DC Circuit Simulator

Juraj Pavlović

Loughborough Grammar School

2020./2021.

Content

[Analysis 4](#_Toc67337092)

[Introduction 4](#_Toc67337093)

[The end-user 4](#_Toc67337094)

[Current system investigation 5](#_Toc67337095)

[Flowcharts for existing and proposed system 7](#_Toc67337096)

[Existing programs 8](#_Toc67337097)

[Complex principles 9](#_Toc67337098)

[Components 9](#_Toc67337099)

[Adding resistors 10](#_Toc67337100)

[Measuring voltage and current in the circuit 10](#_Toc67337101)

[Kirchhoff's current law (KCL) 10](#_Toc67337102)

[Kirchhoff’s voltage law (KVL) 11](#_Toc67337103)

[Ohm’s law 11](#_Toc67337104)

[Mesh analysis 12](#_Toc67337105)

[Nodal analysis 13](#_Toc67337106)

[Incidence matrix 15](#_Toc67337107)

[Sparse matrix 16](#_Toc67337108)

[Modified nodal analysis (MNA) 17](#_Toc67337109)

[Netlists 20](#_Toc67337110)

[List of components 21](#_Toc67337111)

[Limitations 21](#_Toc67337112)

[Objectives 22](#_Toc67337113)

[References 25](#_Toc67337114)

[Design 27](#_Toc67337115)

[IPSO chart 27](#_Toc67337116)

[System hierarchy structure 29](#_Toc67337117)

[System flow chart 31](#_Toc67337118)

[Data dictionary 33](#_Toc67337119)

[Validation 36](#_Toc67337120)

[Data type check 36](#_Toc67337121)

[Presence check 37](#_Toc67337122)

[Length check 37](#_Toc67337123)

[Key algorithms 37](#_Toc67337124)

[Drag and drop system 37](#_Toc67337125)

[Creating a network of components 38](#_Toc67337126)

[Main program loop 38](#_Toc67337127)

[Parsing a file 39](#_Toc67337128)

[Filling MNA matrices 39](#_Toc67337129)

[Solving MNA system 40](#_Toc67337130)

[Data structure 41](#_Toc67337131)

[Lists 41](#_Toc67337132)

[Tuples 41](#_Toc67337133)

[Numpy Arrays 41](#_Toc67337134)

[Hash tables 41](#_Toc67337135)

[File structure 42](#_Toc67337136)

[User interface design 43](#_Toc67337137)

[Python modules 46](#_Toc67337138)

[PyGame 46](#_Toc67337139)

[Tkinter 46](#_Toc67337140)

[NumPy 47](#_Toc67337141)

[SciPy 47](#_Toc67337142)

[Security and integrity of the data 47](#_Toc67337143)

# Analysis

## Introduction

Learning about electric circuits for A – level students can be quite demanding and challenging especially to those who are not familiar with the concept. There are plenty of laws and formulas to be considered when solving a circuit question. I found circuits hard to understand without seeing how the formulas are intertwined and work in real life.

My aim with this coursework is to develop a program with a user-friendly interface that will serve as a teaching aid mainly for A – level students, but also engineering and physics students entering university.

The program will give the user the ability to set up a simple DC electric circuit using a couple of essential electronic components. Components will hold certain values, e.g., resistance for resistors, voltage for voltage source. However, these values are not constants. Users will be able to choose the values suitable for their needs.

## The end-user

Even though my specific end-user is an A – level physics student, this program could be used by anyone who wishes to develop an understanding of basic circuitry or further their knowledge in this very field. As everyone learns differently, I believe this tool will help a lot of students struggling to learn the subject using only a textbook and formulas, which sometimes can be quite confusing. Most of us are not used to seeing how resistors interact when put together in an electric circuit, so being able to visualise the problem can be of great help for those who prefer learning things (like me) practically.

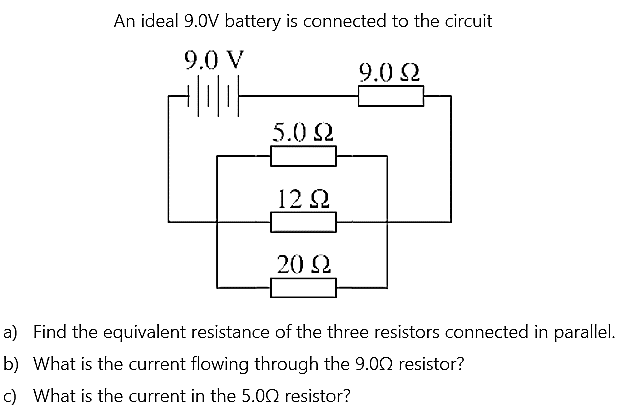
This program could also be used by teachers to give students a better explanation of the topic. When teaching, it can be very time consuming to physically set up the circuits and do the appropriate measurements. This wastes lesson time which can be spent further studying the topic. Furthermore, this program can be of great help during situations like the one in 2020. when the circumstances dictated by the pandemic forced students to study from home. One of the disadvantages of distance learning is that students cannot perform their regular physics practical work. This can have a negative effect on their overall achievement in the subject. Using this program enables students to keep up with their schoolwork even when they are not in school.

I wish users to be able to quickly learn and understand the topic of electricity, which is why the program itself will cover everything an A – level student should know.

## Current system investigation

Electricity is one of the topics in A – level physics covered in year 12. It is most commonly taught by a teacher talking through the concepts using a Power Point presentation and sections in a textbook. Sometimes the teacher would further explain the topic by demonstrating these concepts with a couple of experiments. As a revision and further practice, students are usually given some questions to be answered as their homework.

Below is an example question a student might be assigned as their homework.



To solve such a question student needs to use a couple of different principles and formulas accurately (adding resistors in parallel and in series, Kirchhoff's current law, Kirchhoff's voltage law, etc.). Below is an example of how a student might approach solving this question.

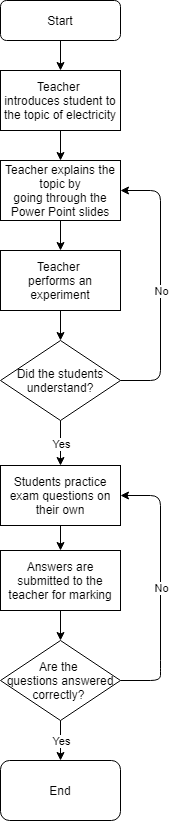
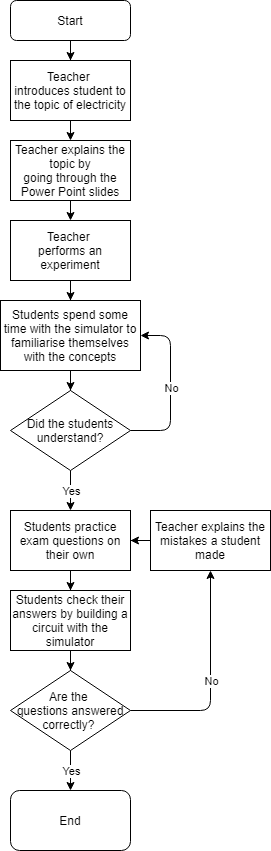
Slika na kojoj se prikazuje tekst

Opis je automatski generiran

For someone with very modest experience in solving circuit questions, this can be quite a challenge. When solving similar questions, I always wanted to be able to quickly make the circuit myself and check whether my results were correct. Unfortunately, due to the lack of electronic components and appropriate equipment for performing measurements, I was unable to do it. My main goal is to enable students to check their results using a simulator that would calculate the needed values.

### Flowcharts for existing and proposed system

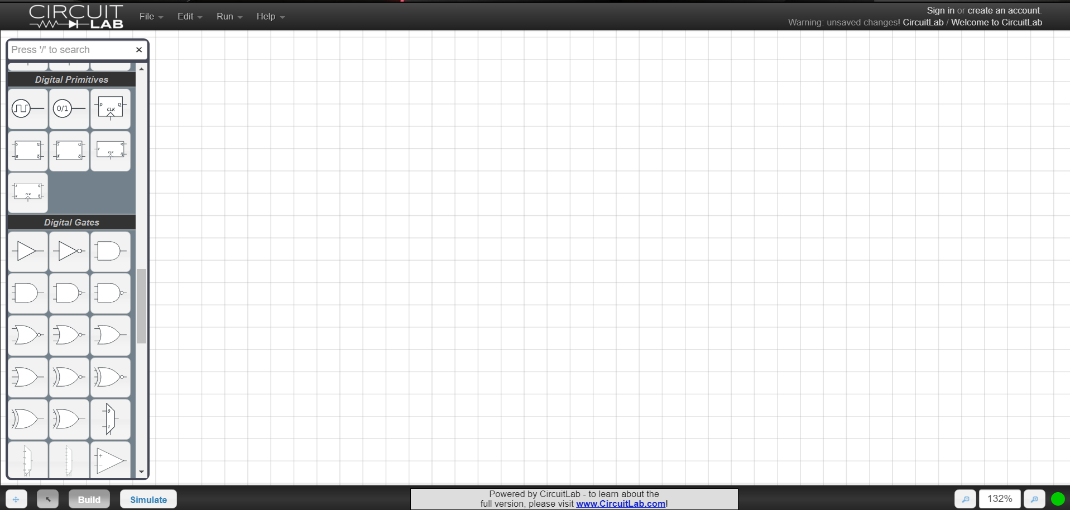
Existing system Proposed system

### Existing programs

There are already several existing programs that can be of help when solving circuit questions. One of them is “DC Circuit Builder” from the *physicsclassroom.com* webpage. With this simulator, a user can perform all the fundamental functions needed for an A – level student. A user can construct a DC circuit with resistors and an independent voltage source and then when the circuit is closed, measure a potential difference and a current flowing through any two points in the circuit. Both voltage of a voltage source and a resistance of resistors can be changed for any individual component. I will be implementing both of these options into my simulator. One component which could be added to this program is the independent current source. That way a student could control a current flowing through a specific part of the circuit.

One example of a more complex simulator is “CircuitLab”. In addition to being able to construct DC circuits, it also offers a wide range of other features. It allows users to control signals of current and voltage sources, add timers, diodes, MOSFETs, relays, and even perform Boolean algebra using digital gates. While these features might be useful for university students and engineers, such a complex program could confuse an average A – level student trying to learn basic circuitry. That is why in my program I will only focus on DC simulation. A benefit of this program over a previous one is that the user interface is much more straightforward. I will build upon this idea of a grid-based system with available components being on the left-hand side of the screen and function buttons on either the top or the bottom of the screen.



Even though all simulators might seem very different because of their user interface design, they all work using the same basic principles (for simulating DC networks at least). For a computer to solve a circuit, some way of describing the topology of the circuit is needed. A common approach would be to use netlists, which is essentially a text describing how different components are connected in a circuit. Then a circuit needs to be analysed using some kind of method. The two most common ones are mesh and nodal analysis. After that, a network of components is turned into a system of linear equations which can be solved numerically using matrixes. These principles were first used in the 1960s and the first such program open to the public was SPICE (Simulation Program with Integrated Circuit Emphasis). SPICE is an open-source analogue circuit simulator that served as a basis for other simulation programs which are today used in academia, industry and commercially. My program will be based upon one aspect of the SPICE engine which is called DC static analysis. There also exists AC analysis and transient analysis, but since these topics are not part of a A – level syllabus, I will not be implementing them.

## Complex principles

### Components

There are two types of electric components which A – level physics students should be familiar with: active and passive. Active components provide a source of energy in the circuit in the form of voltage or current, while the passive components convert or store this energy in other forms (thermal, electric or magnetic energy).

Active components:

* Voltage source
* Current source

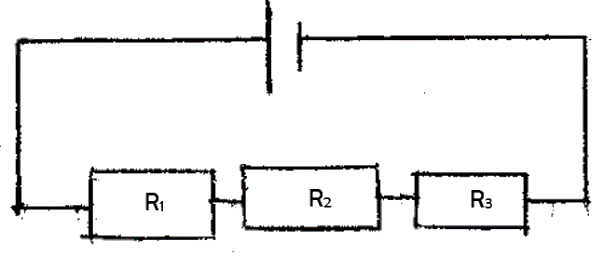
Passive components:

* Resistor
* Capacitor
* Inductor

A DC circuit analysis is a static circuit analysis which means that nothing changes over time. Once a circuit is built, currents and voltages across all the components in the system will remain constant. In such a system, capacitor and inductor can be treated as an open and closed circuit, respectively. Therefore, capacitors and inductors do not need to be included in the simulator since they will not have any impact on the final voltage and current values.

### Adding resistors

#### Series

The total resistance for two or more resistors connected in series is equal to the sum of the individual resistances. The resistance of resistors R1, R2, and R3 can be expressed as:

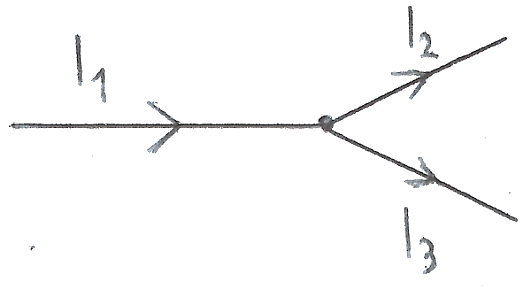
#### Parallel

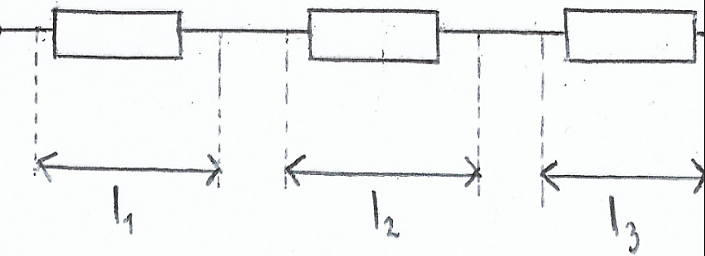
When resistors are connected in parallel different rule applies. The total resistance of resistors connected in parallel is given by:

### Measuring voltage and current in the circuit

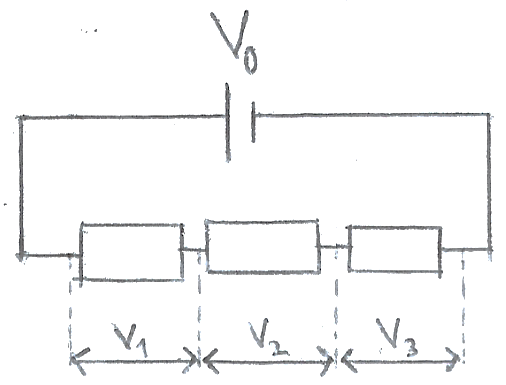
Voltage can be measured across any two points in a circuit using a voltmeter. To do the measurement correctly, the voltmeter needs to be connected in parallel. On the other hand, the current flowing through the circuit is measured by connecting the ammeter in series.

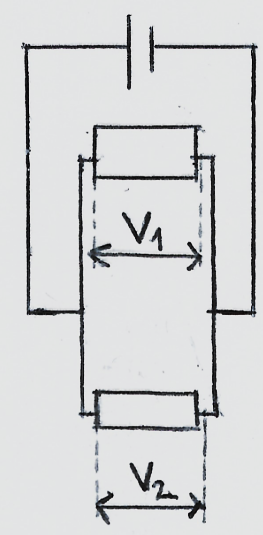
### Kirchhoff's current law (KCL)

At any node in a circuit, the total current leaving the node is equal to the total current entering the node. This is shown in the figure. Current I1 is entering the and currents I2 and I3 are exiting the node. Therefore, the following equation can be applied:

When current is entering a component, the current of the same value is leaving the same component. Also, if the current is passing through two or more components in series, each component receives the same amount of current.

### Kirchhoff’s voltage law (KVL)

The sum of all the voltages in a closed-loop circuit is equal to zero. This means that for two or more components in series, the total voltage across all the components is equal to the sum of the voltage across each component.



When two or more components are connected in parallel their voltages are the same.

### Ohm’s law

Ohm’s law describes a relationship between resistance, voltage and current. According to Ohm’s law, pd (potential difference) across a conductor is proportional to the current through it. In other words, for a component through which passes the current *I* when the pd across it is *V*, its resistance *R* is given by the equation:

Rearranging the above equation gives the following expressions for voltage and current:

Knowing two of those three quantities, a third one can be easily calculated.

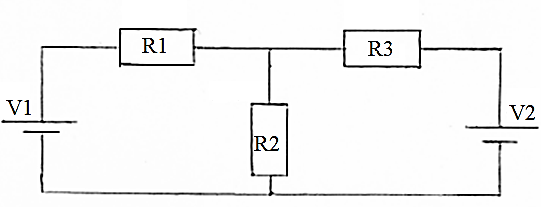
### Mesh analysis

A mesh current analysis uses simultaneous equations, KVL and Ohm’s law to calculate the unknown currents in various loops of the circuit. The steps of the mesh analysis are:

1. Identify meshes in the circuit – Mesh is a loop with no other loops inside it.
2. Assign current variables for each mesh - Inside each mesh, there is a current flowing through it. Its direction needs to be defined as either clockwise or counter clockwise.
3. Apply KVL for each mesh in the circuit – Each mesh in the circuit will have its KVL equation. By writing one KVL equation for each mesh, a system of simultaneous equations will be obtained.
4. Solve the resulting linear system for mesh currents.

An example of these steps being implemented is shown below.

Consider the following circuit:



Where:

V1 = 9 V

V2 = 5 V

R1 = 3 Ω

R2 = 2 Ω

R3 = 1 Ω

Steps:

1. The circuit has two meshes (loops) inside it. The first mesh contains a V1 voltage source, R1 and R2 resistor. The second mesh has a V2 source, R3 and R2 resistor.
2. I will assign mesh currents in a clockwise direction so that the current I1 is flowing through the first mesh and I2 through the second mesh. Since both meshes have a common R2 resistor, a current through it will not be I1 nor I2, but a difference of those two currents, I1 – I2.
3. Applying KVL results in two following simultaneous equations:
4. Plugin in the values and rearranging the equations to solve for I1 and I2 gives:

So I1 = -1 A and I2 = -7 A.

The reason both current values turned out to be negative is that their previously defined directions were wrong. I assumed both currents were flowing clockwise, but their actual direction is counter clockwise. However, the wrong assumption, in the beginning, does not affect the result as only a negative sign needs to be removed.

After applying mesh analysis and calculating the currents, other values (e.g., voltage/power across resistors) can be found using Ohm’s law.

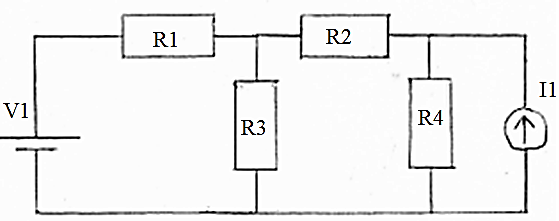
### Nodal analysis

Another basic method used for solving any electrical network is nodal analysis. While mesh analysis is mostly based on the KVL, nodal analysis is based on the KCL and is used to find different nodal voltages around a circuit. The basic principle of nodal analysis is:

1. Identify the nodes in the circuit and assign a reference node – Node is a point of connection between any two or more components. A node with the most connections is taken to be a reference node. Reference node servers as a ground node and therefore has a potential of 0 V. All the other node voltages are referenced with respect to the ground node.
2. Assign labels to all the node voltages.
3. Apply KCL for each node in the circuit, except a reference node – If there are n nodes in the circuit, there will be n – 1 equations.
4. Solve the linear system for node voltages.

