Project Report

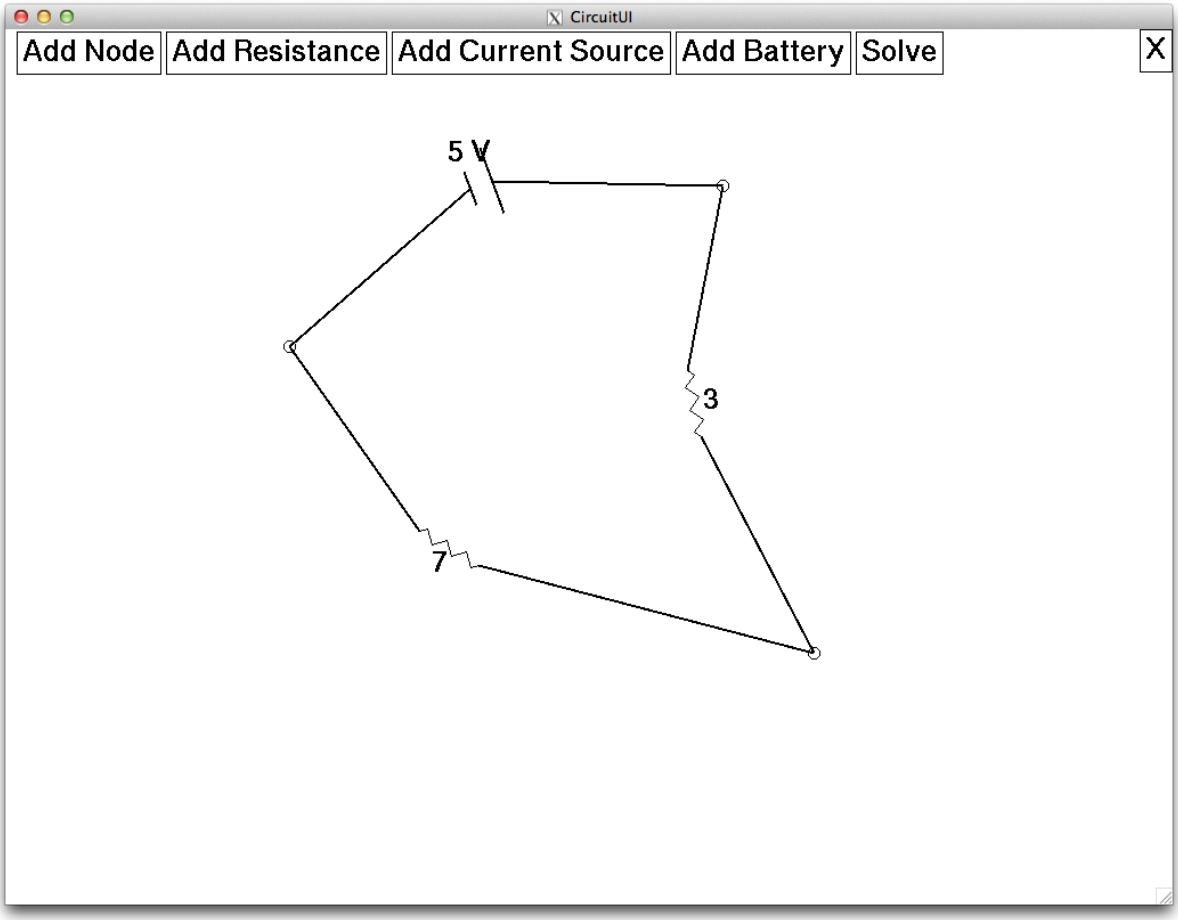
Circuit Solver

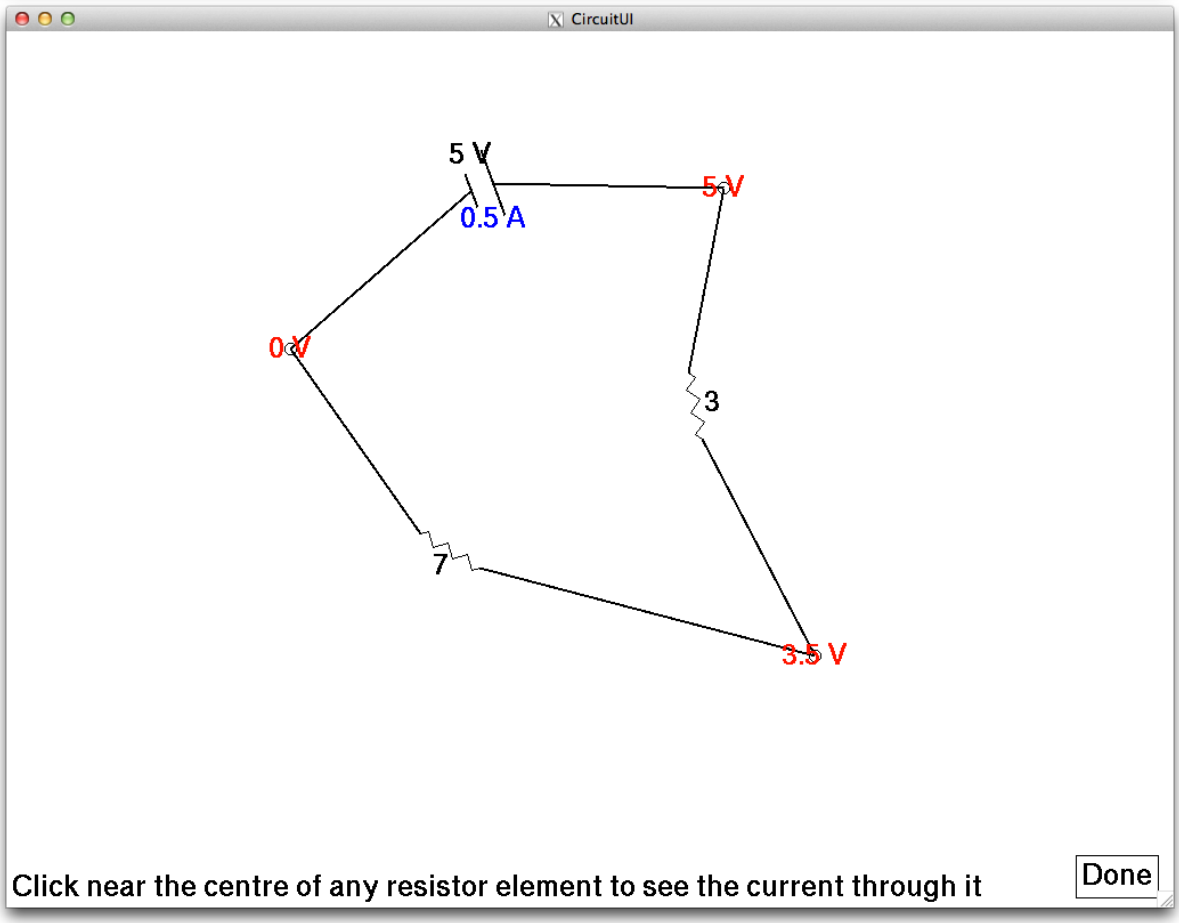
Siddhant Ranade (130260002)
Unmesh Phadke (130260006)

