CSSE1001 Assignment III Reflection

Alex Wilson [41552995]

In total, I feel that the project has worked out quite well. The analysis library is feature-complete as set out in the initial library, and it passes its unit tests with flying colours. The GUI works quite well, although it still has some bugs due to lack of time to do much real testing of it, and implements the vast majority of the features that were intended, although in a different way than I had originally envisaged it. The idea of using a clickflow continuation model for the commands in the GUI was conceived during development, and with the py.magic library, works in a very satisfactory manner.

Time management (or lack thereof) was probably the biggest issue in the course of the project. Most of the documentation was written (literally) at the last minute, including this reflection itself. This is quite typical of software projects, unfortunately, and is often a major factor leading to unsatisfactory solutions. In this case, I do not think the project has suffered terribly for the issue, but it was only with some very late nights in the last week that it was completed on time.

Probably the most 'difficult' issue faced in development was finding a way to create widgets on the GUI representing analytical components, and have a way to serialize them in order to save and load to/from disk. The first design option considered was a multiple inheritance approach, creating widgets that subclassed both the circuit library components and QWidget. The resulting confusing mix of attributes and methods from either framework meant that this approach quickly grew to be far too complex. The second attempt was using wrapper classes subclassing QWidget, holding references to the circuit components and serializing those instead. This also grew quickly to be very complicated and was abandoned. Finally, I thought up the current approach, of using dedicated GUI widgets, as well as "sleepies" – special serialization helper classes that encapsulate a GUI widget and hold all the information necessary to recreate it, whether that may be on the GUI, or in the analysis library. In this way, to analyze a circuit, all that is necessary is to take the GUI classes, call makeSleepy() on each, and pass this to the AnalysisCircuit helper class, which then wakes the sleepy objects into a full analysis tree ready for action.

Another interesting issue was the selection of a system to solve the nonlinear equations held in the maths module. The SciPy optimizer library was a fairly clear choice here after some research, however, due to its flexibility and speed.

Bugs and improvements

There are a few random bugs in the GUI components that I have been unable to track down, mostly involving quirks in the layout code generated by the Qt Designer. Particularly, connecting two components directly to one another without a node in between will throw an exception in analysis (this is a restriction of the analysis library). It would have been nice to produce more

friendly messages detailing this fact rather than a cryptic “object has no next()” on the console.

Other things I would have liked to improve would be the toolbar design and layout (adding icons, for one, instead of just text would be nice), and adding the ability to not only perform a Thevenin/Norton equivalence probe and display the result, but also to actually collapse the circuit into its equivalent circuit for the user, setting appropriate attribute values on the fly.