An example of these steps being implemented is shown below.

Consider the following circuit:



Where:

V1 = 9 V

I1 = 2 A

R1 = 5 Ω

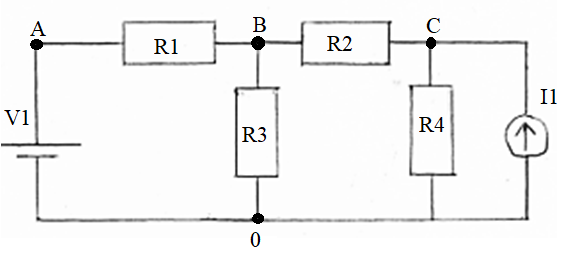
R2 = 8 Ω

R3 = 10 Ω

R4 = 10 Ω

Steps:

1. There are four nodes in the circuit, so there will be three simultaneous equations.
2. Node A connects V1 and R1, node B connects R1, R2 and R3, node C connects R2, R4 and I1, node D connects V1, R3, R4 and I1. Since node D is connected to the most components, it is taken to be a ground node and labelled 0.



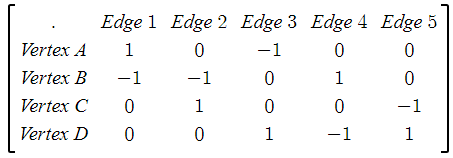
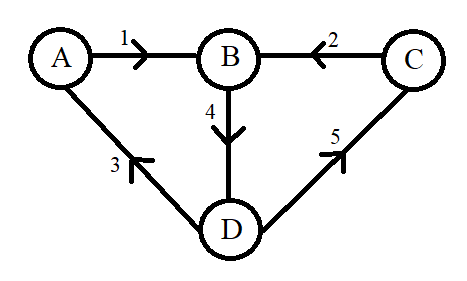
1. Applying KCL results in three following equations:
2. Plugin in the values and solving the equations for unknown nodal voltages gives VA = 9 V, VB = 5.9 V and VC = 12.2 V.

Once all the values of all nodal voltages in the circuit and the resistance of all the resistors is known, electric current across any resistors can be calculated with Ohm’s law.

Both mesh and nodal analysis are very powerful techniques for analysing a circuit. Mesh analysis is more suitable to use when there are voltage sources in the circuit, while nodal analysis is used when there are more current sources in the circuit. Since my simulator will give the user the ability to use both, I will use a combination of two techniques together with incidence and sparse matrices to form a so called “Modified Nodal Analysis”.

### Incidence matrix

“An incidence matrix is a logical matrix that shows the relationship between two classes of objects.” The first class represents vertices and the second represents branches. The matrix has one row for each vertex and one column for each branch. The entry in the matrix is 1 or -1 if a vertex and a branch are related and 0 if they are not. 1 is used when a branch current is leaving a vertex and -1 when it is entering it. Below is an example of a graph and its incidence matrix.



In my simulator, I will use incidence matrices to describe the connection of the voltage sources and so show the effect of voltage sources on the circuit. This method would be fine on its own for straightforward circuits like the ones in the previous examples. As the users will probably want to make more complex circuits, this will result in the matrix being filled with mostly zeros. To optimise it I will be using sparse matrices optimisation techniques.

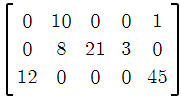
### Sparse matrix

A sparse matrix is a matrix that mostly consists of zero values. When dealing with such a matrix on a computer, it is common to optimise a computer’s processing power and memory by using specialised algorithms and data structures. Using standard matrices can be very inefficient since most of the time during a matrix operation computer would have to deal with a great number of zeros. This can be avoided with various specialised format structures which essentially ignore all the zeros and focus on the non-zero values. However, this does make accessing the individual elements more complex.

There are two types of these formats:

1. Those that use data structures to modify a sparse matrix. These are used to construct the matrices.
2. Those that use data structures which are more suitable for performing efficient operations.

Since my simulator will be using matrices to solve systems of linear equations, a format which allows arithmetic operations is required. Therefore, the second type of sparse matrices format is more suitable for my case. A CSC format is the most common for python implementation using SciPy which is why I will be using it in my program.

Compressed Sparse Row Format (CSR) uses three one dimensional arrays which contain Non\_zero\_value, Row\_pointer and Column\_index. Non\_zero\_value array is made by writing every element in the list which value is different from zero, starting from top to bottom of the first column and then moving right to the next column. Column\_index is an array of indices corresponding to the non-zero values. So Non\_zero\_value and Column\_index will always have the same length. Row\_index contains indices from the Non\_zero\_value array of every value which is the first non-zero value in each row and a total length of the Non\_zero\_value array at the end. An example of this kind of sparse matrix representation can be seen below.

Non\_zero\_value = [12, 10, 8, 21, 3, 1, 45]

Row\_pointer = [2, 0, 1, 1, 1, 0, 2]

Column\_index = [0, 1, 3, 4, 5, 7]

### Modified nodal analysis (MNA)

Mesh and nodal analysis are fairly easy to use by humans, but they are impossible for a computer to understand and solve without any modifications. MNA is a much more complex way of analysing a circuit since it often results in a larger system of equations which is stored inside matrices, but it is also much easier to implement it algorithmically on a computer. Once an MNA is applied to a circuit it will create a matrix equation in the following form:

Where matrices A and z only hold known quantities, while matrix x holds quantities (node voltages and branch currents) which need to be calculated. To Find quantities stored in x, the equation needs to be rewritten. Matrix A is moved to the right-hand side and so it becomes a transpose matrix:

Each of the three matrices are made up of several sub-matrices.

#### A Matrix

A matrix consists of 4 other matrices G, B, C, D which sizes are N \* N, N \* M, M \* N and M \* M respectively, where N is a number of nodes in the circuit and M is a number of independent voltage sources.

The G or conductance matrix is made according to two following rules:

1. 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.
2. The off diagonal elements are the negative conductance of the element connected to the pair of corresponding nodes.

The B or incidence matrix is a sparse matrix which describes a relation between passive components and each voltage source. If a component is connected to a positive terminal of a voltage source, element 1 is written to the B matrix. If a component is connected to a negative terminal, then the element in the matrix is -1. If there is no connection between the two, the element is 0.

The C matrix is just a transpose of a B matrix. A transpose of a matrix is made by interchanging the rows and the columns. So, if a B matrix is: , then the C matrix is: .

Finally, the D matrix is a matrix which is entirely composed of 0s.

#### x Matrix

The x matrix holds the unknown quantities which need to be calculated. It is formed of two smaller matrices, v and j:

The v matrix contains unknown nodal voltages, and its size is determined by the number of nodes. If there are N nodes, then there are N number of nodal voltages and therefore N number of voltage values in the v matrix.

The j matrix is made up of unknown currents flowing through the independent voltage sources in the circuit. If there are M voltage sources, then the j matrix will have an M number of current elements (one for each voltage source).

#### z Matrix

The z matrix holds the values of independent voltage and current sources in the circuit. It is also made up of two smaller matrices, i and e:

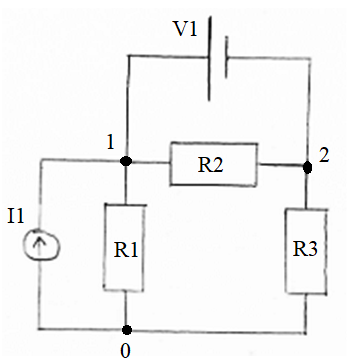
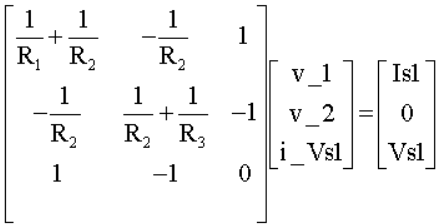
The i matrix consists of the sum of the currents flowing through the passive components. Each element in the matrix corresponds to a particular node. If there are no current sources connected to the node, the current value is 0.

The e matrix holds the values of corresponding independent voltage sources. Therefore, its size is determined by M, a number of voltage sources in the circuit.

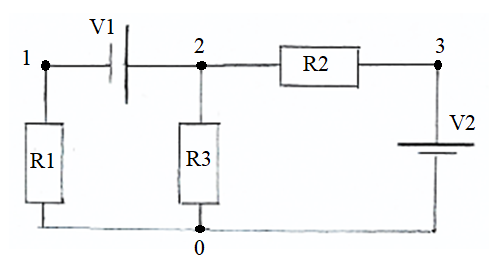
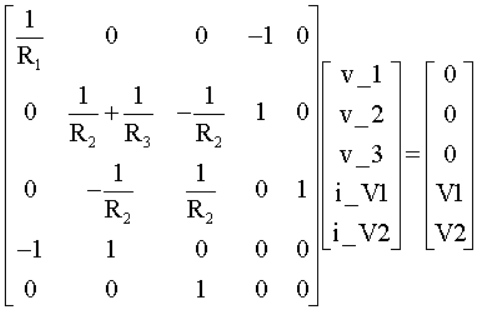
#### Examples

Below are two example circuits and their corresponding matrix equations.

Example 1

Example 2

### Netlists

A netlist is a text representation of a circuit. It consists of a list of electronic components and a list of nodes they are connected to. Its primary function is to describe connectivity in a circuit in a way that a computer can understand it. Netlists are written in lines with each line describing properties of one component. Netlists follow a certain structure. The line begins with the name of the component which tells us if it is a resistor, voltage source, or current source, then follows with two nodes which component is connected to, and in the end, a numerical value with corresponding units of the component. Netlists are also non case sensitive. Multiple spaces in a netlist file are dealt with like a single space.

Below are netlists of the previous example circuits (component values were not defined in the previous examples, so they were added subsequently):

Example 1: Example 2:

V1 1 2 9 V1 2 1 9

I1 1 0 5 V2 0 3 9

R1 1 0 10 R1 1 0 10

R2 1 2 10 R2 2 3 10

R3 0 2 10 R3 2 0 10

## List of components

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Component’s name | Alphabetic Symbol | Unit | Description | Graphical Symbol |
| **Voltage source** | V | Volt (V) | A component with two distinct terminals that provides constant voltage independent of current drawn from it. |  |
| **Current source** | I | Amp (A) | A component with two distinct terminals that supplies the same current to any load connected across its terminals. |  |
| **Resistor** | R | Ohm (Ω) | A component used to reduce current flow and drop voltage potentials by absorbing electrical energy. |  |

## Limitations

A limitation of a current system is that just by solving the questions set by teacher, some students do not develop a real understanding of how components interact in a circuit. A visual or kinetic learner may struggle to see how mathematical equations work in theory. A visual representation of a problem could greatly improve their intuition when approaching a circuit question.

However, while students might develop or deepen their understanding of a topic with a proposed solution, it will not teach them how to use the needed equations to solve a problem. As I stated before, a human and a computer solve the same circuit problem using different methods. The program will not be able to show a full walkthrough of a problem, because the program and the user would have taken different steps to arrive at the same solution. Displaying the program’s mathematical approach to a student who was just introduced to the topic will only further confuse them. Nevertheless, having the ability to construct a circuit and see the final results of it would, on its own, be of a great help to many students.

## Objectives

1. Make a GUI (Graphical User Interface) which will give the user all necessary options to construct a DC circuit.
   1. There needs to be one button for each component which will allow the user to add the component to the main tab. By clicking on the button, a chosen component should appear in the main tab.
   2. A drag and drop system needs to be created for all the components which are currently in the main tab. Once a component is added to the main tab, a user should be able to change its x and y coordinates by clicking on the component and dragging it across the screen.
   3. There needs to be a way for the user to create a network of components in the main tab. User will be able to create a connection between two components by clicking on them with mouse’s right click button. By repeating this process, a user will be able to create a DC circuit network
   4. When a component in the main tab is selected by the user, a suitable feedback should be given to the user showing which component they have selected. This will be done by changing the components appearance so that it becomes unique in respect to other components of the same type in the main tab.
   5. For a selected component, the user should be able to see its name, value, unit prefix, unit and a short description of components use in the circuit. This will be displayed in the specific part of the screen designed for this use.
   6. Each type of component should have their default value and unit prefix, but once a component is selected and its characteristics are displayed, the user should be able to alter the component’s value and unit prefix by typing their own components attributes.
   7. No value a user inputs as a component value and a unit prefix should cause a program to crash. A validation system will be used to check if correct values have been entered. If the values the user entered are incorrect or the unit prefixes do not exist, the user will be shown an appropriate message explaining that only numbers can be used as the component values or the user will be shown a list of possible unit prefixes.
2. In case the user makes a mistake while building a circuit (adds a wrong component to the main tab, makes a wrong connection between two components, etc.) they should be able to correct it by clicking on the specific buttons in the toolbar.
   1. A button which will undo the user’s latest action should be included, so that if the user notices their mistake immediately, they can quickly correct it.
   2. In case a whole circuit is built in an incorrect way, the user should be able to clear the whole main tab. This button will delete all components which are currently there.
3. Once a circuit is completed, it should be analysed and converted into a format which computer can understand.
   1. Once the analysis of the circuit is complete, a netlist in a text format should be created and stored as an external file.
   2. If a computer detects an error, analysis should be stopped, and an appropriate message should be displayed informing the user that circuit was built in an incorrect way.
4. User will have an option to save the circuit once it was analysed. After the circuit’s netlist was created and stored in the external file, user will be asked if they would like to save the circuit. If the user chooses not to save it, the file containing the circuit’s information will be deleted after appropriate calculations have been made (MNA) and final results have been displayed to the user. If the user decides to save it, file will be stored in a specific folder.
5. Once a netlist is stored in a file, MNA will be initiated and once it is complete, final results will be displayed to the user.
   1. The program should begin MNA analysis by processing the information of a netlist. This will be done by first parsing a netlist file and then mapping the nodes into indexes of the MNA matrices.
   2. Make a A matrix by combing G, B, C and D matrices together.
      1. Use the conductance values in the circuit to construct a G matrix
      2. Use sparse matrices to make a B matrix.
      3. Use a transpose of B matrix to get a C matrix.
      4. Fill a D matrix will zero values
   3. Construct an x matrix using the matrix of unknown nodal voltages and matrix of unknown currents flowing through independent voltage sources.
   4. Construct a z matrix using the matrix containing the voltages of independent voltage and matrix containing the currents of the independent current sources.
   5. Once all three main matrices are constructed, using the MNA equation, all unknown values in matrix x need to be calculated.
6. During MNA, once the matrices are constructed, they need to be stored using the sparse matrix optimisation techniques. This needs to reduce the overall time needed to perform all the calculations. To test it, I will be timing the program’s time to calculate the circuit outputs with and without the use of spare matrices.
7. The simulator should be user friendly allowing anyone with limited IT skills to use the program. Five people will be asked to use the program and rate it with a maximum of ten points and a minimum of one point. They will be students in either year 12 or 13 who are taking physics as one of their A – level subjects and preferably not taking computer science.

## References

1. <https://isaacphysics.org/questions/series_parallel_resistors?board=b730b7f6-977c-4f5e-a552-44c6d220ef89> – Picture of a circuit, *Accessed on 5.5.2020.*
2. <https://www.physicsclassroom.com/Physics-Interactives/Electric-Circuits/Circuit-Builder/Circuit-Builder-Interactive> – Example 1 of a current system, *Accessed on 10.5.2020.*
3. <https://www.circuitlab.com/> - Example 2 of a current system, *Accessed on 9.5.2020.*
4. <http://phet.colorado.edu/sims/html/circuit-construction-kit-dc/latest/circuit-construction-kit-dc_en.html> Example 3 of a current system, *Accessed on 9.5.2020.*
5. <https://electronics.stackexchange.com/questions/394068/create-circuit-simulation> – Explanation of how SPICE engine works, *Accessed on 26.6.2020.*
6. <https://en.wikipedia.org/wiki/HSPICE> – SPICE home web page, *Accessed on 27.6.2020.*
7. Breithaupt J. 2015., AQA Physics, 2nd Edition, Oxford University Press, United Kingdom
8. <https://www.tutorialspoint.com/network_theory/network_theory_mesh_analysis.htm>– Explanation of a mesh analysis 1, *Accessed on 11.10.2020.*
9. <https://www.tutorialspoint.com/network_theory/network_theory_nodal_analysis.htm> – Explanation of a nodal analysis 1, *Accessed on 11.10.2020.*
10. <https://www.electronics-tutorials.ws/dccircuits/dcp_6.html> – Explanation of a nodal analysis 2, *Accessed on 11.10.2020.*
11. <https://www.electronics-tutorials.ws/dccircuits/dcp_5.html>– Explanation of a mesh analysis 2, *Accessed on 13.10.2020.*
12. <https://steemit.com/utopian-io/@drifter1/electronic-circuit-simulation-mesh-analysis-python>– Explanation of a mesh analysis 3, *Accessed on 10.10.2020.*
13. <https://steemit.com/utopian-io/@drifter1/electronic-circuit-simulation-nodal-analysis-python>– Explanation of a nodal analysis 3, *Accessed on 11.10.2020.*
14. <http://qucs.sourceforge.net/tech/node14.html#SECTION00415200000000000000> – Explanation of a modified nodal analysis 1, *Accessed on 23.11.2020.*
15. <https://numpy.org/doc/stable/reference/generated/numpy.linalg.solve.html> – NumPy documentation for solving simultaneous equations with matrices, *Accessed on 19.11.2020.*
16. <https://steemit.com/utopian-io/@drifter1/electronic-circuit-simulation-incidence-matrix-and-modified-kirchhoff-laws-python> – Explanation of an incidence matrix 1, *Accessed on 19.11.2020.*
17. <https://en.wikipedia.org/wiki/Incidence_matrix> – Explanation of an incidence matrix 2, *Accessed on 23.11.2020.*
18. <https://en.wikipedia.org/wiki/Sparse_matrix> – Explanation of a sparse matrix 1, *Accessed on 24.11.2020.*
19. <https://docs.scipy.org/doc/scipy/reference/generated/scipy.sparse.csc_matrix.html> – SciPy documentation for CSC sparse matrix operations, *Accessed on 24.11.2020.*
20. <https://steemit.com/utopian-io/@drifter1/electronic-circuit-simulation-sparse-matrix-optimization-part-1-python> – Explanation of a sparse matrix 2, *Accessed on 24.11.2020.*
21. <https://en.wikipedia.org/wiki/Netlist> – Explanation of a netlists 1, *Accessed on 8.12.2020.*

# Design

I have included four different high-level overviews showing how different parts of my system interact with each other and how the simulator works overall. They can be seen below in the following order:

* Input, process, storage, output table
* System hierarchy chart with smaller charts explaining some aspects in more detail
* System flow chart which shows how a user would interact with the system
* Classes and attributes table

## IPSO chart

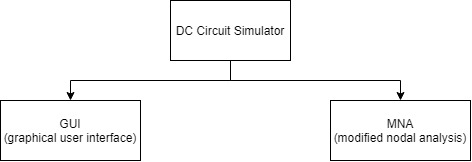
Below is a table showing systems main inputs, processes and outputs.