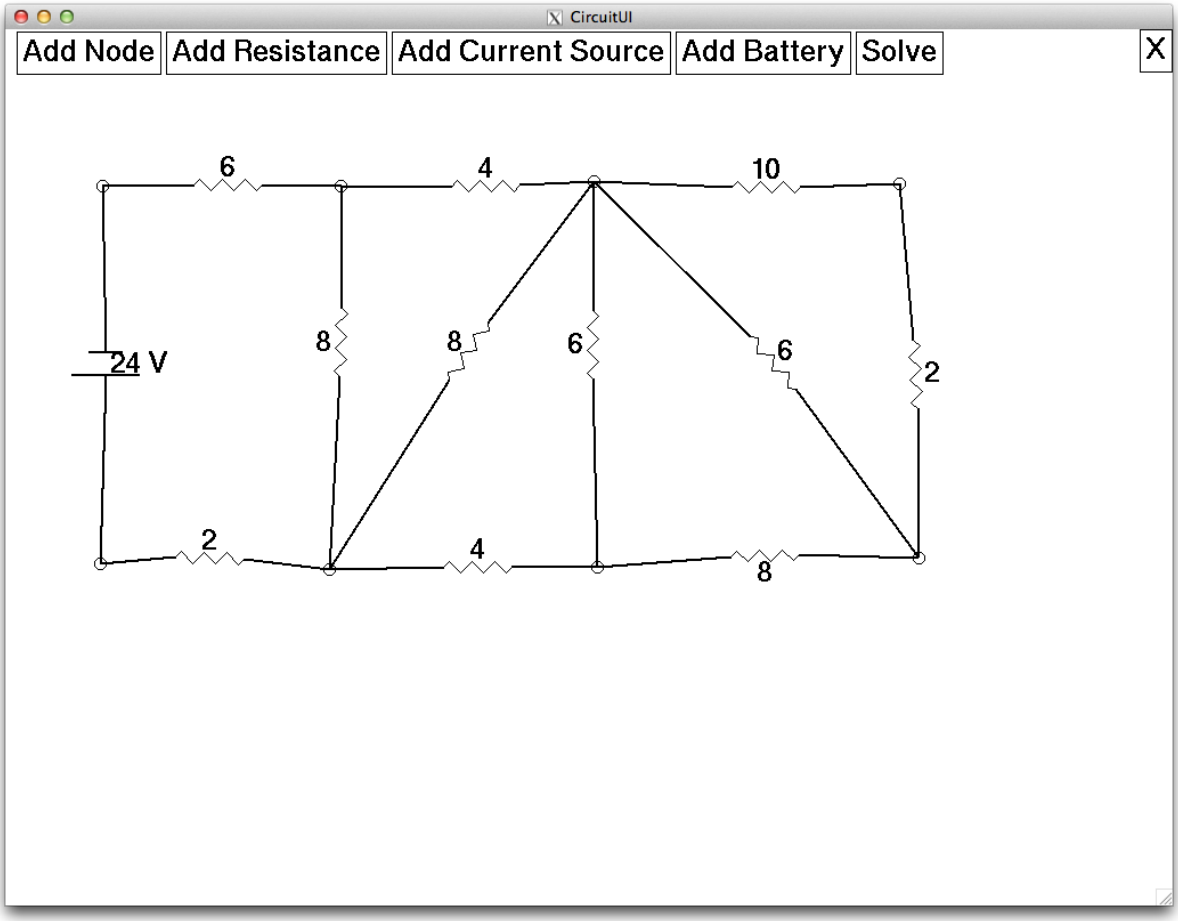
What is this? What does it do?

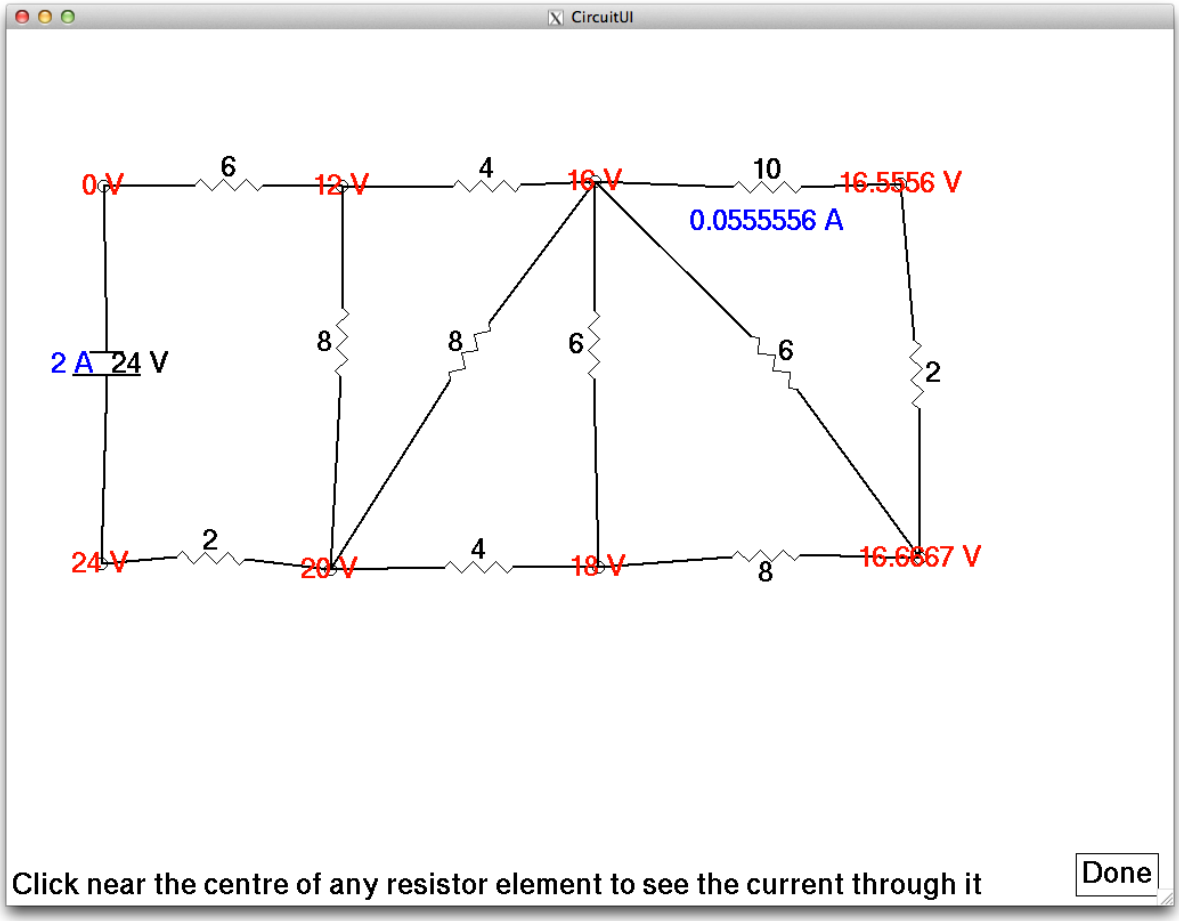
The aim of this project is to, given any DC electric circuit, solve it, i.e. find out the potentials at all the points and the currents through all the Voltage Sources (Battery/Cell).
This program supports both constant current sources and constant voltage sources.
This program takes purely graphical input (no command line) and the output also is shown on the screen.

