the system. Be careful with this as if you delete a file before saving it, you will not be able to recover it. To save a file, you must hold down the "Control" key while clicking on "Download", type the file name in the dialog box, navigate to where you want the file stored, and then click "save".

Be careful. If you try to save a file that is already saved in the same place, the system adds a " (1)" suffix to the file name. Then when we test your program we will end up grading the old version rather then the new one. To avoid this, change the name of the file back to the original before clicking on save.

Program Output In the bottom portion of the window is an area where program output can occur. This can be either text printed out by your program or error messages generated by your program.

Just above the program output is a button that is initially labeled "run", with a triangle next to it. When pressed it will compile the file showing in the window. While it is compiling it will say "Building" (and appear to shake from left to right). After it has built successfully, it will start executing your program. In most cases this will pop up a window with any graphics items you have drawn in the program.

The first line of the Program window may be showing the program Welcome.grace. Read it for information about the system and click on "Run" to see what it does when it executes.

If that program is not there, you can write your own quick program by clicking on the new file icon (the one that looks like a piece of paper with a + on it). Give the program a name (e.g. Hello.grace) and then type the following:

```
print "Hello, world!"
```

Press the "Run" button in the middle of the window. After it finishes compiling, it will run and display the output `Hello, world!` appear in the Program output window. Congratulations! You have just compiled and run your first Grace program.

To make it even more exciting, go to the Program window and change "world" to your name. Make sure the double quotes stay around the whole phrase to be printed.

```
dialect "rtobjectdraw"

//  CS 51 Laboratory 1
//  Enter your name, lab section and the date here.

object {
    inherit  graphicApplicationSize  (500 @ 400)

    method onMousePress(mousePoint: Point) −> Done {
      // commands placed here are executed after the user  clicks  the  mouse
    }

    method onMouseEnter(mousePoint: Point) −> Done {
      // commands placed here are executed when the mouse enters the program window
    }

    method onMouseExit(mousePoint: Point) −> Done {
      // commands placed here are executed when the mouse leaves the program window
    }

    method onMouseRelease(mousePoint: Point) −> Done {
      // commands placed here are executed when the mouse button is released
    }

    // pop up window and wait for mouse actions
    startGraphics
}
```

Figure 1: The contents of file NoClicking.grace

Part 2: Creating Graphical Programs

Now let's begin to create your program for today. Almost every program you write will use the rtobject-draw dialect. This is a special dialect of Grace that provides you with easy-to-use graphics primitives and allows your program to recognize mouse actions. Dialect *rtobjectdraw is a slight variant of the objectdraw library discussed in the text. The only difference is that with this dialect, you must include the types for all new identifiers in your program.*

Click on "Upload". Navigate to the file NoClicking.grace in your "CS51G Workspace" folder. Select it and click on "open".

The filename "NoClicking.grace" should now show up on the upper left corner of the window, beneath the "Upload" button. The code we have provided you with will be in the edit window to the right. Click on the "Run" button just below the editor panel. While it is compiling the button label will say "Building" and wiggle a bit on the screen. This will likely take a few seconds. When it is done a 500 by 400 pixel window will also appear. Right now nothing will be in the window that popped up, but you will fix that over the course of this lab. Close this window by clicking in the red box in the upper left hand corner of the window. You are now ready to start working on your program.

The text of the program as it should initially appear in your Program window is shown in Figure **??**. The text we have placed in "NoClicking.grace" is the skeleton of a complete Grace program. It includes an object expression whose first line is `inherit graphicApplicationSize (500 @ 400)`. The combination of this line and the last one: `startGraphics` ensures that when the program is executed it will pop up a window that is 500 by 400 pixels and that when there are mouse actions, the system will respond by executing the appropriate mouse event handling methods in the program.

Inside the object expression are headers for the event handling methods you will use. We have not, however, included any Grace commands within the bodies of these methods, only a Grace comment that reminds you when the Grace system will follow any instructions you might add to the method body.

You should always follow our lead and begin writing your program by typing comments rather than actual Grace commands. Near the top of the file we have included a temporary comment telling you to enter your name, lab section and the current date. Such identifying comments are absolutely necessary so the the grader can give you the appropriate grade. When they print out your program to grade it, they do not see the name of the file or any folder it might be in. Hence it is crucial that you place your name as a comment in every file you turn in.

You will be writing and editing your program in the web browser. Your first task will be to add Grace instructions to the object expression to draw a “no clicking” warning sign. The code for this can go immediately after the `inherit` line.

As a first step, let's add the single instruction needed to draw the rectangle that frames the contents of the sign. The form of the command you will need to enter is:

```
framedRectAt (pt) size (sz) on (canvas)
```

where pt should be replaced by an argument of the form `x @ y`, where `x` and `y` are numbers and (x, y) is the location where you wish to place the upper left corner of the rectangle. Similarly `sz` should be replaced by an argument of the form `w @ y`, where `w` is the width of the rectangle, and `h` by the height. (The `canvas` stays as it is.)

We suggest that you place the rectangle so it will be centered in the upper portion of the window (see the picture at the beginning of this document). A width and height of about 100 each should look good.

When you are done adding this line to your program, click on the “Run” button below the program text. If it compiles without errors, it should show a framed rectangle where you told it to create it.

Of course you might have made a mistake when you typed the program and you might have gotten an error message in the output pane. If so, you should read the error message and correct the problem. The error message should give you a line number and explain what the problem was. Hopefully it will point out exactly the error you made, but sometimes it gets confused and will point at somewhere different than where the error actually occurred. Try correcting your error and then click “Run” again. If you can't see what your error is, ask a TA or the instructor for help.[1]

You may not be happy with where your framed rectangle appears on the screen. If so, modify the parameters to the construction and “Run” until you are happy with it.

Now that it does something, it would be a good idea to place a comment on the line before the construction to say what it does so far. A comment starts with //. Anything after those two characters to the end of the line will be ignored by the Grace compiler (but not human readers!), so write whatever is helpful to a reader to understand your code. You should get in the habit of adding and updating comments to keep them as accurate as possible as you add instructions to your programs. Not only does this mean you will not have to go back and add comments when you're done, it will help you to remember what everything does as you are writing your programs.

If you are working on the iMacs in our lab and have not used Macs before, there really is not much

[1] Because our programs popup a new window with the graphic in it, if your browser is set to block pop-up window, your programs will not execute. Instead you will get a message asking if you are blocking pop-ups. If this happens to you, look for an icon with a small red “x” on the right side of the field where you type URL's. On my computer, this is just to the left of a blue star. Click on that and then select ”Always allow popups from http://www.cs.pomona.edu.” If that doesn't work, see the instructor or mentor to go into Chrome preferences to fix it.

difference between it and Windows (and Linux) any more. Cutting and pasting on the Mac is similar to that with Windows except you use the "Command" or "fan" key, to the left of the spacebar, in combination with "x" (for cut), "c" (for copy), and "v" (for paste). You can erase text by selecting it and then hitting the backspace key.

Once your rectangle code is working, you should add more instructions to it to turn it into a complete warning sign drawing program. Add additional lines to create a `textAt (pt) with (contents) on (canvas)`, a `framedOvalAt (pt) size (sz) on (canvas)`, a `lineFrom ( start ) to (end) on (canvas)`, and all the other components. You will need to experiment to find reasonable coordinates and dimensions for the various objects. By the way, in our drawing, the post is a rectangle, while the base is a line.

In the picture, we show a yellow background for the sign. Leave that part out of the version your program draws for now; we will add it later. As you work, it is a good idea to "Run" your program every time you add a line or two to ensure that you catch mistakes early. Also, make sure your comment lines get updated by the time you are finished. When you are done, your program should draw a sign when it is executed.

This is a good time to save your program. While holding down the "Control" key, press on the "Download" button, and select "Save link as ...". Navigate to the right folder, make sure that the file name is correct, and then click "Save".

By the way, every once in a while something evil will happen and the web browser will crash. Don't panic! When modern web browsers are restarted, they generally restore whatever state they are in, including your program text. If the browser has difficulty, make sure it is restoring only your program text and not the window with the canvas on it. We've generally experienced little difficulty with this.

Making Your Program Responsive

Now, to explore event handling methods a bit more, revise your program so that it reacts to the mouse in more interesting ways. In the revised version, the sign will change as the program runs. In particular, when the user moves the mouse into the program window or presses the mouse, your revised code will alter the appearance of the sign.

Making the sign change in response to the mouse is a bit tricky. Suppose, for example, that you want to emphasize the sign's warning by changing the color of the word "CLICKING" from black to red when the user moves the mouse into the program's window. There is a method `color:=` that you can use to change the color of the text. There is also an obvious place to tell Grace to make this change. The `onMouseEnter` method is executed whenever the mouse moves into your program window. Placing an appropriate `color:=` in that method would do the trick.

The problem is that you can't simply say `color:=` in the `onMouseEnter` method. If that was all you said, Grace would have no way of knowing which of the several objects' color to change. It could change the rectangle, the oval, the text, all of them, etc. Your code has to be more specific and identify the object that should change.

To be able to specify the text object as the object whose color we wish to set, we will have to give it a name. We will use the name `message`, but we could use any name that seems appropriate.

Associating a name with an object requires that we include a line that introduces the identifier and that provides an initial value. We plan to associate the name `message` with the object created with `text`, so we have to tell Grace that the name will be associated with that object. The form of a Grace definition is simply to write the keyword **def** followed by the name being declared, the type of the value represented, =, and the value the identifier will be associated with. So, the form for our declaration is:

```
def message: Text = textAt (...) with (...) on (canvas)
```

where the code to the right of the = is exactly what you had written originally to create the text. As a result, all you must do now is to add the `def message: Text =` before the existing code.