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **Process** | **Storage** | **Output** |
| User clicks on the button with the image of a component | A component is added to the main tab | Component is added to the sprite list of components in the main tab | Component appears in the main tab |
| User clicks on the component and, without unclicking it, moves mouse cursor around the screen | By clicking on the component, its coordinates change according to the change of x and y coordinates of the mouse | Component’s new x and y coordinates are stored | Component was ‘dragged’ by the mouse and its position changed |
| User clicks with right mouse click button on a component | Component is ready for connecting | Side of the component (left or right) is stored as a side which will make a connecting with another component | Component’s image is changed indicating to the user that the component is ready for connecting |
| User clicks with right mouse click button on another component | Two components which are ready for connecting are connected | Two components are stored in the list of connected components | A line is drawn between two connected components |
| User selects a component | Component’s image changed showing the user a component has been selected and component’s values are displayed accordingly | New image is stored as the component’s current image | Component’s appearance became slightly different with respect to other components of the same type and its characteristics (name, value, unit prefix, unit) are displayed to the user |
| User alters component’s value | Component’s value is adjusted and validated | New value is stored for the component | New value is displayed in the appropriate part of the screen |
| User alters component’s unit prefix | Component’s unit prefix is adjusted and validated | New unit prefix is stored for the component | New unit prefix is displayed in the appropriate part of the screen |
| User selects undo button | Latest change to the main tab is deleted | From the variable which stores user’s actions, latest one is deleted | Circuit layout before the latest change was made is displayed |
| User selects clear button | Every component in the main tab is deleted | List containing components is emptied | No components are displayed, and main tab is clear |
| User selects build button | Circuit is analysed, validated and the MNA starts running | Circuit is stored in netlist format in an external file | Values calculated using MNA are displayed for the user |
| User connects a measurement tool to the circuit | Values calculated using MNA are adjusted and tool’s position is validated | Adjusted MNA values are stored as a string for displaying on the tool | Appropriate values with appropriate unit prefix and unit are displayed on the tool |
| User saves the built circuit | Circuit is saved to a text file | Store circuit layout and component variable values | A message telling the user a circuit has been saved is displayed |

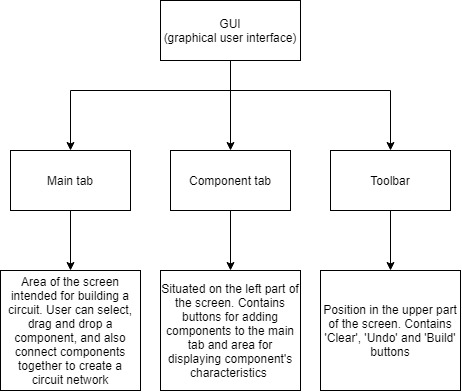
## System hierarchy structure

Below is a hierarchy chart with smaller charts explaining some aspects in more detail.

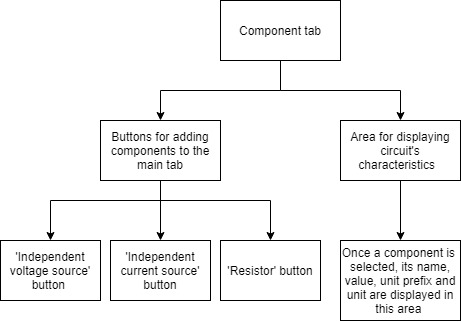
The simulator is split into two main parts: GUI and MNA. GUI gives the user an ability to construct a circuit and MNA then analyses the circuit and performs mathematical operations to obtain the final results.



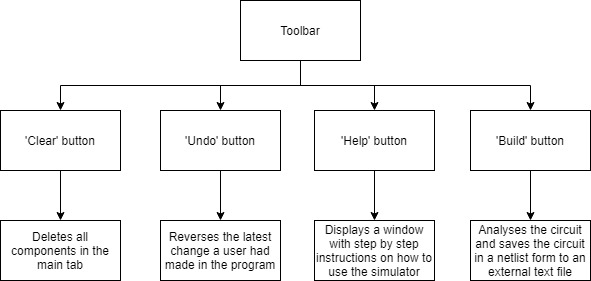
GUI is visibly split into three main parts on the screen.



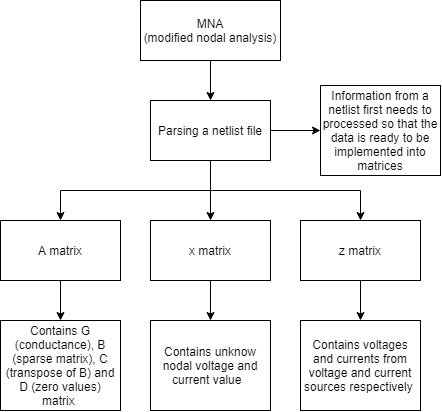
Component tab consists of two main parts both related to components which form a circuit. One part has buttons with icons of components. These are used to add components to the main tab. Other part is used to give the user more detailed information about each specific component.



Toolbar is a part of GUI with buttons which help the user during the process of building a circuit. There four buttons in the toolbar.



After building a circuit, MNA uses the netlist text file (file describing a circuit in a text format) to construct a linear equation with three matrices. Using the data from two matrices, two-dimensional A matrix and one-dimensional z matrix, a matrix with unknown values, x matrix, can be calculated. Final results are then displayed to the user.



## System flow chart

Below is a system flowchart showing all possible ways a user could interact with the simulator.



## OPP table

|  |  |  |
| --- | --- | --- |
| Class Name | | |
| Attribute | Data type | Use |
| Methods | | Use |
|  | | |
| Component | | |
| image | PNG image | Component’s image |
| rect.y | Integer | Variables which hold component's current y position. |
| rect.x | Integer | Variables which hold component's current x position. |
| id | Integer | Unique number for every component. |
| clicked | Boolean | Used for changing component's position. |
| selected | Boolean | Used for displaying components characteristics. |
| ready | Boolean | Used for connecting two components together. |
| RightSide | Boolean | Boolean showing if component's right side is selected. |
| LeftSide | Boolean | Boolean showing if component's left side is selected. |
| right\_links | List | List of connections to the right side of a component. |
| left\_links | List | List of connections to the left side of a component. |
| clicked\_count | Integer | Used to determine whether a component is selected or not. Component is only selected when it is clicked odd number of times. |
| type | String | Type of component (voltage source, current source or resistor). |
| name\_id | String | Unique name made from component's type and number of same type components in the main tab. |
| value | Float | Component's value. |
| unit\_prefix | String | Component's unit prefix. |
| unit | String | Component's unit. |
| high\_node | Integer | Node a component is connected to on its right side. |
| low\_node | Integer | Node a component is connected to on its right side. |
| display\_characteristics() | | When a component is selected, its characteristics get displayed in the component tab |
|  | | |
| button | | |
| x | Integer | Button x position |
| y | Integer | Button y position |
| width | Integer | Button width |
| height | Integer | Button height |
| colour | Touple | Button colour |
| surface | object | Button's surface |
| draw\_button() | | Draw a button to the screen |
| is\_pressed() | | Check if button is pressed. |
|  | | |
| circuit\_component | | |
| comp\_type | List | Component type |
| high\_str | String | High node obtained from the netlist |
| low\_str | String | Low string obtained from the netlist |
| high | List | Mapped high node |
| low | List | Mapped low node |
| value | Float | Component’s value |
|  | | |
| CSC | | |
| rows | List | Row index |
| colums | List | Column pointer |
| data | List | Non zero value |
| insert() | | Add new values to its attributes |
|  | | |
| MNA | | |
| file\_name | String | Name of the netlist file. |
| optimised | Boolean | Variable which determines whether optimisation is used. |
| components | List | List of components |
| hash\_table | Dictionary | Table of hashed nodes |
| voltage\_count | Integer | Number of voltage sources |
| node\_count | Integer | Number of nodes |
| matrix\_size | Integer | Determines matrix size |
| unit\_prefixes | Dictionary | Used for converting unit prefixes into floats |
| Parse\_netlist() | | Parses the text file. |
| Nodes() | | Maps string nodes to integer nodes |
| A\_matrix() | | Constructs A matrix |
| z\_matrix() | | Constructs z matrix |
| Optimised\_A\_matrix() | | Constructs optimised A matrix |
| x\_matrix() | | Solves linear matrix equation |
| print\_results() | | Displays results to the user |

## Data dictionary

Below is a data dictionary with all key variables used in the program. Each variable is explained in detail.

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Data Type** | **Example** | **Description** |
| WIDTH | Integer | 1280 | Width of the user’s monitor which is collected at the start of the program |
| HEIGHT | Integer | 720 | Height of the user’s monitor which is collected at the start of the program |
| done | Boolean | False | Used in the main loop of the program. Once the user decides to exit the program it become true and the program stops running |
| FPS | Integer | 60 | Value which determines the number of frames per second the GUI is updated |
| x\_button\_pos | Integer | 10 | x coordinates of a button |
| y\_button\_pos | Integer | 20 | y coordinates of a button |
| width | Integer | 150 | Width of a button |
| height | Integer | 110 | Height of a button |
| colour | Tuple | (0, 0, 0) | RGB value of a colour stored as a tuple filled with integers |
| name | String | V | Components name |
| component\_id | Integer | 1 | Index number given to a component when added to the main tab |
| high\_node | Integer | 5 | First of two nodes a component is connected to |
| low\_node | Integer | 0 | Second of two nodes a component is connected to |
| unit\_prefix | String | k | Based on the metric system, a unit prefix precedes the component’s unit to indicate a multiple or submultiple of a unit |
| unit | String | Ω | Component’s unit value. Each type of component has different unit |
| selected | Boolean | True | When a component is selected, its characteristics are shown in component tab |
| Clicked | Boolean | True | While a component is clicked, its x and y coordinates will be the same as the coordinates of the mouse |
| image | PNG file |  | Image of the component which a user will be able to see on the screen |
| y\_component\_pos | Integer | 344 | y coordinates of a component |
| x\_component\_pos | Integer | 561 | x coordinates of a component |
| RightSide | Boolean | False | Variable which becomes true if a right side of a component has been clicked with the right click mouse button |
| LeftSide | Boolean | False | Variable which becomes true if a left side of a component has been clicked with the right click mouse button |
| links | List | [1, 2, 1, 5] | Tuple which stores all links a one component has |
| right\_links | List | [1, 5] | Tuple which stores all links a one component has on its right side |
| left\_links | List | [1, 2] | Tuple which stores all links a one component has on its left side |
| netlist | Text file |  | A text file which stores netlist information |
| A | Two-dimensional array | [[0.25, -0.25, 1.0], [-0.25, 0.75, 0.0], [1.0, 0.0, 0.0]] | Matrix containing G, B, C and D submatrices |
| z | One dimensional array | [0.0, 1.0, 9.0] | Matrix with values of voltage and current sources |
| x | One dimensional array | [0.3, 5.667, 0.1367] | Matrix with values calculated using A and z matrix |
| allowed\_unit\_prefixes | Dictionary | {'f': 'e-15', 'p': 'e-12', 'n': 'e-9', 'u': 'e-6', 'm': 'e-3', 'k': 'e3',  'meg': 'e6', 'g': 'e9', 't': 'e12'} | A list of all letters which are allowed to be used as a unit prefix |

## Validation

Every input that the user enters into the program has to be validated to prevent them from breaking the program. The best way to prevent this is to use buttons in a GUI so that the user can only input values accepted by the program. I tried to design my program mostly based on this idea. Therefore, validations will only be required for only two variables: value of the component and component’s unit prefix.

### Data type check

A data type check makes sure the user’s input is the desired data type. This validation is required for both variables. Component’s value needs to be a float since it will later be used in mathematical calculations. Unit prefix is expressed as letter so it will have to be stored as a string. To check if a variable is a string command .isalpha() can be used.

Function DataTypeCheck(unit\_prefix):

if unit\_prefix.isalpha() then

Valid = True

else

Valid = False

end if

end Function

### Presence check

When the user tries to enter a input, to prevent from crashing, program needs to check if the user has actually entered something or left their input blank.

Function PrsesenceCheck(value):

if len(str(value)) == 0 then

Valid = False

else

for x in value

if x != “ “ then

Valid = True

end if

end for

end if

end Function

### Length check

Length check ensures that a variable has a certain number of characters. User should be prevented from entering values with a huge number of characters. An example below checks if the length of a unit prefix entered by user is one (a single character).

Function DataLengthCheck(unit\_prefix):

if 1 < len(unit\_prefix.) < 1 then

Valid = False

else

Valid = True

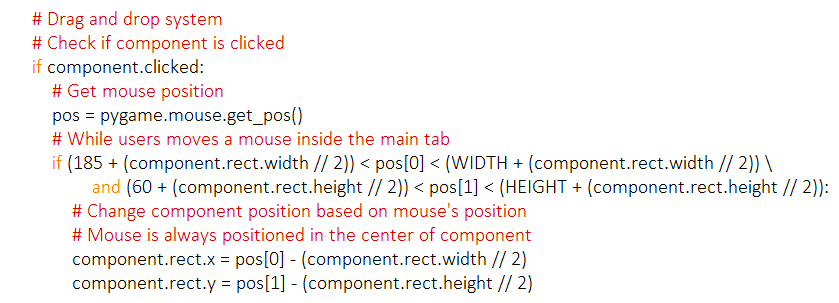
end if

end Function

## Key algorithms

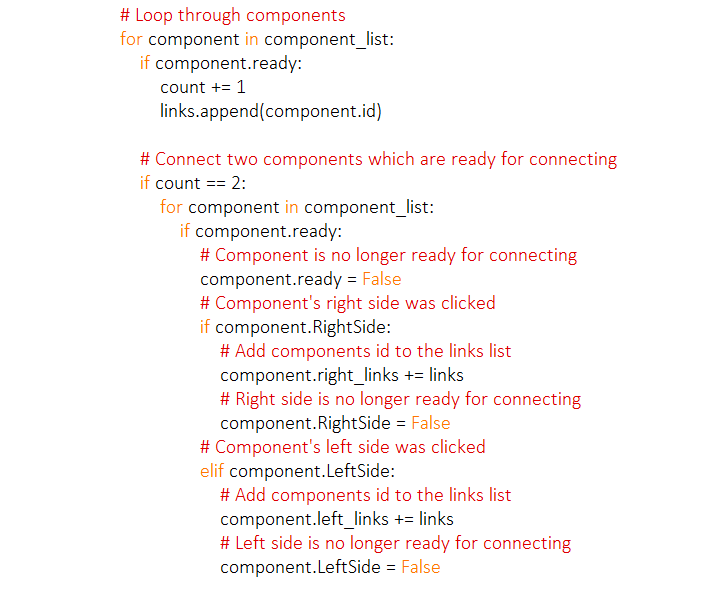
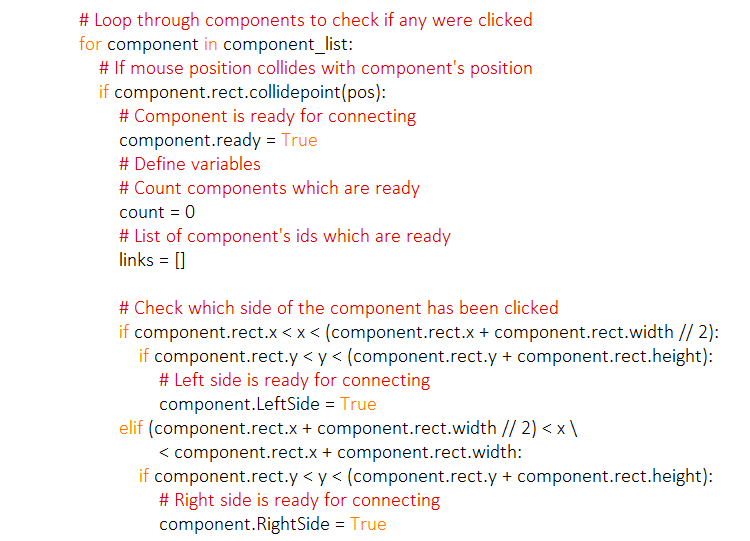
### Drag and drop system

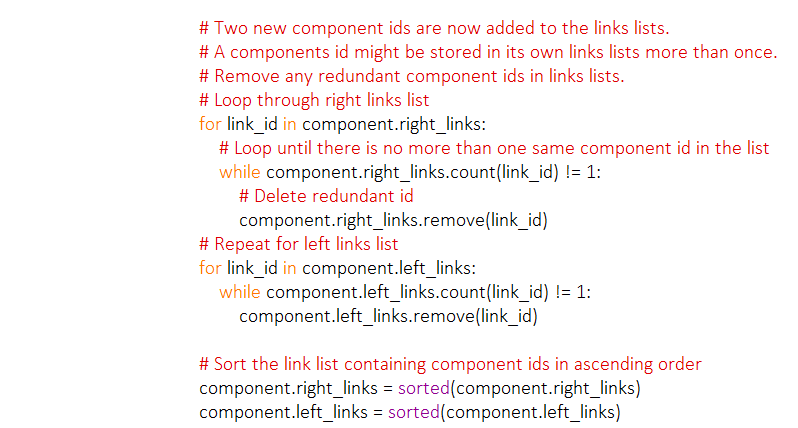
When a computer detects that mouse has been clicked, it will check if current mouse position corresponds to position of any of the components. If that is the case, component gets marked as ready. As long a component is ready its position will be the same as mouse’s position. Once a button on the mouse is released, component is no longer ready and its position stops changing.

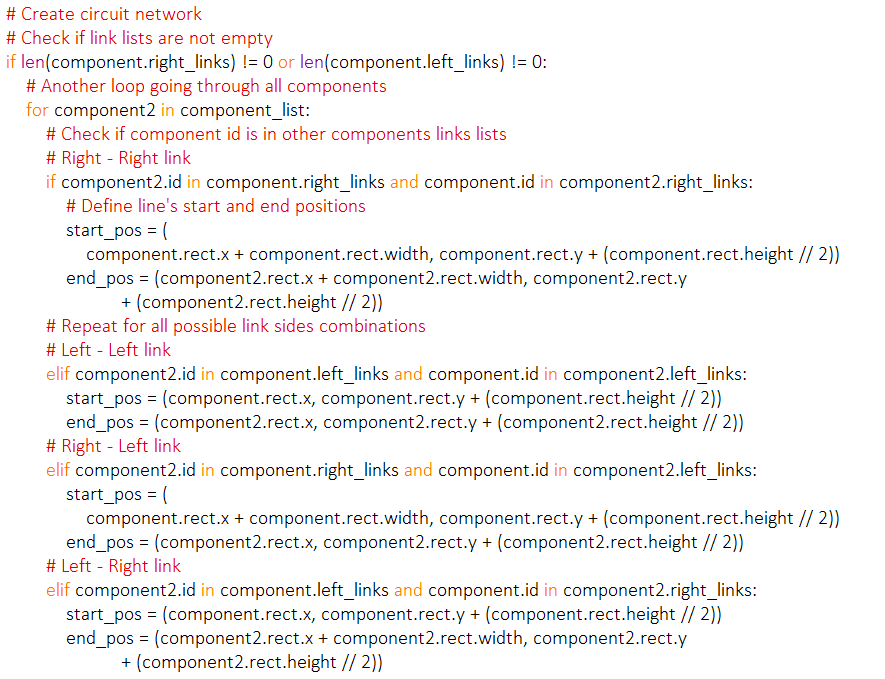


### Creating a network of components

If a component at some point appears to have the same position as the mouse does, component gets marked as ready and it gets added to the list of links. As long as another component is not marked as ready, nothing will happen. Once two components are ready they both get stored in the list of links between components.