Some Sample Inputs (and Outputs)






CircuitUI
Add Node
Add Resistance
Add Current Source
Add Battery
Solve
X
6
4
10
24 V
8
8
6
6
2
2
4
8


CircuitUI
0 V
6
12 V
4
16 V
10
16.5556 V
0.0555556 A
2 A
24 V
8
8
6
6
2
24 V
2
20 V
4
18 V
8
16.6667 V
Click near the centre of any resistor element to see the current through it
Done

How does it work?

This program uses the following classes:

- `Node`
- `Resistance`
- `Voltagesource` (Battery/Cell)
- `Currentsource`

When we start constructing the circuit, we first create Nodes. Nodes can be connected by resistors, cells, or current sources. All the nodes are stored in a vector(`std::vector`).
The other 3 classes have data members that determine to which node that element is connected. When we create these “connector” objects, they are also stored in separate vectors.

For a circuit with n nodes and m voltage sources, when the solve button is clicked, three matrices are constructed: A((n-1)+m x (n-1)+m), Z((n-1)+m x 1) and X((n-1)+m x 1).

A is of the form: $\begin{bmatrix} G & B \\ C & D \end{bmatrix}$

Where:

- G is an (n-1) x (n-1) matrix such that:

- Each element in the diagonal matrix is equal to the sum of the conductance (one over the resistance) of each element connected to the corresponding node.
- The off diagonal elements are the negative conductance of the element connected to the pair of corresponding node.

By default, Node number 0 is grounded, and there are no entries for it anywhere in A.

- B is an (n-1) x m matrix such that if the positive terminal of the i^{th} voltage source is connected to node k, then the element (i,k) in the B matrix is a 1. If the negative terminal of the i^{th} voltage source is connected to node k, then the element (i,k) in the B matrix is a -1. Otherwise, elements of the B matrix are zero.

- C is an m x (n-1) matrix, and is the transpose of B.

- D is an m x m matrix consisting of all zeroes.

The X matrix holds all the unknowns. It is of the form $\begin{bmatrix} V \\ J \end{bmatrix}$ where:

- V is an n-1 x 1 matrix such that each value corresponds to the voltage of a node in the circuit (w.r.t. ground)
- J is an m x 1 matrix with one entry for the current through each voltage source.

The Z matrix is of the form $\begin{bmatrix} i \\ e \end{bmatrix}$ where:

- i is an n-1 x 1 matrix with each element corresponding to a node (except node 0) with a value equal to the algebraic sum of current sources attached to it.
- e is an m x 1 matrix with each value equal to the value of the voltage source.

Now, $A \times X = Z$ is a well known theorem, and we solve for X by using the row transformations method[1].

After that we display all the voltages near the respective nodes.

The classes are defined in the include/RCV.h , member functions being in src/RCV.cpp

The solver ("`matrixsolve(matrix<T> A, matrix<T> B, matrix<T>& res)`") (row transformation) is in `src/solve.h`[2]

[1] We have also done this by constructing A^{-1} matrix, that program is v0.9.cpp, CircuitSolverOld

[2] The old "solver" is in src/MatrixMath.h as in there are separate functions for calculating the inverse "`inverse(matrix<T> A, matrix<T> &ans)`", multiplying "`multiply(matrix<T> first, matrix<T> second, matrix<T>& result)`", etc.

