Use Case Tests

Use Case: Open a file
Test: Main Flow
Result: Success
Test: Exceptional Flow b
Results: Fail, Success

Use Case: Save a new file
Test: Main Flow
Results: Success
Test: Exceptional Flow b
Results: Fail, Success

Use Case: Save a previously saved file
Test: Main Flow
Results: Success

Use Case: Create a new Button
Test: Main Flow
Results: Success
Test: Exceptional Flow b
Results: Success

Use Case: Create a new Command Box
Test: Main Flow
Results: Success
Test: Exceptional Flow b
Results: Success
Test:
Results: Fail, Success
Test:
Results: Fail, Success

Use Case: Create a Shortcut for a Button
Test: Main Flow
Results: Fail, Success
Test: Exceptional Flow b
Results: Fail, Success
Test: Exceptional Flow c
Results: Fail, Success

Use Case: Create a shortcut for Save, Save as
Test: Main Flow
Results: Fail, Success

Use Case: Edit a Button
Test: Main Flow
Results: Fail, Success
Test: Exceptional Flow b
Results: Success

Use Case: Edit a Command Box
Test: Main Flow
Results: Fail, Success
Test: Exceptional Flow b
Results: Success
Test: Exceptional Flow c
Results: Fail, Success

Use Case: Delete a Button
Test: Main Flow
Results: Fail, Success

Use Case: Delete a Command Box
Test: Main Flow
Results: Success

Use Case: Save Settings
Test: Main Flow
Results: Fail, Success
Test: Exceptional Flow b
Results: Success

Use Case: Load Settings
Test: Main Flow
Results: Fail, Success
Test: Exceptional Flow b
Results: Success

Use Case: Run a command with a Button
Test: Main Flow
Results: Success
Test: Exceptional Flow b

Results: Success

Use Case: Run a command with a Command Box
Test: Main Flow
Results: Fail, Success
Test: Exceptional Flow b
Results: Success

Use Case: Close and Open Code Workstation
Test: Main Flow
Results: Fail, Success

ORIGINAL DEPRECATED CONTENT - 1.0 Released - Rolling development and Bug fixing from here on out.

Buttons

How it is implemented:

We will be extending the QPushButton class and overloading the mouseRelease(event) function. In our version of the function it will check if the button has been hit by using the hitButton() function. If this function returns true we know the button has been pressed and that we want we want to run a certain command. We will run these commands using the subprocess library and popen to create a subprocess and retrieve the output of the command. The button should then append to the text in its assigned Command Box.

What could go wrong:

- Users could create too many buttons
- Users could insert nonsense into the functionality of the button
- Users could press a button too often/quickly
- Users could press but not release mouse cursor over button
- Users could fail to assign a command box to the button or delete the selected command box, leaving the button with nowhere to output
-

Tests/Results:

- Fast Pressing - Should result in commands displaying in selected command box
  Results: Commands attempt to run 10 times per second and in testing no one would press a button that fast
- Bad Command testing - Bad command should execute and spit error into selected command box
  Results: The error code is displayed in the selected command box
- No command box pressing
- Pressing while command boxes are updating - No text should be lost and each print into the command box should be relatively atomic.

Text Editor

How we'll implement it:

We will extend the QTextEdit class. We will override the existing keyPressEvent() function in order to implement shortcuts.

What could go wrong:

- Users could input too much text for the editor to hold
- Users could try typing in command box(es) rather than text editor
- Users could drag/paste unsupported text formats into it.

Tests/Results:

- Drag in a bunch of unicode characters - Characters or '?'s should appear no crashing
- Paste in 500mb of text - it might crash idk

- Paste in 100mb of text - window should hang for a few seconds then paste successfully

Command Boxes

How we'll implement it:

We will be extending the QDockWidget class to create movable resizable text boxes. Inside the dock we will have a QTextBrowser and server settings for autorunning. The command boxes should hopefully run commands the same way that buttons do. We will edit their mousePressEvent() function to allow them to right click to open a menu which gives them access to the command box dialog.

What could go wrong:

- Users could try typing in command box(es) rather than text editor
- Users could try to have multiple buttons outputting to the same command box

Tests/Results:

- Have buttons and autorun pushing text at the same time.

File Dropdown

How we'll implement it:

What could go wrong:

- Users could click on a menu option but then cancel every time
- Users could click on a menu option, then lose track of the popup

Tests/Results:

- Click save with no filename specified

New Dropdown

How we'll implement it:

Menus exist in QMainWindows already and their functionality can easily be changed.

What could go wrong:

- Users could click on 'button' option, then exit out of the popup before creating the button
- Users could click on 'command box' option, then not choose a dock to place it on
- Users could place too many command boxes on any given dock

Tests/Results:

-

Keyboard Shortcuts

How we'll implement it:

We will simply read in the shortcuts text file on program load. And have a loop in the overridden keyPressEvent function which loops over the shortcuts and checks for them.

What could go wrong:

- Users could try to use keyboard shortcuts not in the shortcuts file
- Users could edit the shortcuts file but not save their changes
- Users could assign program-breaking commands to keyboard shortcuts

-

Tests/Results:

Button Creation Dialog

How we'll implement it:

An extension of the QDialog, these will save all the data contained in their QLineEdits when the 'okay' button is pressed.

What could go wrong:

- Users could quit before applying changes
- Users could enter nonsense commands into dialog
- Users could press 'okay' before filling in all fields

Tests/Results:

Command Box Creation Dialog

How we'll implement it:

This should be similar to the Button Dialog but with data from QAbstractButtons as well.

What could go wrong:

- Users could press 'okay' before filling in all fields
- Users could quit before applying changes
- Users could put too many command boxes on one dock

Tests/Results: