CMSI 370-01

INTERACTION DESIGN

Fall 2015

Assignment 1211 Feedback—Direct Manipulation Widget

Christopher Dellomes *cjdellomes / cjdellomes@gmail.com*

Notes while running (asterisks indicate major observations):

- Your demo runs generally OK, but the default browser behavior is interfering with the drag. That is probably why you allowed a click, but ideally the drag feels more natural. (*2b*, *3a*, *3b*, *4a*)
- Upon an initial click, the "ghost" object temporarily appears in the wrong location. It shouldn't—this can disorient users. (*2b*, *3a*, *3b*)
- The trash can should provide some kind of feedback that a drop can take place. (*2b*)
- Mouse positioning within the dragged object can be better too (i.e., sync mouse position with mouse-down location). This was in the boxes sample code. (*2b*, *3a*, *3b*, *4d*)
- Integration with the front-end is straightforward and natural. Of course, with full permissions, this action would perform an actual delete (or anything else that can be signified by a drag-and-drop). (+*2b*, +*4a*, *4b*)

Code review (asterisks indicate major observations):

1. Yay, no tabs…in your widget code. Tabs lingering in *api-front-end.js*. (*4c*)
2. **** Hmmm, not a good sign: this is (should be) just the plugin code. The code that *invokes* the plugin needs to be *outside* the plugin… (*4b*)
3. **** And here's the culprit—this should have been *outside* the plugin code. (*4b*)
4. Indent line-broken constructs. (I usually do double-indent to emphasize that it is a continuation of the line above) (*4c*)
5. **** Magic numbers/hardcodes in the condition below. Not good. (*4b*, *4c*)
6. An `else` clause is still in the same statement as the preceding `if`, so don't break them up. (*4c*)
7. In a fully-realized version of the plugin, this should be a customizable callback. (*4b*)
8. **** Why is your plugin file *copied?* I told the class that I would run the web server *above* the directories so that you can use relative URLs, avoiding copied code. (*4b*)
9. **** And of course, the other issue is the very order of loading/invocation. You are forced to load in this way because you are *invoking the plugin from within its own file*. That is not right. The plugin should be treated like a *library*—load it first so it is ready to use when needed. Then, your main code starts up, and *somewhere in there*, the plugin is invoked. This is shown in all of the code you've seen, whether it's my samples, or Bootstrap, or any other plugin usage. (*4b*)
10. This is where you can trigger some trash feedback so that the user knows when they are "in range." Will need some refactoring of course (i.e., the bounds-checking code). (*2b*, *3a*, *3b*)

2b — **|** …Decent general experience, but with some lingering glitches.

3a — **+**

3b — **+**

4a — **+**

4b — **/** …So the functionality all checks out; reusability and generality are the major areas of improvement.

4c — **|** …Code is mostly presented well but with a few hiccups…magnified because the overall code is relatively short.

CMSI 370-01

INTERACTION DESIGN

Fall 2015

Assignment 1211 Feedback—Direct Manipulation Widget

4d — **|** …Information used well for functionality; need more background and exposure for proper code reuse and generalization.

4e — **+**

4f — **|** …Work started and done one day after the due date.