How to run it:

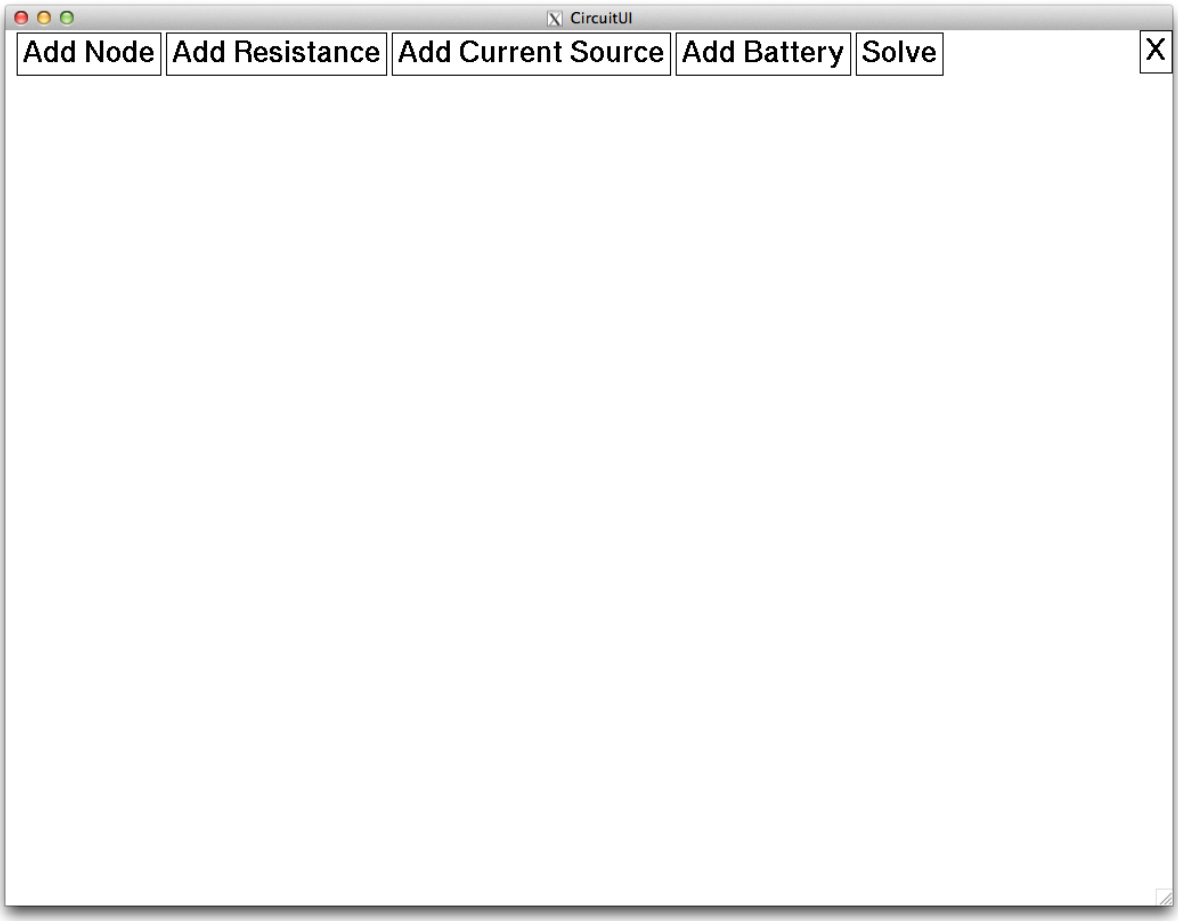
Extract the `.tar.gz` file.

From the terminal, change directory to this directory.

Then do:

