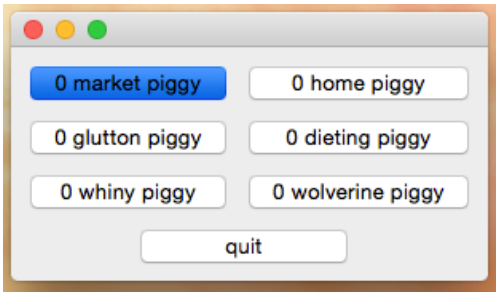
EECS 493 F15

Assignment 3: piggy-stack

This assignment continues with subclassing and callbacks. Its major purposes are to have you to stress your understanding of how to arrange code (specifically to deal with subclasses, callbacks, and user functionality), as well as to create an execution stack, which is common in UI-based systems.

We're going to play with the "piggy" application. It should still look like:




1. The top-level widget for the application should remain a subclass of QMainWindow. However, this time, we're going to add menus. The File menu should have greyed out buttons for Open, Save, and Save As.... as well as a functional menu item for Quit. (Notice: On the Mac, the Quit menu item will get automatically moved to the menu item for the app, in this case, piggy-stack.) The Edit menu will have Undo and Redo; we'll talk about those below. This menu, or others, may also have some menu items automatically added by the platform. Finally, there will be an Actions menu, consisting of Increment, Decrement, and Print Amount. Increment should have the shortcut Command- or Cntl-I, and Decrement should have the shortcut Command– or Cntl-D. Print Amount should have the shortcut Command- or Cntl- P. Undo and Redo should have shortcuts Command- or Cntl-z and Command- or Cntl-Y, respectively.
2. You should turn the piggy buttons into radio buttons. That is, at most one can be "active" at a time. This is a bit arcane in Qt, and this assignment is not about the syntax, so the next few paragraphs walk you through one way of doing this – there are multiple ways. First, create a QButtonGroup. You will place all of your QPushButtons or subclasses into the button group with an addButton() call (part of QButtonGroup). Call setCheckable() for each button and setChecked(false) to each button except the one you want as an initial default. Each time a button in the button group is toggled, the button group will throw a toggled(bool) signal. No need to show a message box with nursery rhymes when clicked like the assignment 2.
3. Each button will have a number in front of the name of the button, which indicates the count for that piggy button (starting at 0). When the user selects Increment, the value for the active button should be increased by 1; when the user selects Decrement, the value for the active button should be decremented by 1. When the user selects Print Amount, the current value for the active button should be printed (including the button number and value). You may use a cout statement.
4. The Undo and Redo should be a functional command execution stack that undoes and redoes the most recent operation based on a stack. The stack should be infinite in

length (although the GSIs are unlikely to test this). Print Amount does not need to be undone or redone.

The easiest way to do this, although it does require some overhead, is to make sure your commands into a subclass of QUndoCommand. This is required.

The emphasis in this assignment is, again, not the syntax, but the basic abstractions involved in undo/redo and actions overall and how to arrange code between instances. Therefore the next few paragraphs walk you through what you will need to use.

You can create an undo stack for your program by creating an instance of QUndoStack. Then you add can add the menu items for Undo/Redo with the createUndoAction() and createRedoAction() methods, which are part of QUndoStack.

You place a subclass of QUndoCommand on the stack with:

```
undoStack->push(incrementCommand);
```

or something similar. The undo stack then takes it from there; you don't have to do anything else to get undo and redo in your program. Note that QUndoCommand or subclasses have two methods, the first being an undo() method and then the redo() method. The redo() method will get triggered when the QUndoCommand is constructed – be aware of this. You must maintain whatever state you need (not much in this case) to restore state in either direction.

5. All that should be in your main() is the constructor for your subclass of QMainWindow. ( The main will also include the bookkeeping of creating a QApplication, the show(), and the main event loop. You would not need more than 4 lines of code to do this. )
6. You may choose the margins for the buttons and any white space. You may take the defaults.
7. You are ***not*** allowed to use a GUI builder (e.g., Qt Designer) nor have a ui file included in the project. You must hand-code this assignment.
8. You will be graded on basic functionality – correct use of subclasses, hooking in the events (signals and slots) appropriately, increment/decrement, undo/redo, output, and correct layout (*including* the capability to resize appropriately, menus).
9. You are responsible for monitoring the Canvas site for any modifications or corrections of this assignment (or any assignment).

This is due Sunday 10/4 at 11:55pm. You must upload your code to Canvas to submit. Same rules as assignment 1 on the naming and format of the submission.

This assignment is less straightforward than the previous two. However, if anything is taking a long time, you should ask for help.

To startup in QtCreator: See assignment 1.