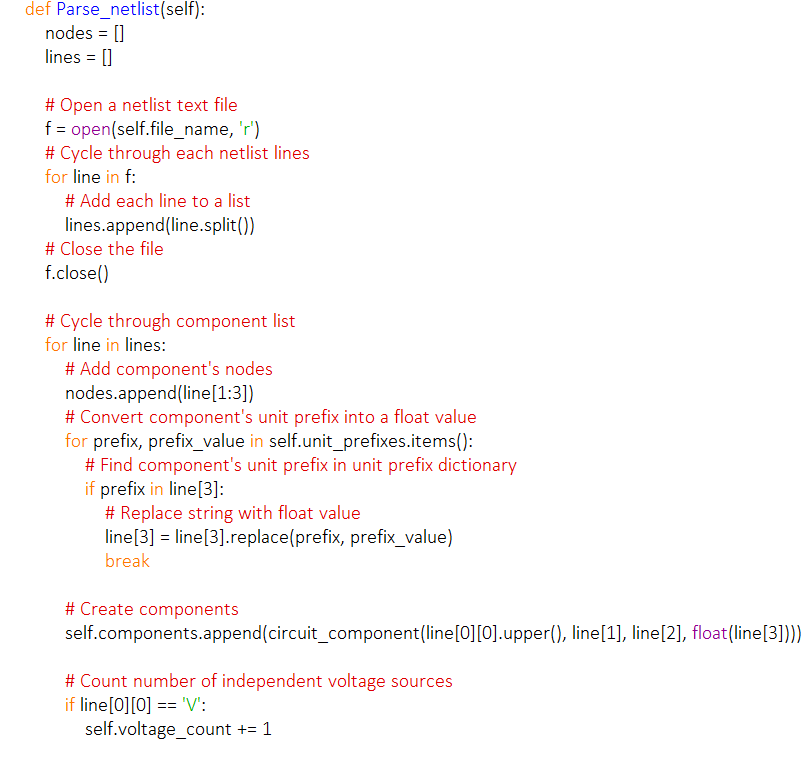






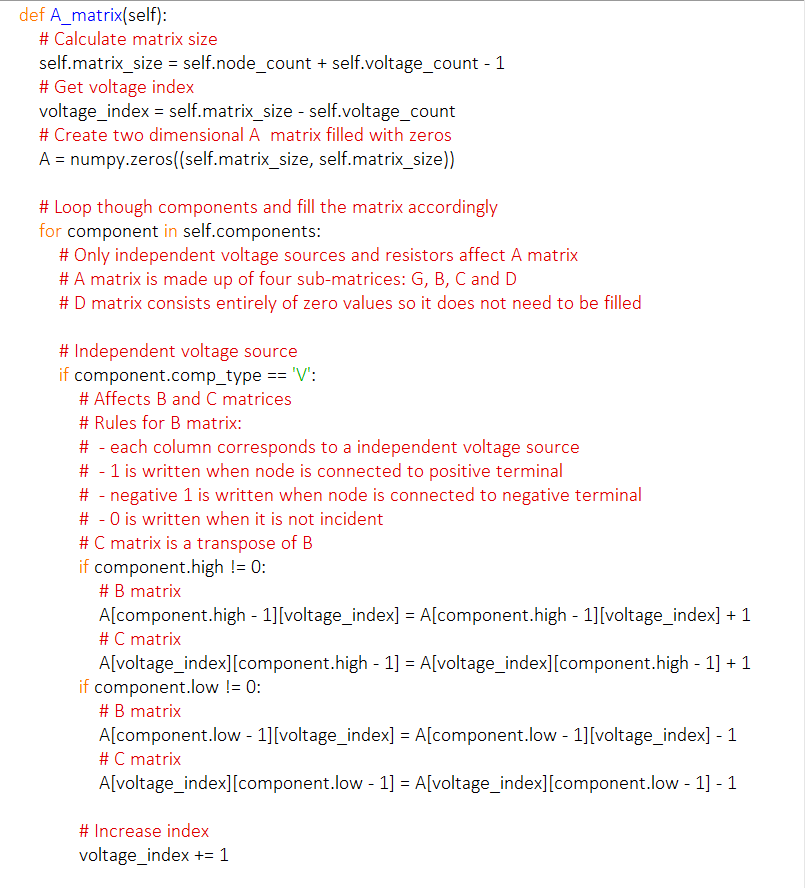
### Parsing a file

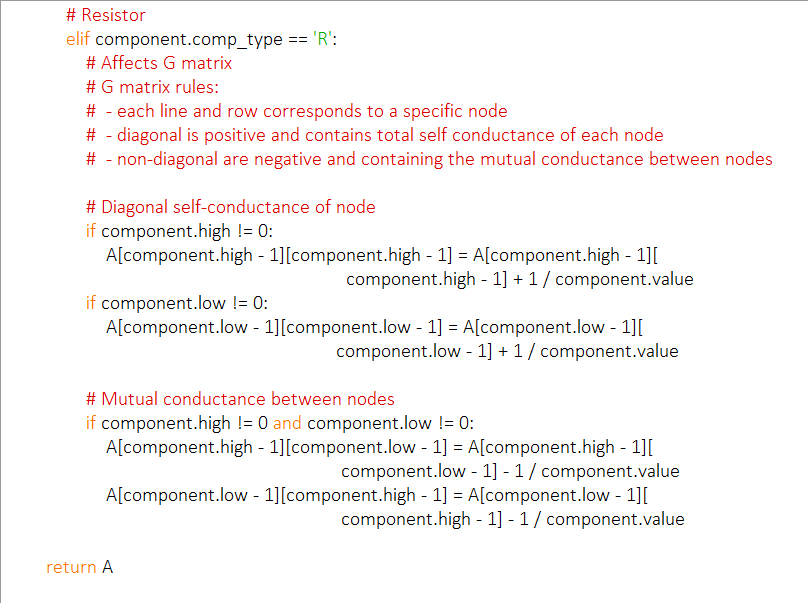
For a computer to understand what each part of the netlist means, the netlist file needs to be parsed. Computer loops through each line of the netlist and divides it into four parts: component type, high node, low node and value. This can be done easily using the .split() function. To make sure the netlist is non case sensitive and that every string value gets recorded as capital letter, .upper() function is used. The value will be stored as a float using .float() since it is going to be used for mathematical calculations.



### Construct A matrix

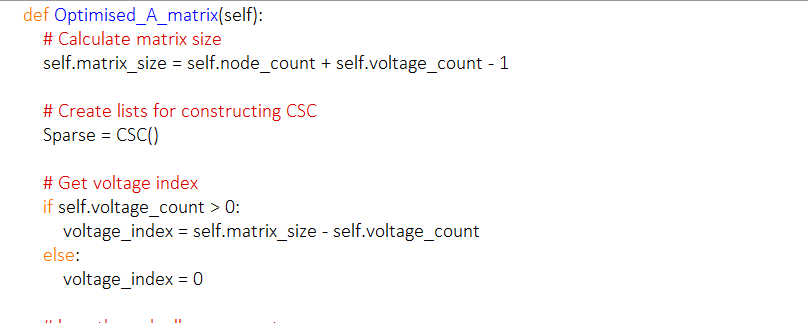
Once all the necessary information has been obtained (component type, nodes, component’s value, size of the matrix, number of independent voltage sources), MNA matrices can be made. A for loop runs through each component and based on its type and nodes (since dose two variables dictate how each matrix is made), fills the matrix.

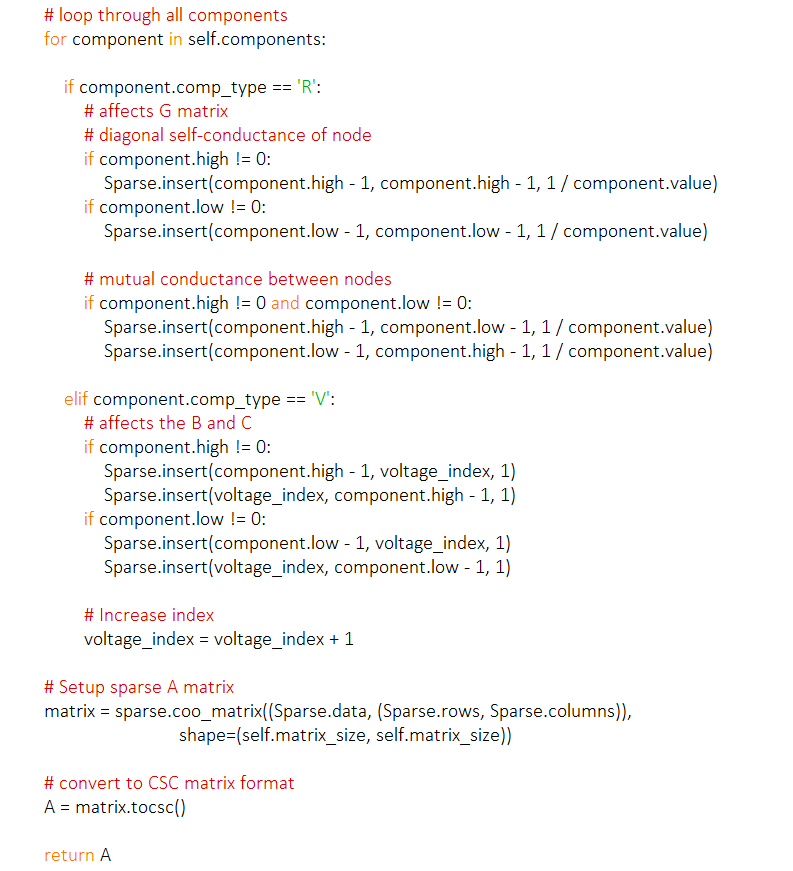




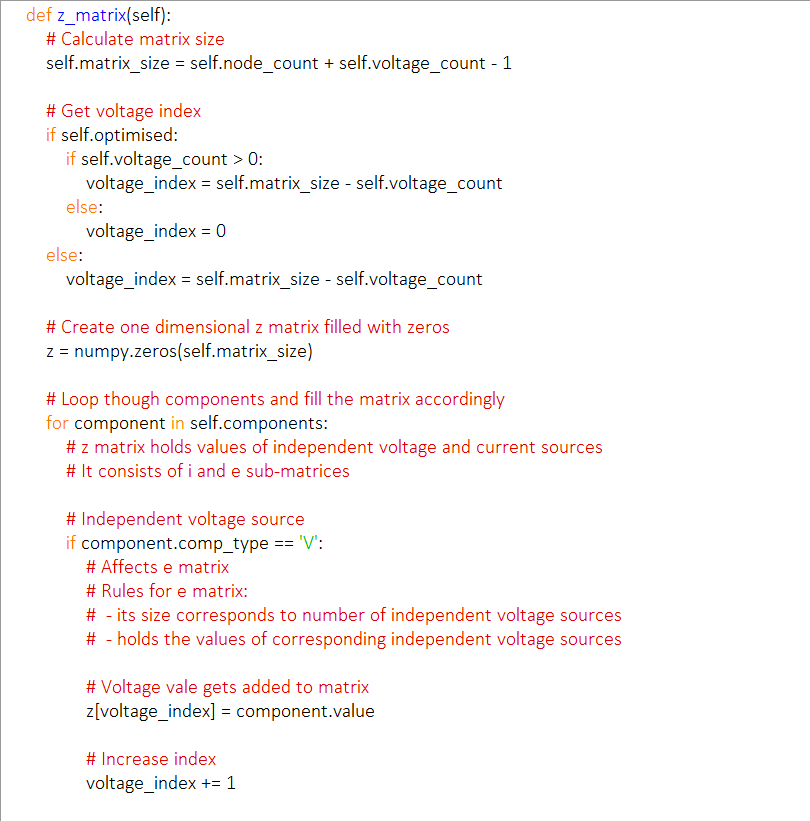
### Optimised A matrix

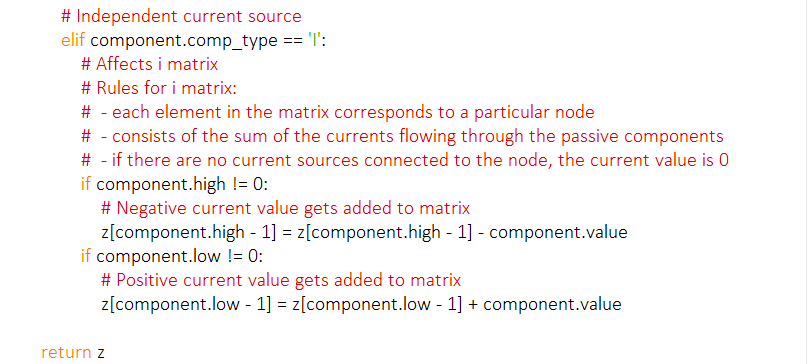
Since A matrix is a sparse matrix, it can be optimised using the sparse matrix techniques. A matrix will be represented by three string: row index, column pointer and value. Using these three lists a CSC matrix can be constructed. This way mathematical calculations can still be done with A matrix, but now it does not store any zero values.





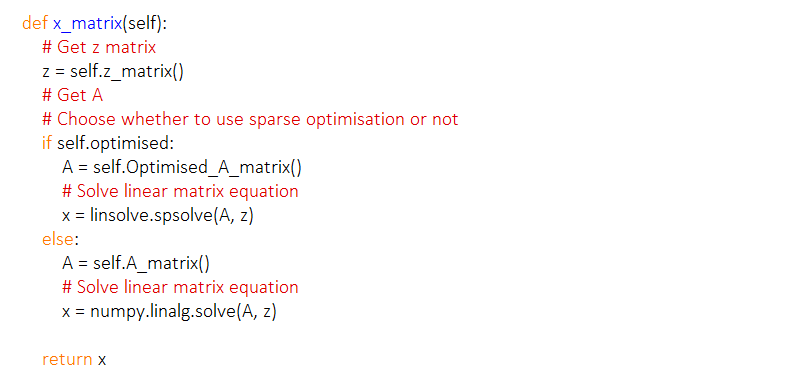
### z matrix





### x matrix

Once the MNA matrix equation is obtained, it can easily be solved using the NumPy’s linalg.solve function or linsolve.spsolve() when an optimised A matrix is used.



## Data structure

### Lists

Lists will be used in my program to store all types of data. The use of lists in my program will allow me to join multiple pieces of data (like several strings) into one list. A list is a dynamic data structure, which means that a new item can be added at any time. A new item can be added to the list using the .append built in python function. Existing items in the list can also be replaced with new data by referencing the index of the item to be replaced in the list.

### Tuples

A tuple will also be used to store some data such as the colour of an object to be displayed on screen as 3 separate integer elements between 0-255 to indicate the RGB (red, green, blue) value. A tuple is very similar to a list data structure. The only difference is that a tuple is a static data structure and all the values within the tuple cannot be changed once they are assigned.

### Numpy Arrays

Arrays will be used as an implementation of a mathematical matrix, holding multiple values in table like structure (with rows and columns). This will allow me to perform complex mathematical calculations, such as solving the system of linear equations and calculating the unknown values using linalg.solve() function.

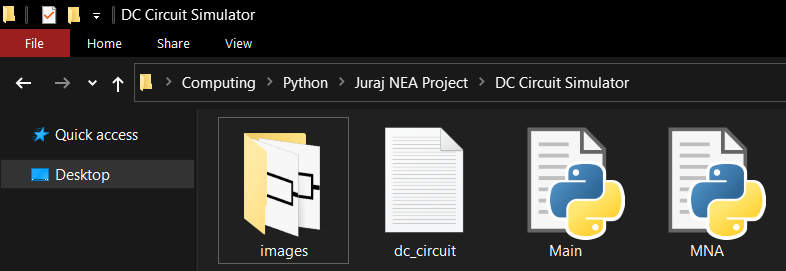
### Hash tables

Hash tables are related to a list/dictionary data type. The difference is that the items in hash tables are stored using mathematical calculation based on key and value stored.

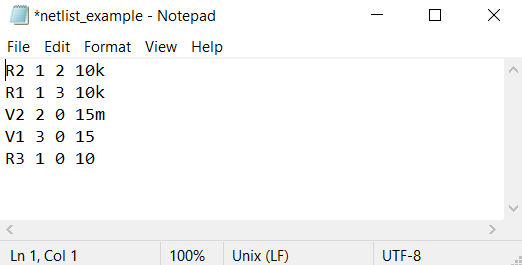
In the MNA system’s matrices each row and column correspond to a specific node, but in netlists these nodes are stored as strings. In order to use netlist’s string nodes, these strings need to mapped into specific integer. Since the computer does not know where exactly each node occurs in the netlist beforehand, these string-integer relations will have to be stored in a hash table.

## File structure

All files required to run the simulator will be stored in a single folder to allow each module to import from others. The program itself will consist of two python files with addition to extra text or image files. Python files will be called ‘Main’ and ‘MNA’. The Main.py file will include the main loop of the program and the instructions on how to make the programs GUI. Running this file will start the program. MNA.py will include all functions needed to carry out complex mathematical operations of the program. Images used in the GUI are all stored in one folder called ‘images’.



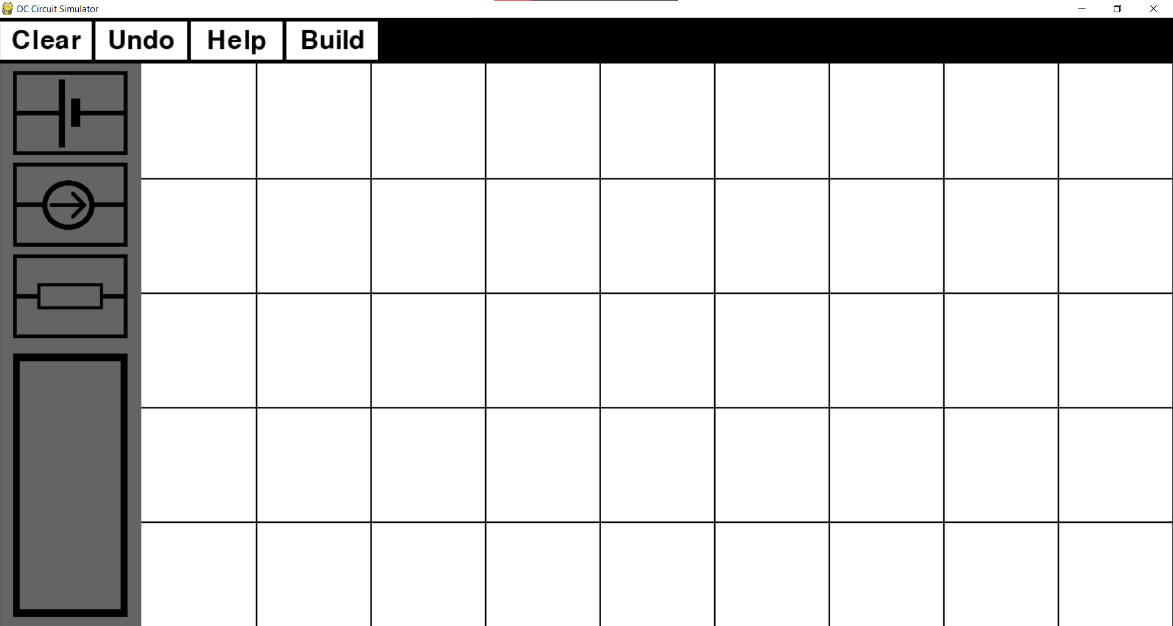
When a user decides to save a circuit layout, a new text file will be created which will contain the circuit’s netlist. An example of how this file will look like is below.



