CSSE1001 Assignment III Design

"DC Circuit Analyzer"

Alex Wilson [41552995]

Description and overview

The proposed project is to create a simple DC circuit analyzer library and accompanying GUI circuit editor. The approach used for circuit analysis is a hybrid of node-centric and mesh-centric approaches, in order to create a robust and reasonable-performance analyzer using the Levenberg-Marquadt algorithm for constraint solving.

It comes in two parts: a circuit analysis library, limited to constant-time analysis methods, and a GUI interface for intuitively creating and editing circuits, as well as performing analysis and reading returned figures.

Design details and structure: circuit library

The circuit library is broken down into three fundamental pieces:

1. Mathematical solutions [maths]
2. Basic classes for components, nodes, connections [base]
3. Individual component implementations [components]

Design: mathematical solutions module

This module is primarily responsible for managing the systems of non-linear equations resulting from circuit analysis, solving them, and organizing the returned data so that it is simple to interpret. This task is broken down into several key classes:

- **Variable**:
  A data component representing a variable in the equation system. It possesses an owner and a type which are used to uniquely refer to it within a VariableSet.
- **VariableSet**:
  A set of Variables, generated from the current state of the solver. This is a helper class to provide a consistent interface to the current semi-optimized values of variables in the system. It supports the subscript operator, taking two parameters, owner and vartype, used to look up a unique variable.
- **FunctionSet**:
  This class represents a set of functions and unknowns that can be solved using optimization. It keeps two lists, of functions and variables, and supports a function optimize() that returns the optimized values of the variables as a hash.

This module also contains the library's dependency on SciPy, in the FunctionSet.optimize() method.

Design: base classes module

This module is broken down into classes representing individual components, the connections between them and nodes used in circuit analysis. It also has higher-level classes representing entire circuits.

- **Connection:**
  Represents a connection between a node and a component. It keeps track of the variables associated with the connection (usually current and voltage), and their coefficients from the perspective of the node.
- **Node:**
  Represents a voltage node in the circuit. Nodes have an infinite number of connections attached to them, to which they apply Kirchoff's Current Law. Additionally, one node in each circuit is usually nominated as the "ground reference", whereby its voltage is specified as zero. Other node voltages are then measured relative to this ground (the actual calculation of node voltages is implemented in the Circuit class).
- **Component:**
  A base class for circuit components. It includes a simple initializer to set the component's nickname, and several utility functions for connections and finding a component's orientation. Base classes override the addTo(fs) method to add their own variables and constraints to the circuit.
- **Circuit:**
  Represents an entire circuit. It keeps lists of nodes and components involved in the circuit, and provides a front-end to solving it. As well as collecting the combined variables and constraints of all circuit components, it also recursively applies Kirchoff's Voltage Law in loops around the circuit, and creates constraints to calculate node voltages relative to ground.

Design: components module

This final module consists predominately of subclasses of the classes in the base class module, particularly subclasses of Component. Some of the classes that will be implemented:

- **IdealVoltageSource:** an ideal source of voltage in the circuit. Takes one parameter only: the voltage to raise.
- **IdealCurrentSource**: an ideal source of current in the circuit, similar to IdealVoltageSource. Takes one parameter only: the current to produce.
- **IdealResistor**: an ideal resistor, with the resistance specified in ohms.
- **ShockleyDiode**: an idealized diode obeying the Shockley Equation. Takes the parameters of the Shockley function: saturation current, thermal voltage, and an ideality constant.
- **RedLED**: a real-life model of a 3mm red LED used in School of ITEE lab pracs, using a simple exponential function.

- **YellowLED**: same as RedLED, but for the yellow variety (which has slightly different characteristics)

Design details and structure: GUI components

The GUI classes are divided into two primary groups: active UI classes, and "sleepy" classes, which can be serialized or used for analysis. For each of the active UI widget classes representing circuit components, there is a corresponding Sleepy class that encapsulates all of the information necessary to recreate the object, or to represent it for the analysis library.

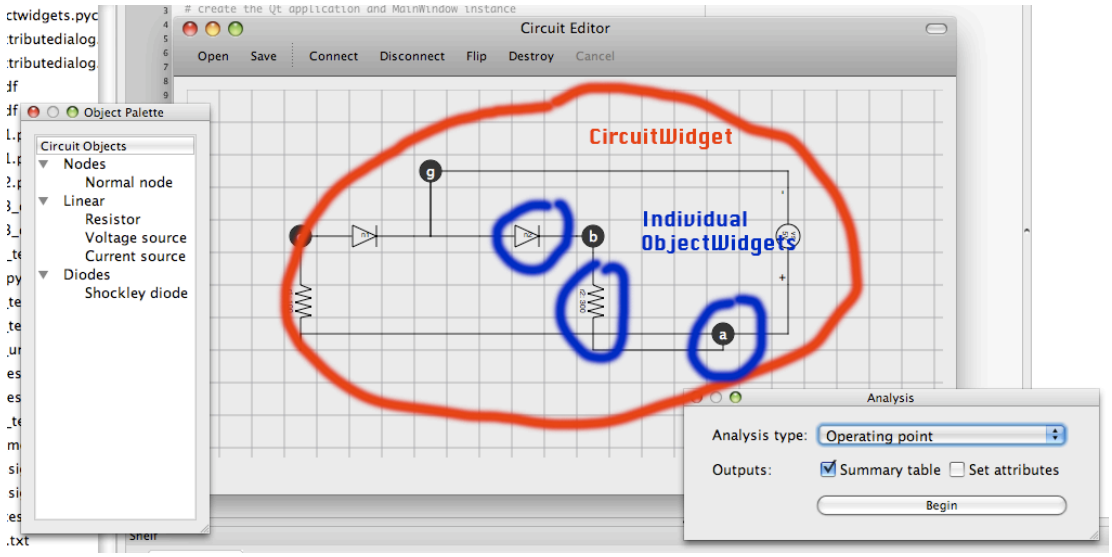

Figure 1: GUI layout on-screen

Figure 1 shows some idea of the overall hierarchy of the GUI objects. The red circled grid area is the CircuitWidget instance, while the blue-circled widgets are individual instances of ResistorWidget, ShockleyWidget and NodeWidget. All circuit component widgets subclass ObjectWidget, which provides many common services such as drag movement, editing of attributes, and connection management.

New instances of widget classes are created by dragging from the Object Palette (to the left of the figure) onto the main grid. They can then be manipulated by dragging, double-clicking to edit attributes, or through the use of click flows on the toolbar.

The GUI uses a flow-based approach to handling actions such as connecting and disconnecting objects, flipping and destroying. By making use of continuations (from the py.magic.greenlet class), the flow of each command can be interrupted to wait for a user mouse click before continuing (this is contained in circuit_gui/mainwindow.py)

Analysis is carried out from the Analysis dock, and allows the user to select from the different available analysis modes, as well as whether output should be displayed in a summary table and/or set on attributes of the widgets involved.

Deliverables assessment

In total, based on the original design proposition, the implemented functionality is a combination of List D (in total) with some elements of List E. This represents a successful outcome from the initial design.