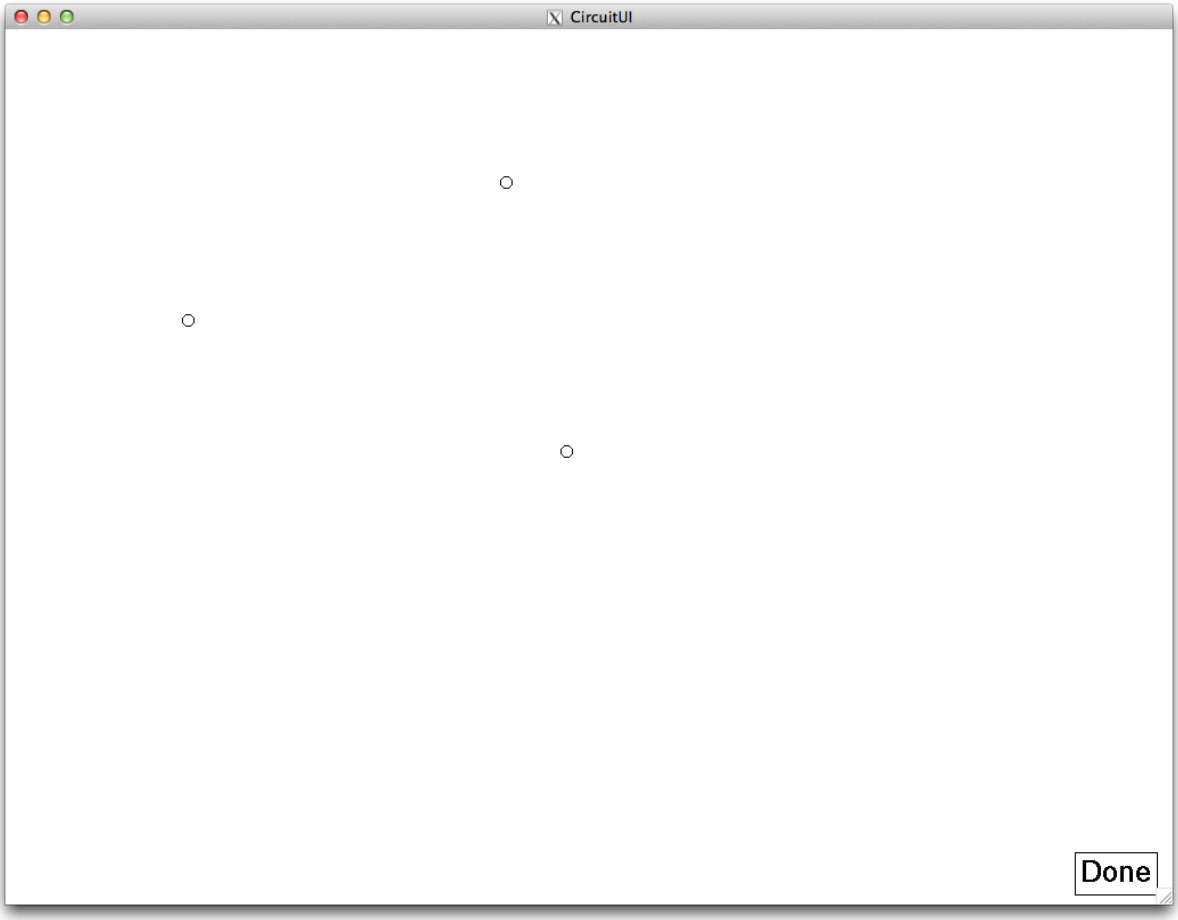
- `$./configure.sh` to compile everything.
- `$./CircuitSolver` to run the program.[3]

When the program starts, you will see this screen:



Click on the Add Node button to start adding nodes. They appear as small circles. When you're done, click the Done button.

[3] do $ ./CircuitSolverOld to run the older version, which uses a different solving algorithm.



Now add “connectors”, say resistors.
Click on the Add Resistance button.

Click a node, and then another one to join them with the resistance. If you click on the blank part, the click will be ignored. After doing that, enter the resistance value (in ohms) and press enter. (The bksp key works)
Then click on a point on the screen (not a node) to place the resistor element. (Don’t choose a point that is far away from the line joining the nodes, else the drawing will be messy.)
Continue adding more resistors the same way.
When you’re done adding, press the Done button.

In the same way, add Batteries[4] or current sources[5].

[4] The point that is clicked first is connected to the negative terminal of the battery.

[5] The point that is added first is connected to the positive terminal of the current source.

When your circuit is ready, click on the Solve button. This will display the voltages at all the nodes, and the currents supplied by all Batteries on the screen. Clicking on any resistor[6] gives the current through that resistor.

When done examining the circuit, press Done.

You can even extend the circuit, add more nodes, etc.

To quit, click on the X button in the top right corner of the screen.

References:

- Representation of circuits in memory - Introduction to Programming through C++, Abhiram Ranade, pg 359.
- An Algorithm for Modified Nodal Analysis - `http://www.swarthmore.edu/NatSci/echeeve1/Ref/mna/MNA3.html`

[6] You need to click near the centre of the resistor element to see the current through it.