In a netlist, each row begins with a letter which is determined by the component’s name. After a letter comes component’s id. Next two numbers tell us which two nodes a component is connected to. Last number in a row represents a component’s value. A component’s value can also have a unit prefix, but it is not necessary.

## User interface design

Below is the picture of the system’s design which will be visible to the user upon opening the program. The GUI consists of three main parts: main tab, components tab and toolbar.

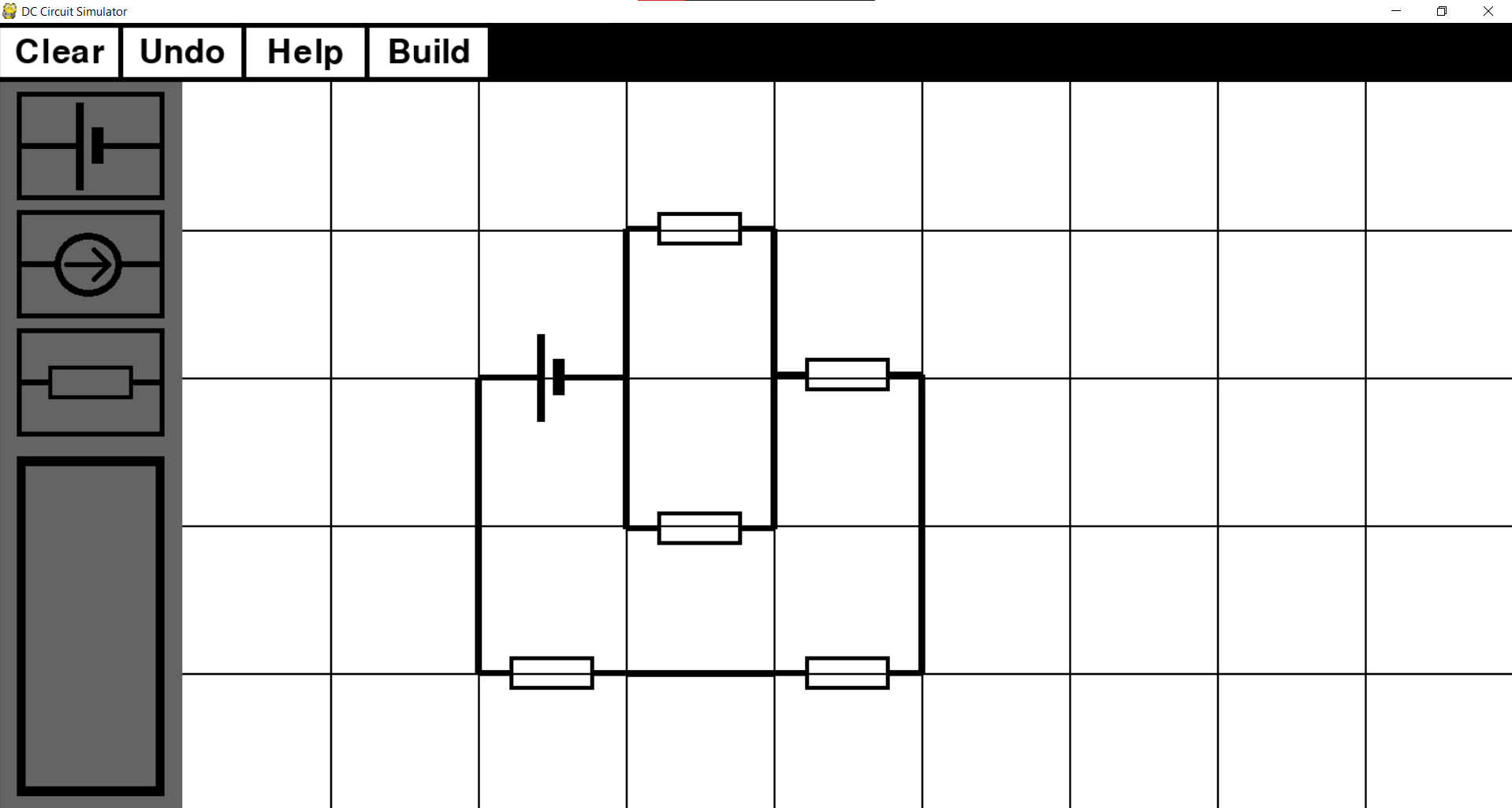


Toolbar

Component’s tab

Main tab

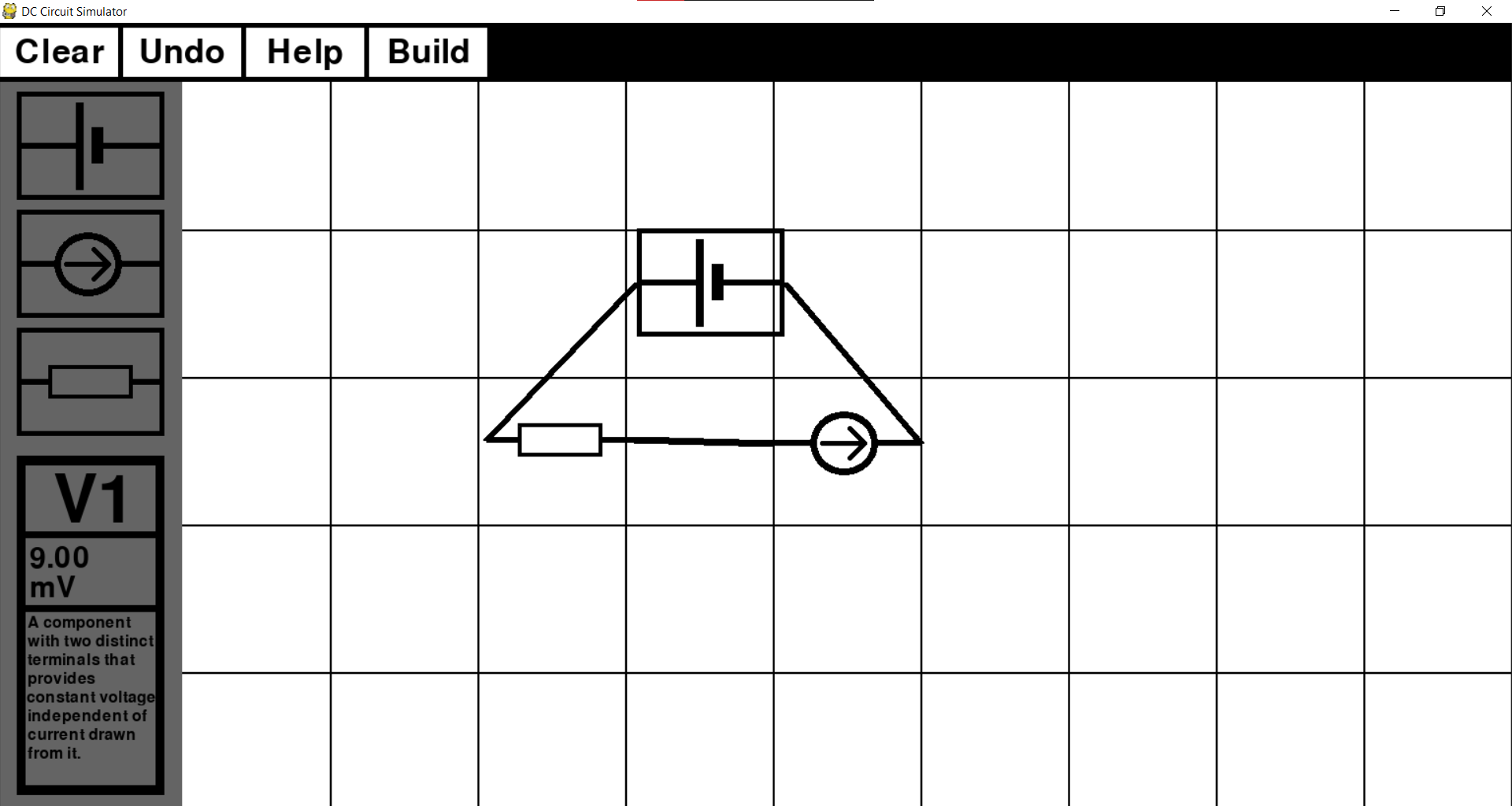
Main tab is a place on the screen where user can add components to, change their position, and connect them into a network. Below is a picture showing a GUI with a completed circuit in the main tab. The circuit consists of independent voltage source and resistors connected both in series and parallel.



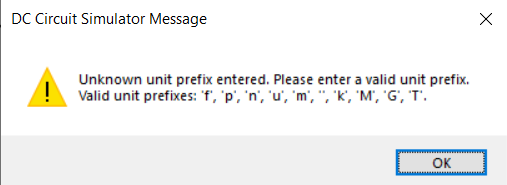
Toolbar includes four buttons on top of the screen:

* ‘Clear’ button deletes all components which are currently in the main tab (‘clears’ the screen)
* ‘Undo’ button deletes only the last component that was added to the main tab as well as all of its connections to other components.
* ‘Help’ button open a new Tkinter window with detailed instructions on how to use the simulator.
* ‘Build’ button can only be used once the circuit has been completed. It analyses the circuit, creates a netlist text file and initialises the MNA calculations.

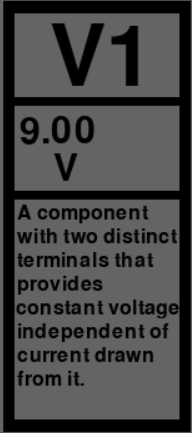
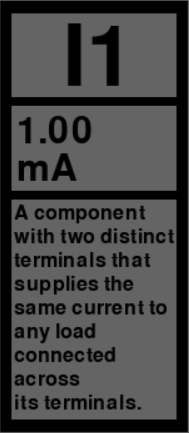
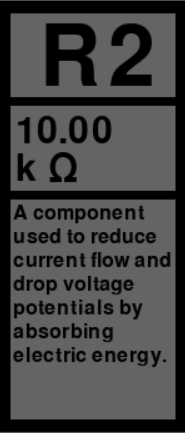
A grey area on the left side of the screen is where user can add components to the main tab by clicking on their icons. User can also see component’s characteristics which will appear in the grey rectangle with black sides. Below is an example of what a user will see if they select a component in the main tab.



In this case, an independent voltage source was selected. When selected, a component’s icon in the main tab gets framed by black rectangle, showing the user which component they have selected. In the top of the grey rectangle, a user can see components unique name, which was assigned to the component once it was added to the main tab. Component’s name consists of a letter, which determines their component type, and a unique component id. Middle part of the rectangle is where user can see and change the component’s value, unit prefix and unit (unit cannot be changed). By clicking on the component’s value or unit prefix, a black line appears on the last digit, showing the user that they can now change it. User input gets validated and new data is stored for that component. In case a user inputs invalid data, e.g., a number is inputted as unit prefix, an appropriate massage is displayed to the user (example below).



Bottom part of the grey rectangle displays a short description of the component that was selected, so a user can quickly remind themselves how that specific components interacts with other components, what affect it has on the rest of the circuit and what it is used for. Below are three pictures showing the grey rectangle for all three types of components included in my simulator.

**  **

Below is a list of explanations, where I justified some my design choices in relation to my chosen audience (A Level physics student).

### Visibility

User interface needs to be self-explanatory, so upon opening a program, the user should be able to know how to use it. For this reason, I decided to use the universal component icons which every physics student should be familiar with since those are taught in lesson and are used when sketching a circuit on a piece of paper. Other option was to use a real life pictures of these components, but since appearances of these components in real life can vary, it is impossible to find one universal picture for a component which every student will recognise instantly.

Each button in the simulator has its name written in the middle of it. Button’s name gives the user a good idea what the button’s function is. To avoid uncertainty, gaps between buttons are not too large, but at the same time, buttons are also not too close to each other. That way a user can clearly see each button’s start and end positions.

### Affordance

To give the user a better idea of what to do upon opening a program, the simulator is divided into three different parts: components tab, toolbar and main tab. Component’s tab contains icons of each component framed in black rectangle, which affords the user to press it. Buttons in the toolbar are painted in the inverse colour of their background. Whole toolbar is black, while the buttons inside it are white, indicating that upon pressing them, something will happen in the program. Relative size of the main tab to the other parts of the simulator and the lack of buttons or any other icons in the main tab is telling the user that this is where they can add circuit components, move them around and connect them into a circuit.

### Feedback

A feedback will be displayed to the user in form of a message box. This will be done when the user has inputted an invalid value into the program, when a circuit was built in such a way that it will cause the program to crash or to show the user results of applying MNA on the built circuit. At any time during the construction of a circuit the user will have a clear understanding of what the program is doing behind the GUI.

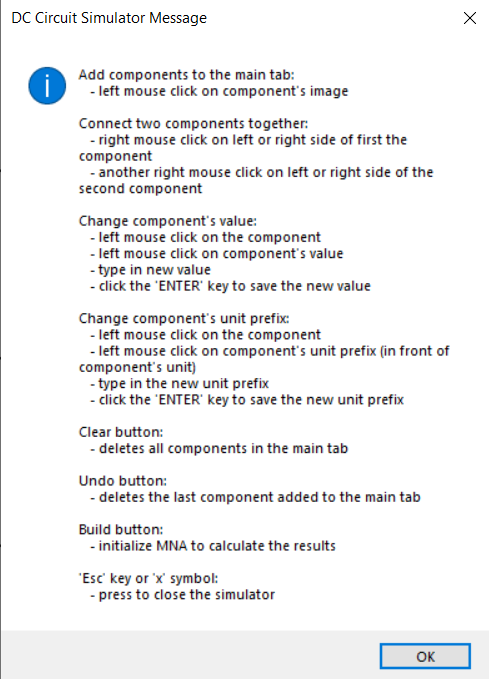
### Minimizing the user’s memory load

While making a circuit, the user should be aware of every value each component holds. In order for user to be able to use the program without having to remember component’s value, each value of every component can quickly be checked by clicking on the specific component. Once the component is selected, component’s value will be displayed to the user.

### Speaking the user’s natural language

Since the program is a teaching tool for A level physics students, a language in the simulator should be easy to understand but it also should not be too simplistic. Words and phrases used in the simulator are similar to what a student would hear in a physics lesson or read in a textbook. Therefor a simulator assumes that a student is familiar with some of the basic circuit terminology. Through using the simulator, a user will deepen their understanding and learn to use the physics phrases and language that they are expected to know in the final A level exam.

If the user is unsure how to perform a certain action while construction a circuit, e.g., how to connect two components together, they can click on the ‘Help’ button. This will display a new window with a step by step guide explaining all of the simulator’s functions. Below is a picture of what the user will be able to see when they press the ‘help’ button.



### Consistency

To maintain consistency each of the three parts of the simulator’s GUI have a common theme. Component tab is essential for constructing a circuit. It contains buttons which allow the user to add components to the main tab and is used for displaying component’s characteristics. Once displayed, these characteristics can also be customised so that the user can build a circuit which suits their needs. Toolbar provides additional buttons which help the user to construct a circuit. Buttons ‘Clear’ and ‘Undo’ can erase any mistake the user might have made. ‘Help’ button provides clear instructions if the user is unsure how to properly use the simulator. ‘Build’ button is the most important as it initialises the MNA and then displays the calculated results.

GUI was made using online three colours: white, black and grey. The minimalistic design of the GUI reduces any distractions and encourages the user to stay focused on finishing the task in as little time as possible.

## Python modules

### PyGame

PyGame is a python library which allows users to create simple GUI’s by creating a new window and then drawing various shapes and displaying images on it. With the use of classes and sprites, Pygame can be very effective and efficient way of displaying information to the user. With functions like event.get(), it also allows the user to interact with the program. In my simulator, I will make use of Pygame by allowing the user to select components, drag and drop them anywhere in the main tab, create a network of components and use buttons which enhance the user’s experience.

### Tkinter

Tkinter is a python module for creating GUIs in a fast and easy way. My program will use Tkinter’s message boxes to display text to the user. Messagebox will mostly be used to warn a user about their unvalidated input, but also to give user instructions on how to properly use the simulator or ask the user if they would like to save the built circuit by giving them ‘yes’ or ‘no’ option.

Even though Tkinter is a powerful module which is, in a lot of ways, simpler to use than Pygame, I am not going to use it to construct a main part of my program’s GUI. While Tkinter is great for creating buttons on the screen and displaying some text to the user, it is not the best option for rendering objects to the screen and changing their position.

### NumPy

Numpy is a library used for scientific computing in python. It is most commonly used when working with arrays or various derived objects (in my case – matrices). I will use NumPy for performing mathematical operations on arrays/matrices using linear algebra. I will use NumPy function linalg.solve() to obtain values of unknown variables in system of simultaneous equations.

### SciPy

SciPy is python module which also allows the use of complex mathematical methods by using NumPy underneath. The difference between SciPy and NumPy is that SciPy provides more utility functions for optimisation. I will use SciPy to optimise the ‘A’ matrix by using sparse matrix techniques. Function .sparse will allow me to convert a sparse matrix into a CSC or CSR format and then perform suitable calculations.

### OS

The OS module in Python provides a way of using operating system dependent functionality. The functions that the OS module provides allows interface with the underlying operating system that Python is running on – be that Windows, Mac or Linux. I will be using this module for file handling. When a user builds a circuit and they decide to save, a unique name will have to be given to that file. Using os.path.isfile() function, a program can check if a file with the same name already exists.

## Security and integrity of the data

My program will not be storing or processing any personal data of the user. There is no need for data to be encrypted, so anyone can access the program without authorisation.

To maintain integrity, the simulator will always keep floats to at least 5 decimal places where possible to prevent huge rounding errors as this could result in final results being incorrect.

# Testing

In order to properly test the program a series of tests will have to be planned and completed so that when the program reaches the end user there are no unexpected errors or outputs within the program. Tests will be based on the objectives from Analysis section in order to see if all objectives have been achieved.

## Objective 1

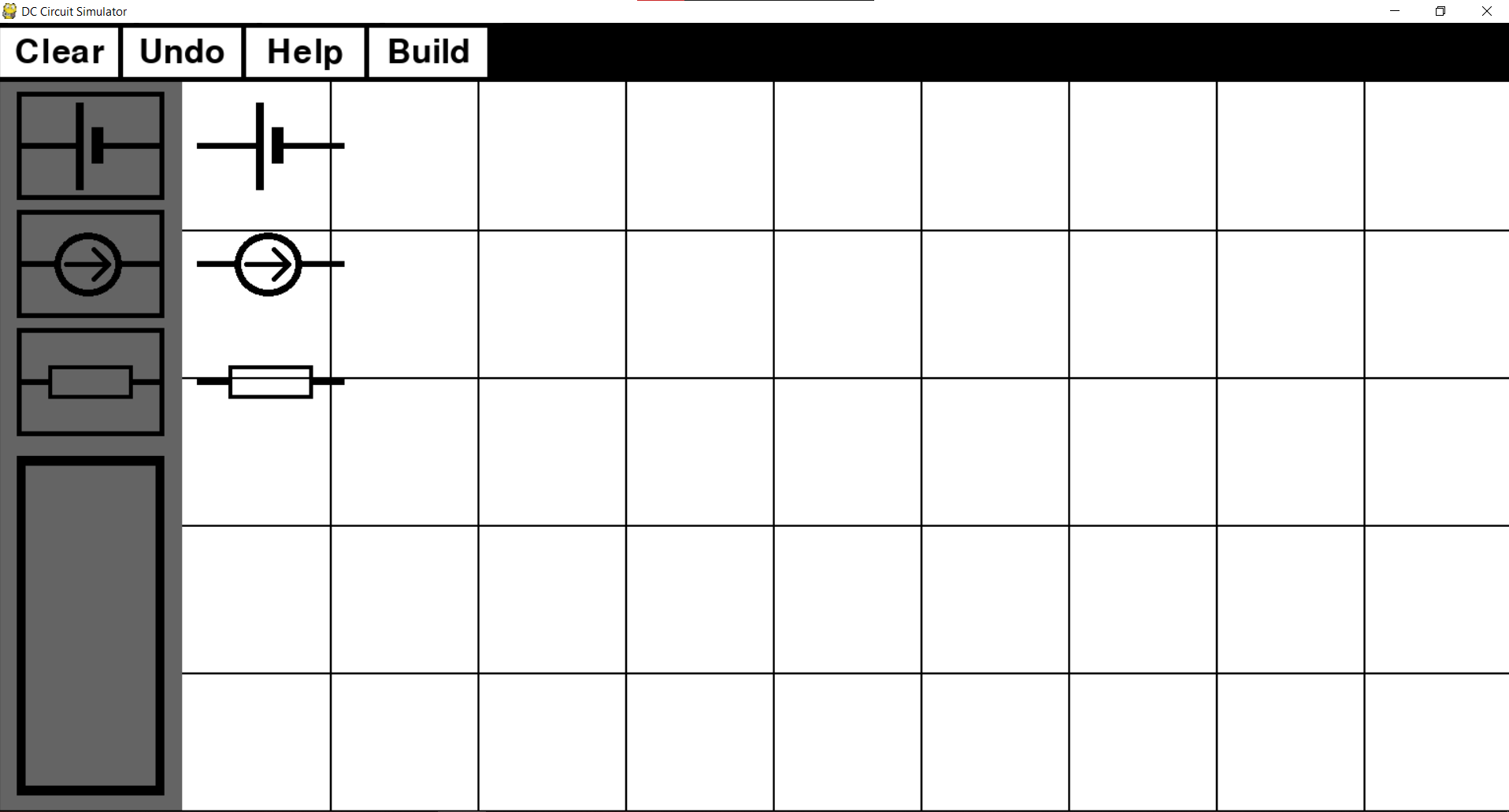
Objective 1 focuses on allowing the user to construct a circuit. Below is my objective 1.

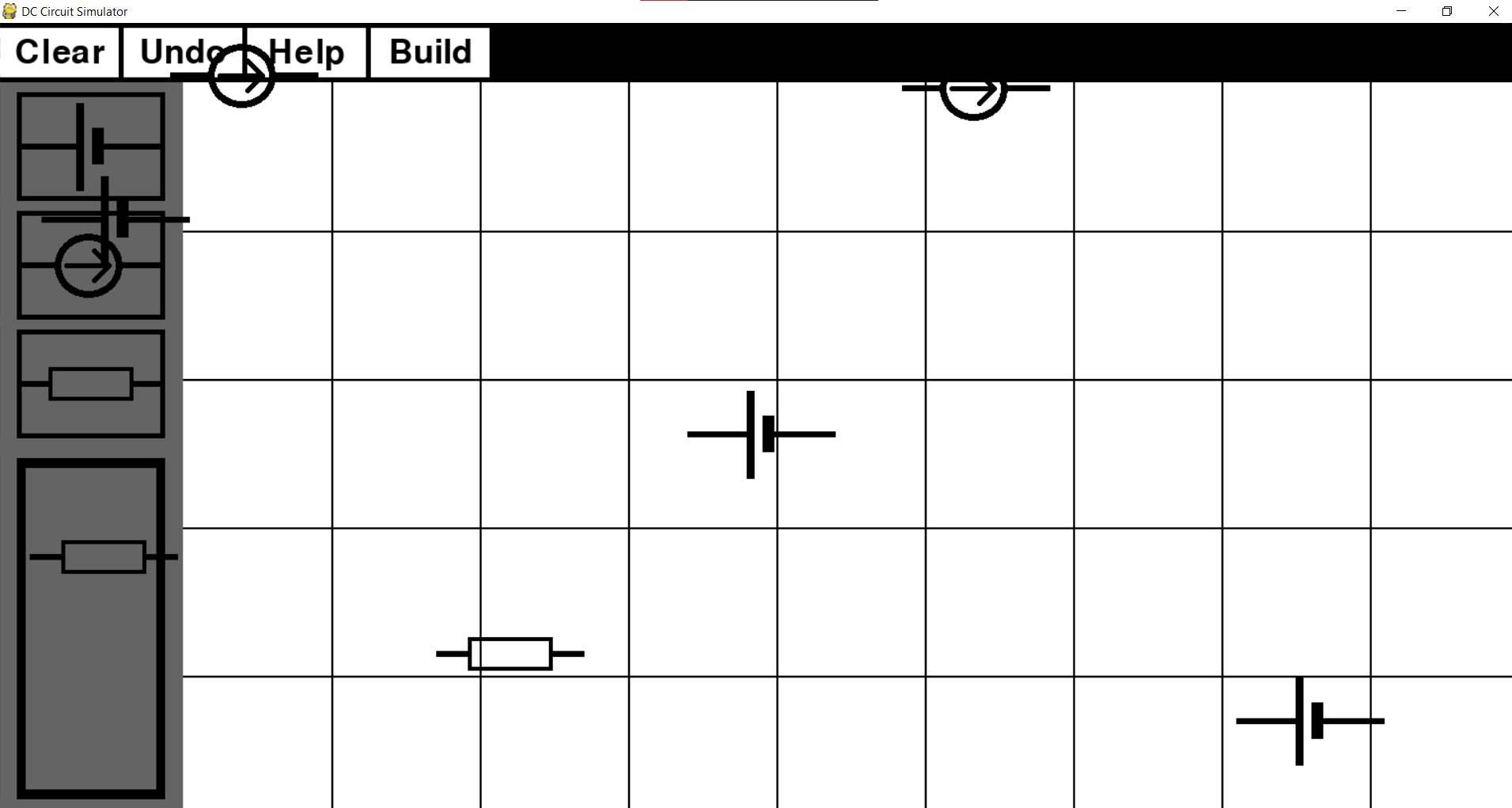
1. *Make a GUI (Graphical User Interface) which will give the user all necessary options to construct a DC circuit.*
   1. *There needs to be one button for each component which will allow the user to add the component to the main tab. By clicking on the button, a chosen component should appear in the main tab.*
   2. *A drag and drop system needs to be created for all the components which are currently in the main tab. Once a component is added to the main tab, a user should be able to change its x and y coordinates by clicking on the component and dragging it across the screen.*
   3. *There needs to be a way for the user to create a network of components in the main tab. User will be able to create a connection between two components by clicking on them with mouse’s right click button. By repeating this process, a user will be able to create a DC circuit network*
   4. *When a component in the main tab is selected by the user, a suitable feedback should be given to the user showing which component they have selected. This will be done by changing the components appearance so that it becomes unique in respect to other components of the same type in the main tab.*
   5. *For a selected component, the user should be able to see its name, value, unit prefix, unit and a short description of components use in the circuit. This will be displayed in the specific part of the screen designed for this use.*
   6. *Each type of component should have their default value and unit prefix, but once a component is selected and its characteristics are displayed, the user should be able to alter the component’s value and unit prefix by typing their own components attributes.*
   7. *No value a user inputs as a component value and a unit prefix should cause a program to crash. A validation system will be used to check if correct values have been entered. If values the user entered are incorrect or the unit prefixes do not exist, the user will be shown an appropriate message explaining that only numbers can be used as the component values or the user will be shown a list of possible unit prefixes.*

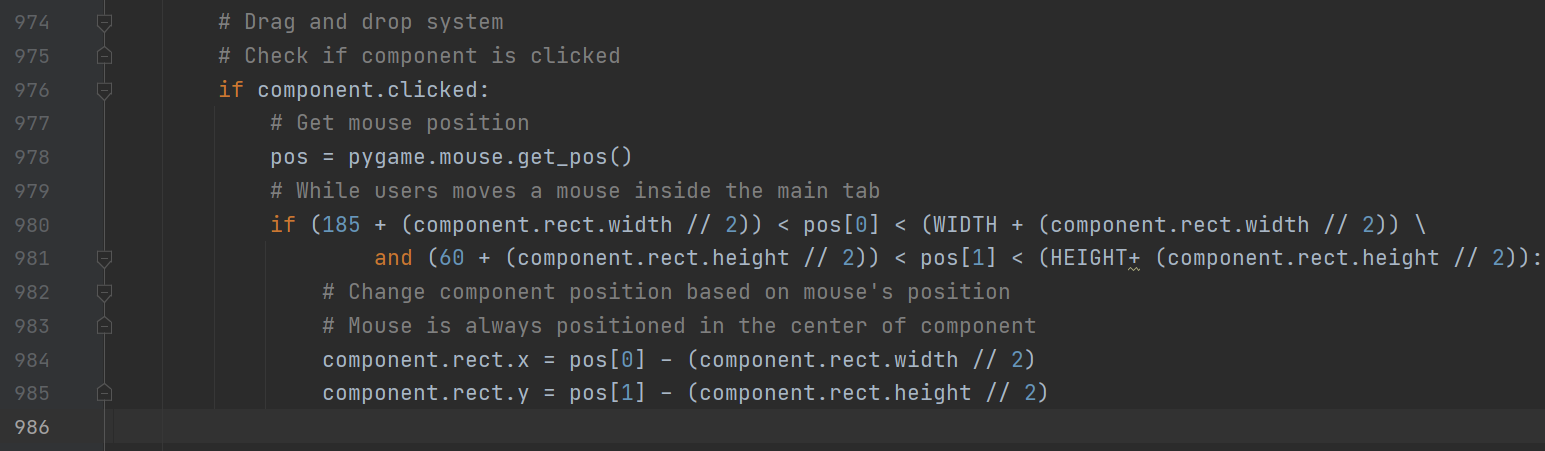
Below is a test table with description of each test, users input and outcomes. A short comment on the test is also included. Screenshot evidence of the tests are included below the table. Each screenshot is referenced by a test number.

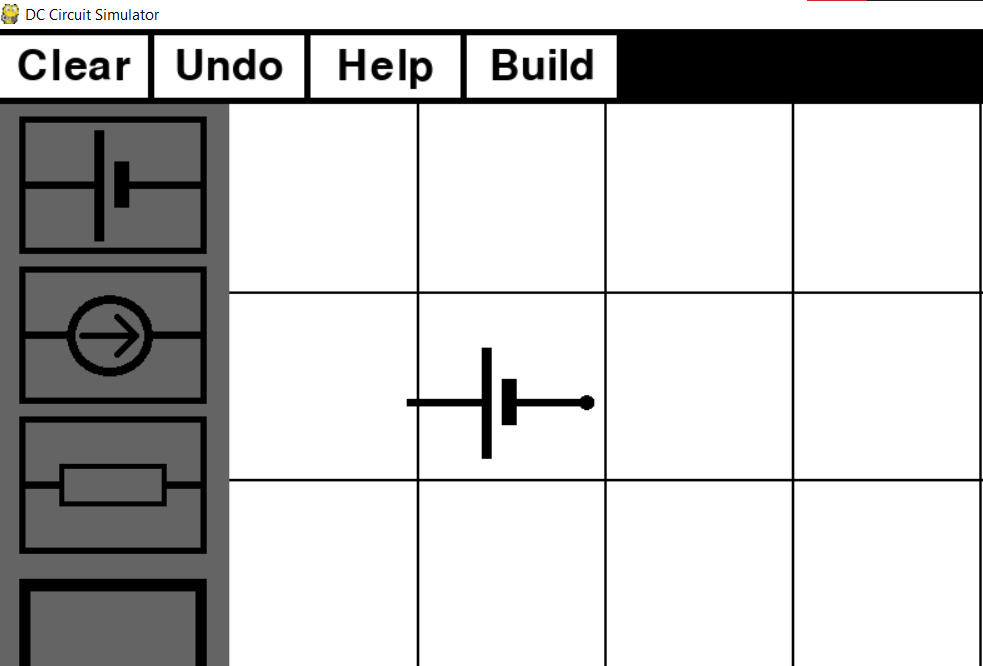
If the test failed, suitable improvements are made to the code. Screenshot of code improvements are included below test’s evidence.

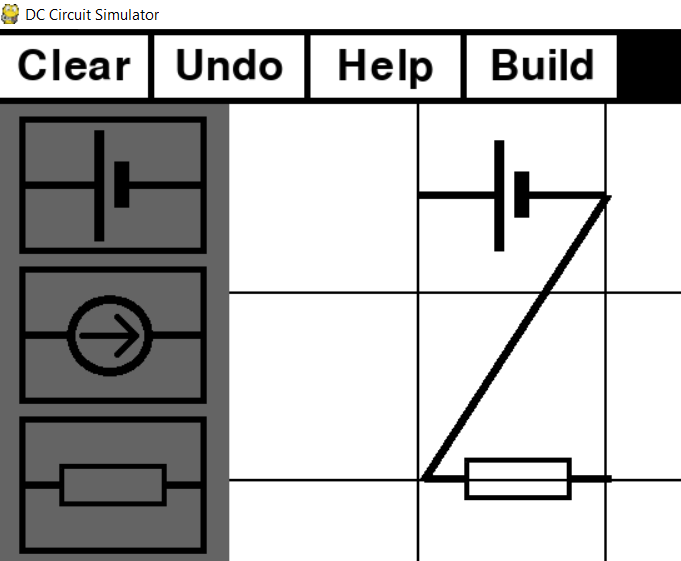
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Objective number being tested** | **Test description** | **User input** | **Expected outcome** | **Actual outcome** | **Comment on the outcome**  **(Pass/Fail)** |
| 1 | 1.1. | Button with independent voltage source image is pressed | Left mouse click on the independent voltage source image | Independent voltage source gets added to the main tab | Independent voltage source appeared next to the independent voltage source image | Correct component was added to the main tab.  **Pass** |
| 2 | 1.1 | Button with independent current source image is pressed | Left mouse click on the independent current source image | Independent current source gets added to the main tab | Independent current source appeared next to the independent current source image | Correct component was added to the main tab.  **Pass** |
| 3 | 1.1 | Button with resistor image is pressed | Left mouse click on the resistor image | Resistor gets added to the main tab | Resistor appeared next to the resistor image | Correct component was added to the main tab.  **Pass** |
| 4 | 1.2 | Component is dragged to the other side of the main tab and then dropped. | User clicks on the component with left mouse click. While holding the mouse button, mouse cursor position is changed. Mouse button is released. | Component’s position is changed according to the change of the mouse cursor position. | Component is no longer at its previous position. Its position was changed according to the change of the mouse cursor position. | User was able to drag and drop a component (change its position).  **Pass** |
| 5 | 1.2 | Component is dragged to the toolbar and then dropped. Component should not move outside the component tab. | User clicks on the component with left mouse click. While holding the mouse button, mouse cursor position is changed. Mouse button is released. | Component’s position is changed according to the change of the mouse cursor position until it reaches the edge of component tab. | Component is no longer at its previous position. Its position was changed according to the change of the mouse cursor position. | User was able to drag and drop a component (change its position), even though component’s position should stop changing once it reached the edge of the component tab.  **Fail** |
| 6 | 1.2 | Component is dragged to the component tab and then dropped. Component should not move outside the component tab. | User clicks on the component with left mouse click. While holding the mouse button, mouse cursor position is changed. Mouse button is released. | Component’s position is changed according to the change of the mouse cursor position until it reaches the edge of component tab. | Component is no longer at its previous position. Its position was changed according to the change of the mouse cursor position. | User was able to drag and drop a component (change its position), even though component’s position should stop changing once it reached the edge of the component tab.  **Fail** |
| 7 | 1.3 | To connect two components together, the user first needs to mark first component as ‘ready to connect’ | Right mouse click on component’s right side | Component gets marked as ‘ready to connect’ indicated by black circle on right side of the component | Component’s image changed to image with black circle on its right side. | Right side of component got marked as ‘ready to connect’.  **Pass** |
| 8 | 1.3 | User needs to be able to create a network between two components. | Right mouse click on another component’s left side. | A black line should be drawn, joining first component’s right side and second component’s left side. | A black line joining right and left side of two components appeared. | A network between two components was created.  **Pass** |
| 9 | 1.4 | Selected component needs to change its appearance to show the user that it was selected. | Left mouse click on a component. | Component image needs to change. | A black rectangle appeared, framing component’s image. | Component’s image changed in such way that user can differentiate selected and non-selected components.  **Pass** |
| 10 | 1.5 | User should be able to see characteristics of a selected component (name, vale, short description). | Left click on a component. | Component’s characteristics get displayed to the user. | Component’s characteristics are displayed in the component tab. | User can check component’s characteristic by selecting a component.  **Pass** |
| 11 | 1.6 | User should be able to change component’s value | Left mouse click on selected component’s value. User enters new value. | New value gets displayed and stored. | New value got displayed and stored. | User was able to change component’s value.  **Pass** |
| 12 | 1.6 | User should be able to change component’s unit prefix | Left mouse click on selected component’s unit prefix. User enters new unit prefix. | New unit prefix gets displayed and stored. | New unit prefix got displayed and stored. | User was able to change component’s unit prefix.  **Pass** |
| 13 | 1.7 | Presence check for value. | Value is left empty. | Appropriate message occurs. | ‘No value entered. Please enter component’s value.’ is displayed | Component’s value cannot be empty.  **Pass** |
| 14 | 1.7 | Range check for value | Only ‘0’ is inputted as component’s value. | Appropriate message occurs. | ‘Component’s value cannot be zero. Please enter a valid value.’ is displayed | Component’s value cannot be ‘0’.  **Pass** |
| 15 | 1.7 | Cross-reference check for value | Letter ‘T’ is entered as component’s value | Appropriate message occurs. | ‘Only numbers can be entered as component’s value.’ is displayed | Component’s value cannot contain letters.  **Pass** |
| 16 | 1.7 | Cross-reference check for unit prefix | A number ‘9’ is entered as a unit prefix | Appropriate message occurs. | ‘Unknown unit prefix entered. Please enter a valid unit prefix. Valid unit prefixes: 'f', 'p', 'n', 'u', 'm', '', 'k', 'M', 'G', 'T'.’ is displayed. | Component’s value needs to be a valid unit prefix. User was shown a list of valid unit prefixes.  **Pass** |
| 17 | 1.7 | Cross-reference check for unit prefix | A letter ‘b’ is entered as a unit prefix | Appropriate message occurs. | ‘Unknown unit prefix entered. Please enter a valid unit prefix. Valid unit prefixes: 'f', 'p', 'n', 'u', 'm', '', 'k', 'M', 'G', 'T'.’ is displayed. | Component’s unit prefix needs to be a valid unit prefix. User was shown a list of valid unit prefixes.  **Pass** |
| 18 | 1.7 | Length check for unit prefix | Two letters are entered as a unit prefix | Appropriate message occurs. | ‘Unit prefix be only one character.’ is displayed. | Component’s unit prefix can only consist of one valid letter.  **Pass** |

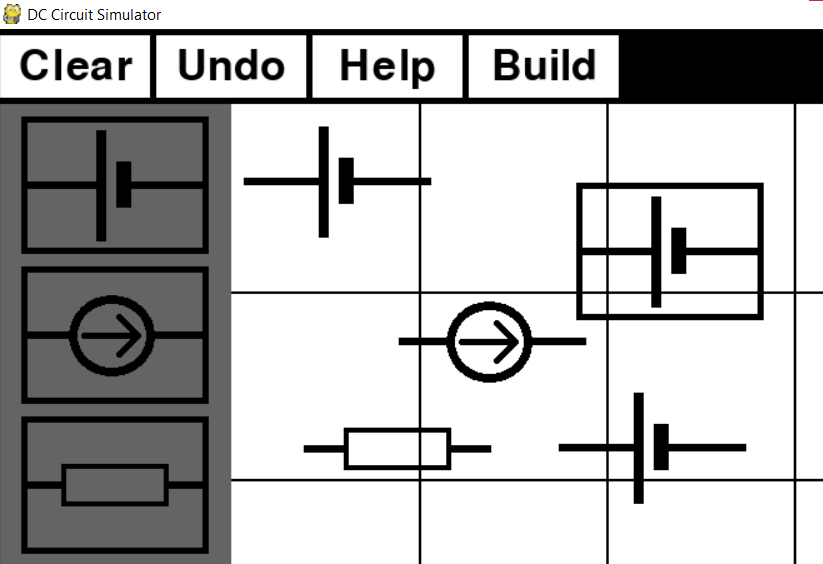
Screenshot evidence for tests 1, 2, 3:

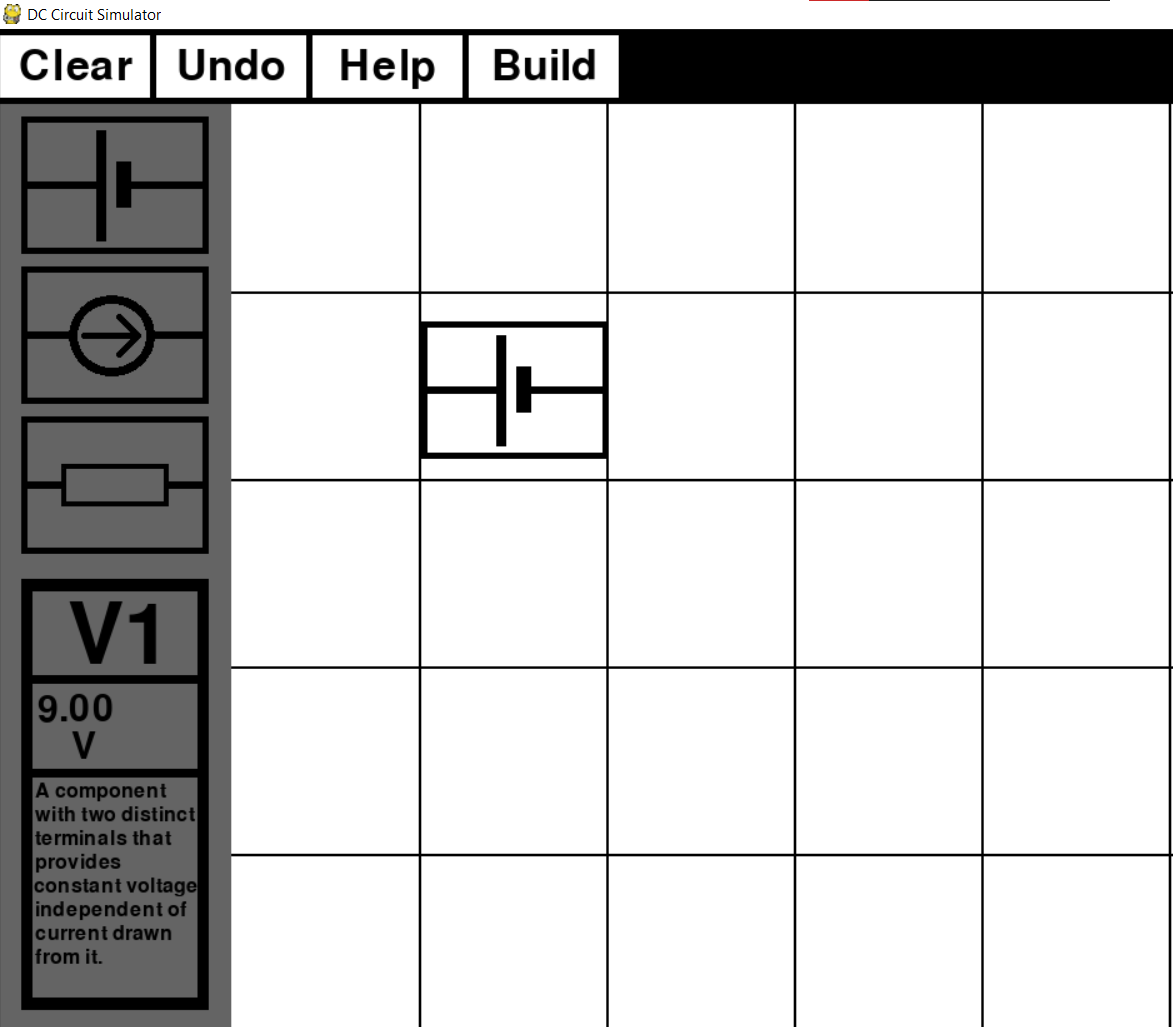
Screenshot evidence for tests 4, 5, 6: 

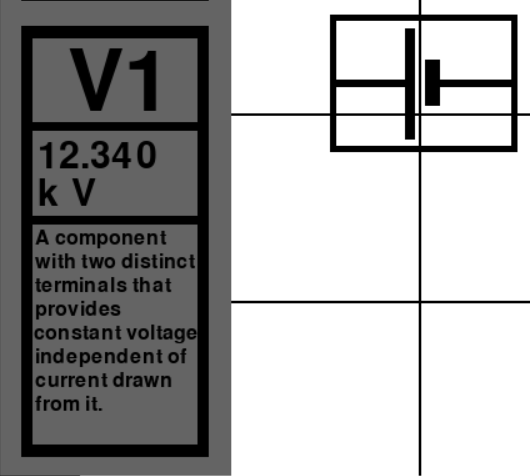
Component should not be allowed to move outside the component tab. Below is a code which fixes this issuse. 

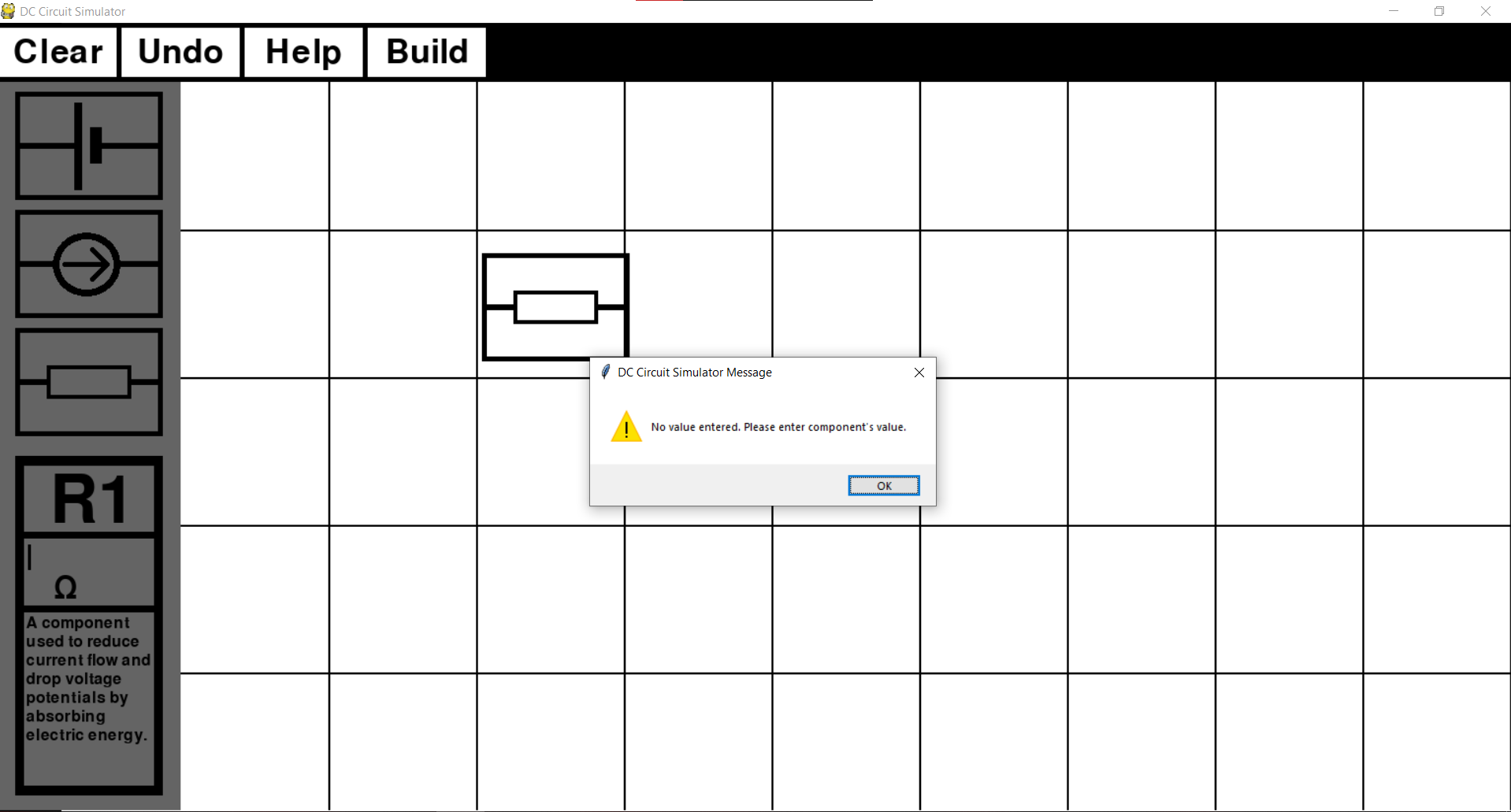
Screenshot evidence for test 7: 

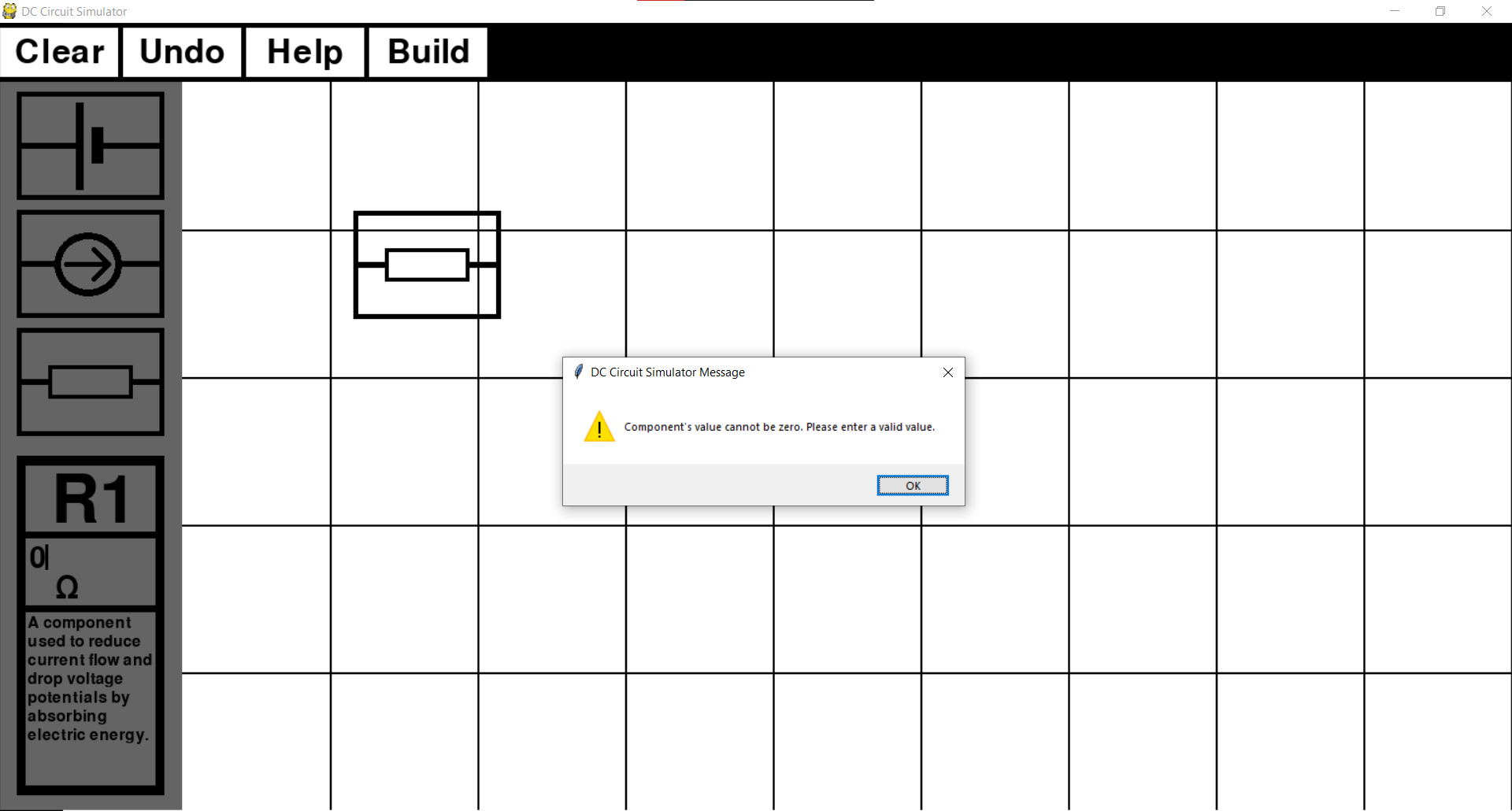
Screenshot evidence for test 8: 

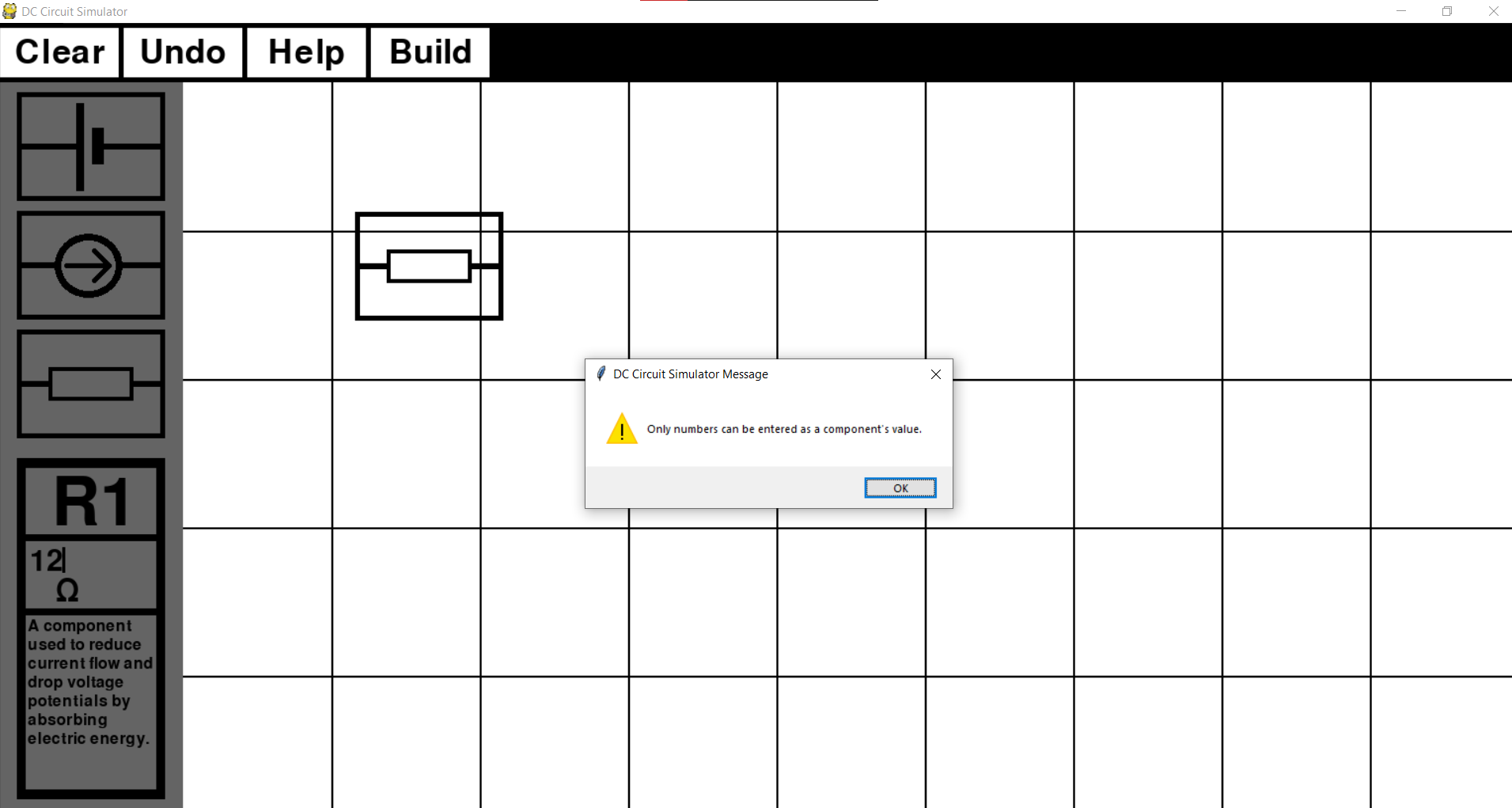
Screenshot evidence for test 9: 

