Attendance:
Margaret Kreutz, Nick Potter, Jarrett Nelson, Katie Hancock

Agenda:

1. Discuss how we plan to plan color schemes without recompiling.
    a. Have a drop down menu of color schemes
    b. On enter, refresh the frame with the new color scheme
    c. This can be done in the
    d. Could use Enum to define each of the color schemes and pass these into the UVSim. Import it in and then instantiate it
2. Discuss what sections will be what colors?
3. What class changes need to be made?
    a. Change the class that deals with parsing the input file
    b.
4. How will we address our TODO items.

5. How will we allow manual editing? Will it be a line-by-line edit?
    a. Push some button to open a new window
    b. We don't immediately run the file. Have the editing screen automatically pop up
    c. Don't parse the data until after the user has updated it
    d. Leave it in a string format.
    e. The editor window would be more similar to a text editor.
    f. OR we can populate something more similar to an Excel sheet.
6. How will we provide tools for cut, copy, and paste multiple lines at a time?
7. How will we do error checking to prevent going beyond 100 lines of data?
    a. This can happen during the parsing after edits are submitted.

TODO:

1. I'm still getting 2 of the unit tests failing (write_1 and write_2) (8/10)
    a. Possibly just a test rewrite is needed

2. The major issue is as you've noted that you need to figure out how to do any inputs through the GUI rather than the console,
    a. Implement a call-back function that will send something in once we receive user input
    b. We need to tell UVSim that we received an update
    c. We run lines until we need to get input then we move up in layers
    d. Once the input field is finished, then a listener command will use a call-back function to send the data back to UVSim

3. The other issue to put on your TODO list is it's inconvenient and non-user-friendly to have to shut down and restart the app if we want to switch files, look at how you can combine your file code from the first window into the run code on the second window so you can select a new file and run the currently selected file any time.

    a. Have the file selection as its own function and have a listener function (on -press) for when a new file has been selected to be run. Then start the application from the beginning, with the newly selected file.

4. I'd recommend splitting off the file processing code to its own class (the idea with the n-tier layer architecture is that if you, for instance, switched your data input code from a file input to a database or web service input (or all of the above), you would be able to go to your 'data import' class and change it there without needing to change your UVSimulator class at all (because it doesn't and shouldn't care where the data comes from, just that it gets the function codes into memory in the end).
    a. Create a new class that will parse and return the file code

5. Only comment would be non-functional requirement #2 says to make sure your window can be resized, but non-functional requirement #3 seems to say you want a fixed size window that's not resizable (or perhaps you mean minimum size instead).

6. Just remember your README is the "user manual" for your app, so be as 'beginner/tutorial' focused as you need to be to make sure anyone can run your app with this open and won't be confused about what to do or what they are seeing. As we expand the GUI functions here in the next few milestones, just make sure your "user manual" is keeping up.
    a. Update Readme to be more descriptive about how to use the GUI

7. TODO for next time: Fix the input and file issues outlined above, take another pass at your architecture with the milestone 4 features in mind. Otherwise, looking pretty good -- let me know if you have any follow-up questions, but you should be in good shape for milestone 4

Nick will get some advice from his dad and figure out how we should split up doing research on new topics.