CIS 410/510: User Interface Programming
Programming Exercise #3

Due February 8, 2007 8:30 am

PART III – Extra Credit: 20 points

After getting warmed up with the minor GUI calisthenics of Part I, Part III asks you to explore some aspect of interface programming in more depth for extra credit. To make this assignment challenging for everyone, we've divided the choices into two sets: one for graduates and one for undergraduates. Add new functionality to *sketch* by choosing **one** of the extensions from below. As a general rule, keep in mind that we are not only interested in code that works but also in aesthetics, elegance, and a discussion of the design choices you make. What are some alternative design choices? Why do you believe your choice is better (or worse) than the alternative(s) in terms of the performance measures we have talked about in class?

Choices for UNDERGRADUATE students:

(UG#1) Add polygon drawing commands. The drawing facility must incorporate at least the following: a) a way to do this; and b) implementation of polygon drawing in the drawing space. What polygons should you implement? What's the best way to select them? How does this change the existing control widgets (menus, etc.) for *sketch*? Will you need any additional functionality, e.g. filling?

(UG#2) Add the ability to change the size of the brush. This facility must incorporate at least the following: a) a way to do this; and b) implementation in the drawing space. What sizes should you implement? What's the best way to select them? How does this change the existing control widgets (menus, etc.) for *sketch*? Will you need any additional functionality, e.g. zooming?

(UG#3) Add draggable text labels in the drawing area. This facility must incorporate at least the following: a) a way to do this; b) implementation in the drawing space; and c) minimum text editing functions. What editing functions should you implement? What's the best way to select them? How does this change the existing control widgets (menus, etc.) for *sketch*? Will you need any additional functionality, e.g. fonts?

EXTRA EXTRA CREDIT: For undergraduates, do any of the following Graduate student extensions.

Choices for GRADUATE students:

(Grad#1) To UG#1 above, add filling and patterns.

(Grad#2) To UG#2 above, add zooming.

(Grad#3) To UG #3 above, make it possible to change the font and the font size in the same general way that it is used in popular applications. That is, if text is selected, the change is applied to that text. Otherwise, it changes the current mode so that the change applies to whatever is typed next. Add the ability to change character style (bold, italic, etc.)

(Grad#4) Add the ability to save and retrieve files. This will require additional menu items as well as the ability to dynamically name files. Test this cross-platform. It should run on UNIX and at least one other platform (Mac, PC, etc.)

(Grad#5) Add multiple drawing windows. This will require additional menu items. Give windows distinct names as they are opened. For example, the first time you use the New option the new window should be named "Untitled1," the second time the new window should be named "Untitled2," and so on. Also, add a Window menu to Sketch. This menu might be similar to the Window menu in other applications . Why? It will list the names of all open windows, with the currently active window checked or highlighted. The user may switch active windows by selecting one of the others from the menu. Obviously, the list of names will grow and shrink in size as text windows are opened and closed. If a window name changes the corresponding item in the list will also have to change. Finally, indicate in the menu whether a window is dirty or not. (A window is dirty if it has been modified since the last save.)

Things to think about:
(a) It has been observed that the Window menu is often underutilized, especially by novices. Even though they have been instructed about the functionality of the window menu novices tend to use non-optimized methods to activate windows. For instance, if a window is completely hidden by other windows novices tend to resize and/or move the windows on top in order to activate the window underneath (by finally clicking on it). What do you think some reasons for this behavior might be?
(b) Conjecture how we, as user interface designers, can make it easier for novices to realize/remember that there is an easier way to activate hidden windows (besides reminding him or her about it a bezillion times). Try to be creative. Maybe there is

some way besides using a menu that will work better. Write a paragraph or two in which you describe your solution to an improvement of the Window menu.

Extra Extra Credit: Add one of the following extensions to the above:

1) One way to make the Windows menu more accessible is to allow it to appear as a "floating" on-screen control, which is continuously visible. Under the Window Menu, implement a "Show Menu Onscreen" function that, when selected, makes the Windows menu appear as a (draggable) control window on-screen. This window should "float" above all text windows so that it can't be obscured. For an example of how this sort of thing works, look at some of the popular drawing packages.

2) Provide a graphical short-cut to the "Show menu Onscreen" menu item by allowing the user to simply "tear-off" the Window menu. This means that when you move beyond the bottom of the menu after you pull it down, the menu "tears off" (insert fun sound effect?), follows the cursor, and becomes a new onscreen control wherever you drop it.

Turn-in a brief README that explains which extra credit challenge you did, the path to your Tcl/Tk source programs and executables on the CIS computers, and the version of Tcl/Tk you are using. All of your Tcl/Tkcode (.tcl) must be world-readable, and the directory that contains your java code must be world-executable. (You can set this the day you hand it in to me.) Please be sure your program will run on the CIS system (Tcl/Tk version 8.)