Notice the difference between the class name text, and the type Text. The first represents the code used to actually create the object on the screen, while the second tells you the kinds of things you can do with it. Think of it as being like the difference between being able to build a car and understand how everything works, to instead just understanding that turning the wheel and pressing the gas pedal change the direction and speed of the car. For the objects constructed using the objectdraw library, we just need to know how to use the objects (hence the types are most important). In a few weeks, however, we'll start building our own objects, and we'll have to focus on the details of actually building them. We distinguish between types and classes by always starting type names with capital letters and classes with lower-case letters.

Now that the text has a name, we can use the name to change its color. Within the body of the onMouseEnter method add the line:

```
message.color := colorGen.red
```

Then, run your program (correcting any errors as needed).

The program isn't quite complete. It draws the sign immediately and makes the text turn red when the mouse enters the window, but it doesn't make the text black again when the mouse is moved back out of the window. You should be able to figure out what to add to make it black when the mouse exits. Give it a try.

To get more practice using names and other event handling methods, we would like you to modify your program a bit more. First, change the program so that while the user is depressing the mouse button, the circle with the diagonal line through the text disappears. (Of course, it should reappear when the mouse is released.) This will require that you declare names for the circle and the line and associate them with the correct objects. This can be accomplished by adding **def** ... = to the beginning of the statements creating those objects. The circle can be given type Graphic2D and the line type Line. Your code to make the objects disappear and reappear goes in the onMousePress and onMouseRelease methods. You can use the visible := methods with right hand side either false or true to handle the disappearing and reappearing.

Finally, add the yellow background to the sign using a filled rectangle. We didn't have you do this earlier because you had not yet seen how to associate names with objects. Associate a name with the background rectangle when you create it and then use the name to set its color to yellow. Think carefully about where to put this code in your program so that the shapes are stacked in the right order. If you put it in the wrong place then the circle, line and text will be hidden behind the yellow filled rectangle!

Grading Guidelines

In general, about half the points in each lab will be for correct functionality and the other half for the quality of the code you write in the program. This lab is worth 10 points, which are distributed as shown in Table **??**:

The last point in the table is worth additional discussion. Often there are several Grace commands that can be used to accomplish the same result. It is important to pick the most appropriate commands. For example, in this lab you could make your circle and slash disappear by sending them behind the background, but this would not be the best choice as it is not obvious what your intent is and there is a more direct way of accomplishing the same result.

Table 1: Grading Guidelines

Value	Feature
2 pts.	Drawing the sign correctly when the program starts
2 pts.	Changing the color of the text correctly
2 pts.	Making the slash and circle disappear and reappear correctly
1 pt.	Meaningful names used in definitions
1 pt.	Informative comments
1 pt.	Good and consistent formatting
1 pt.	Good choice of Grace commands

Submitting Your Work

Congratulations, you have written a Grace program! You are encouraged to continue experimenting with it. Before you do, however, you should submit it as a completed assignment. The submission procedure is electronic and will be basically the same every week.

- First, return to the editor and make sure you included your name and course number in a comment at the start of the program. Then save it one more time.
- On the desktop, navigate to your folder called "Lab1-Sign" in your CS51GWorkspace folder. You should now rename that folder by clicking on it and pressing return, and giving it a name of the form Lab01_LastnameFirstName, where you replace Lastname and Firstname by your namesr. Thus, if your name is Jane Doe, you would name this lab Lab1_DoeJane. Given the number of students in CS 51 (all flavors) we need you to use this kind of uniform naming scheme to make sure we don't accidentally miss your program. Do not use spaces or periods in your folder names.
- Now open the "cs051G" folder icon on the desktop by double-clicking on it. Within the "cs051G" folder you should see a "dropbox" folder.
- Drag the folder you just created into the dropbox folder. When you do this, the computer may warn you that you will not be able to look at this folder. That is fine. Just click "OK".

You can submit your work up to 11 p.m. on Tuesday evening. If you submit it and later discover that your submission was flawed, you can submit again. We will grade the latest submission made before the 11 pm deadline. The computer may not let you submit again unless you change the name of your folder slightly. It does this to prevent another student from accidentally overwriting one of your submissions. Add a version number to each new submission. For example:

- Lab01_LovelaceAda . . . original submission
- Lab01_LovelaceAda_V2 . . . first revision
- Lab01_LovelaceAda_V3 . . . second revision

Please do not wait until the last minute to submit your program. We will use the time recorded by the computer to determine whether the program was on time.

When you are done with your session, be sure to log out of the computer by selecting "Log Out" from the "Apple" menu as you did earlier. Please log out each time you leave as otherwise anyone who sits down at the computer can get full access to all of your files.