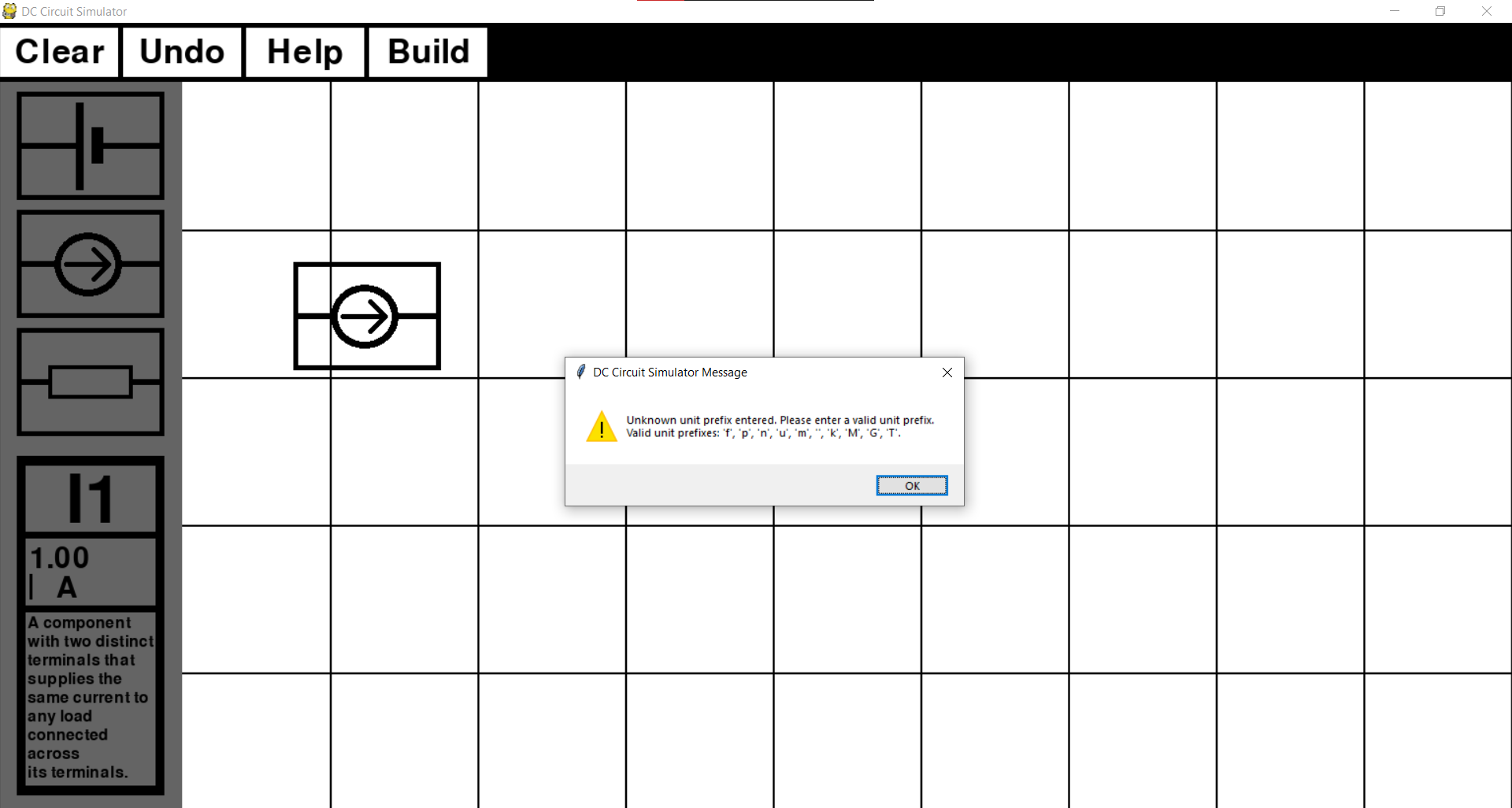
Screenshot evidence for test 10: 

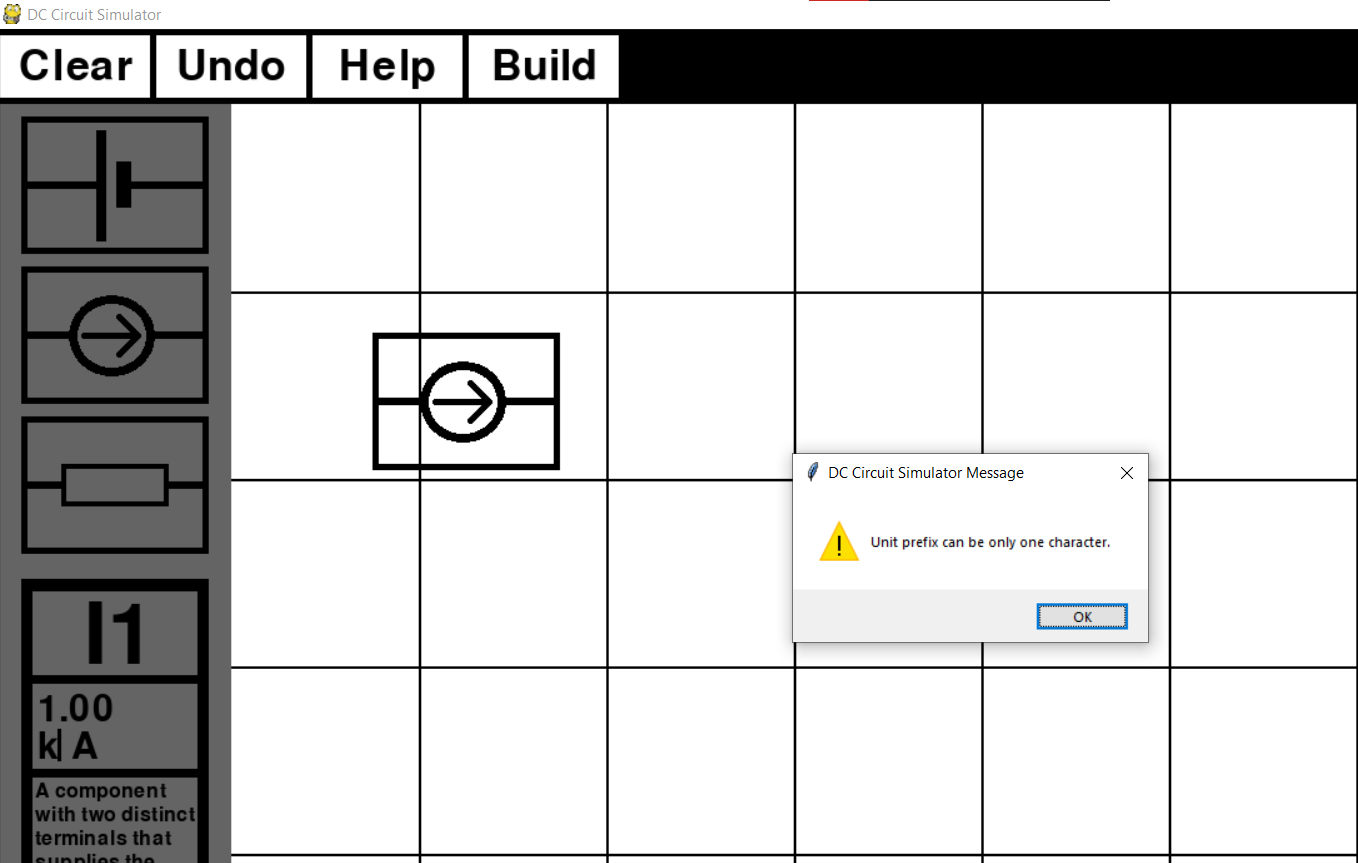
Screenshot evidence for test 11, 12:

Screenshot evidence for test 13:

Screenshot evidence for test 14: 

Screenshot evidence for test 15: 

Screenshot evidence for test 15, 16: 

Screenshot evidence for test 17: 

## Objective 2

Objective 2 builds up on idea of giving user the ability to construct a circuit. With objective 2 I wanted to provide additional features to the simulator which can help the user to construct a circuit but are not necessary to do so. Below is my objective 2.

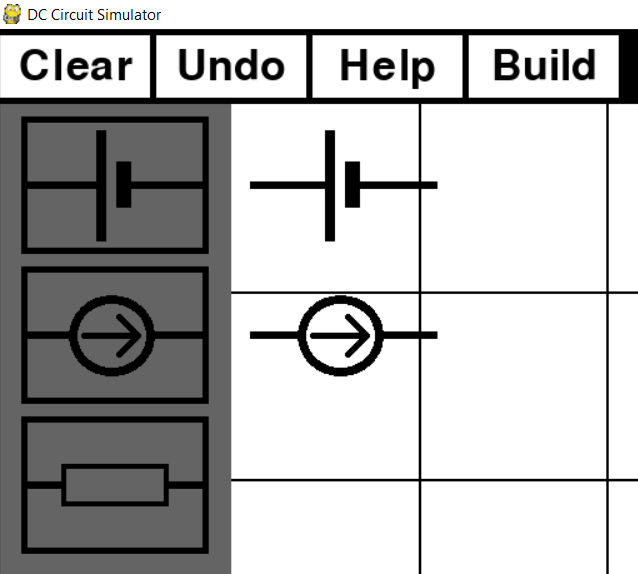
1. *In case the user makes a mistake while building a circuit (adds a wrong component to the main tab, makes a wrong connection between two components, etc.) they should be able to correct it by clicking on the specific buttons in the toolbar.*
   1. *A button which will undo the user’s latest action should be included, so that if the user notices their mistake immediately, they can quickly correct it.*
   2. *In case a whole circuit is built in an incorrect way or the user would like to build a new circuit, the user should be able to clear the whole main tab. This button will delete all components which are currently there.*

Test table is below.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Objective number being tested** | **Test description** | **User input** | **Expected outcome** | **Actual outcome** | **Comment on the outcome**  **(Pass/Fail)** |
| 1 | 2.1 | Three components are added to the main tab and ‘undo’ button is pressed. | Left mouse click on ‘undo’ button. | Last component added to the main tab gets deleted. | Last component added to the main tab was deleted. | A user can delete last component added to the main tab.  **Pass** |
| 2 | 2.1 | Latest link made between two components should be deleted with the ‘undo’ button. | User connects three components together and presses ‘undo’ button. | Last link made between two components gets deleted. | Last component added to the main tab was deleted together with all its links. | A user was not able to delete last link made.  **Fail** |
| 3 | 2.2 | Three components are added to the main tab and ‘clear’ button is pressed. | Left mouse click on ‘clear’ button. | All three components get deleted from main tab. | All three components get deleted from main tab. | The user can ‘clear’ all components from the main tab.  **Pass** |
| 4 | 2.2 | Three components connected in a network should get deleted together with their links. | Left mouse click on ‘clear’ button. | All three components get deleted from main tab together with their links. | All three components get deleted from main tab together with their links. | The user can ‘clear’ all components and their links from the main tab.  **Pass** |

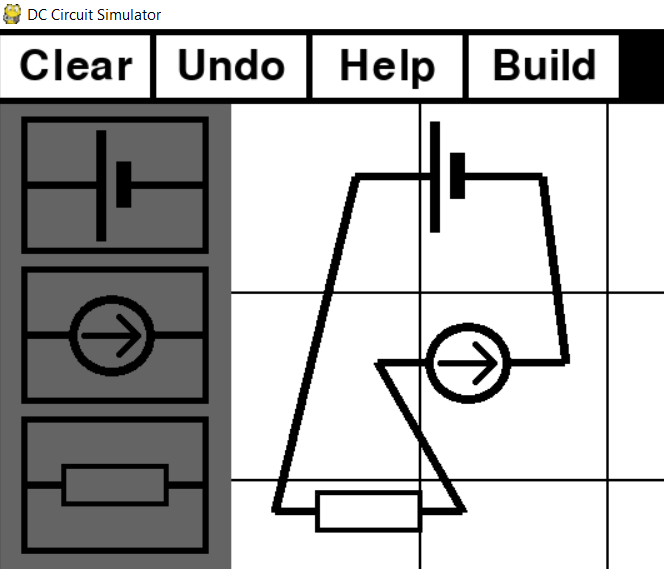
Screenshot evidence for test 1:

Before undo button was pressed: After undo button was pressed:

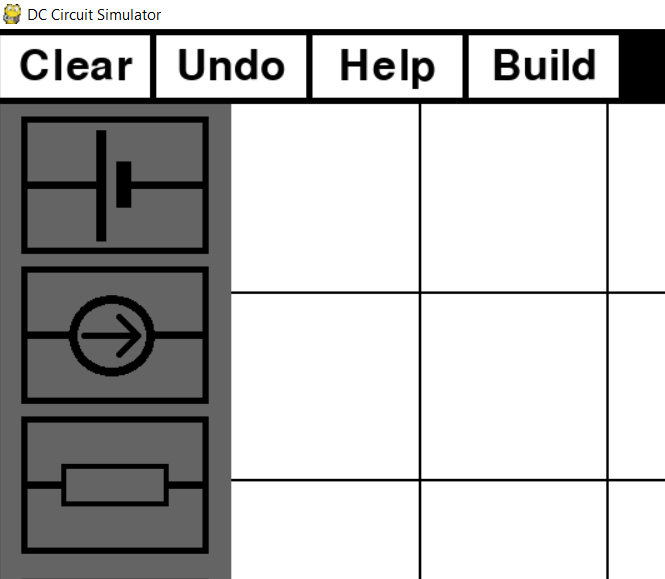
Screenshot evidence for test 2:

Before undo button was pressed: After undo button was pressed:

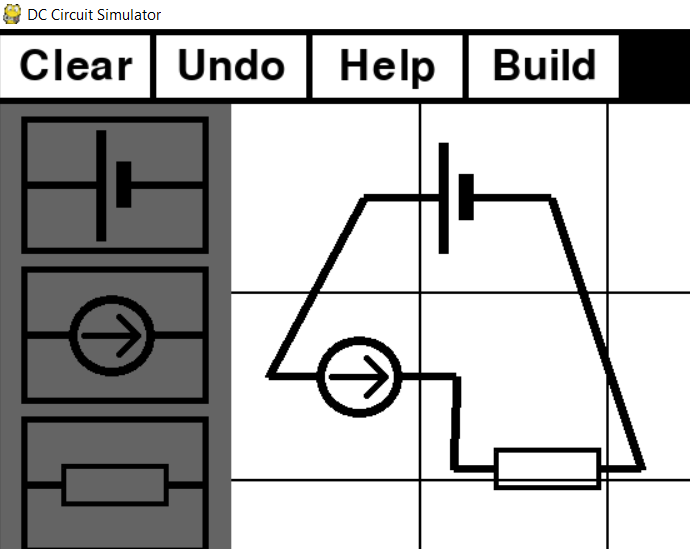
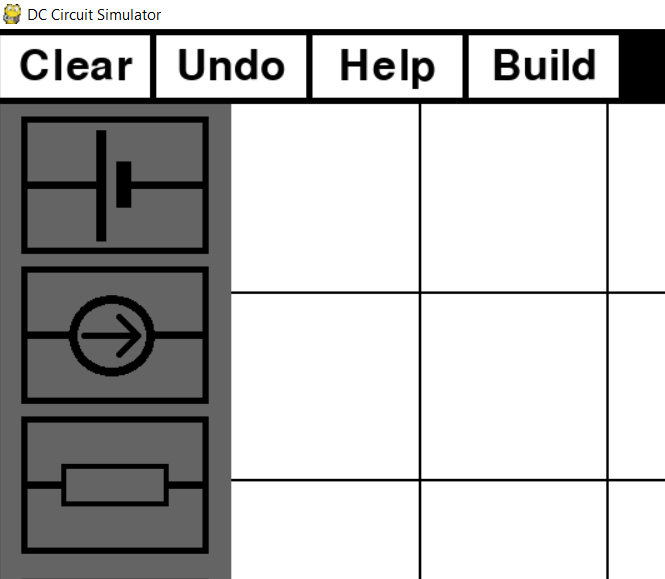
Screenshot evidence for test 3:

Before clear button was pressed: After clear button was pressed:

Screenshot evidence for test 3:

Before clear button was pressed: After clear button was pressed:

# Objective 3

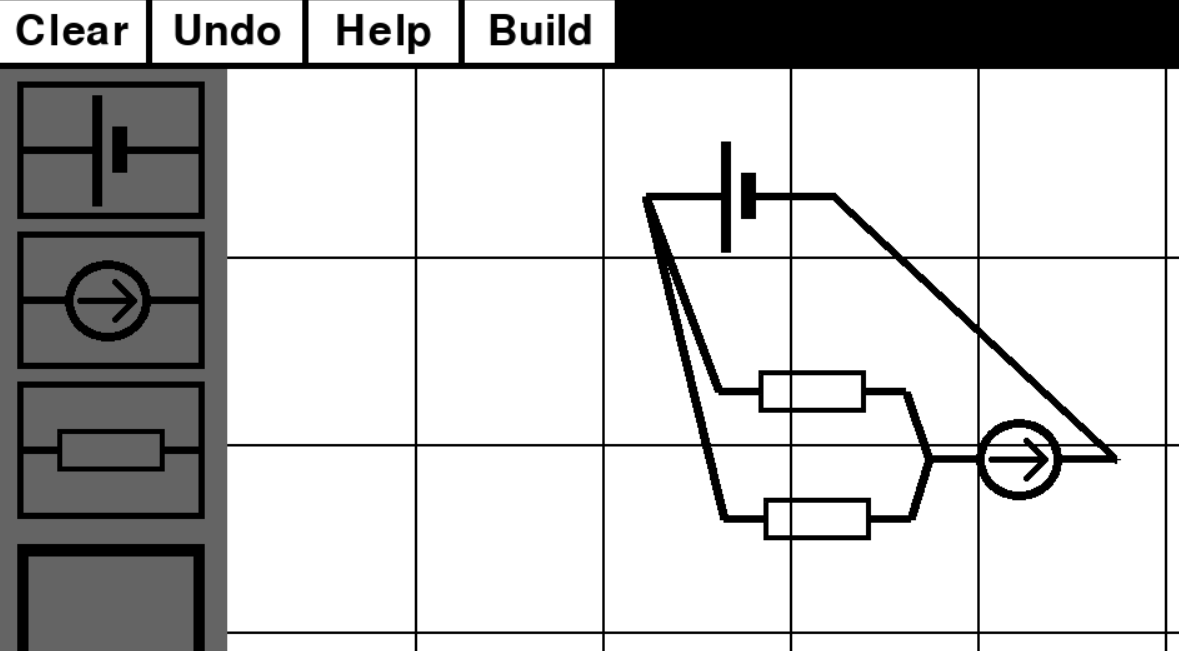
With objective 3 I aimed for my program to gather information it of the build circuit and covert it into a circuit netlist.

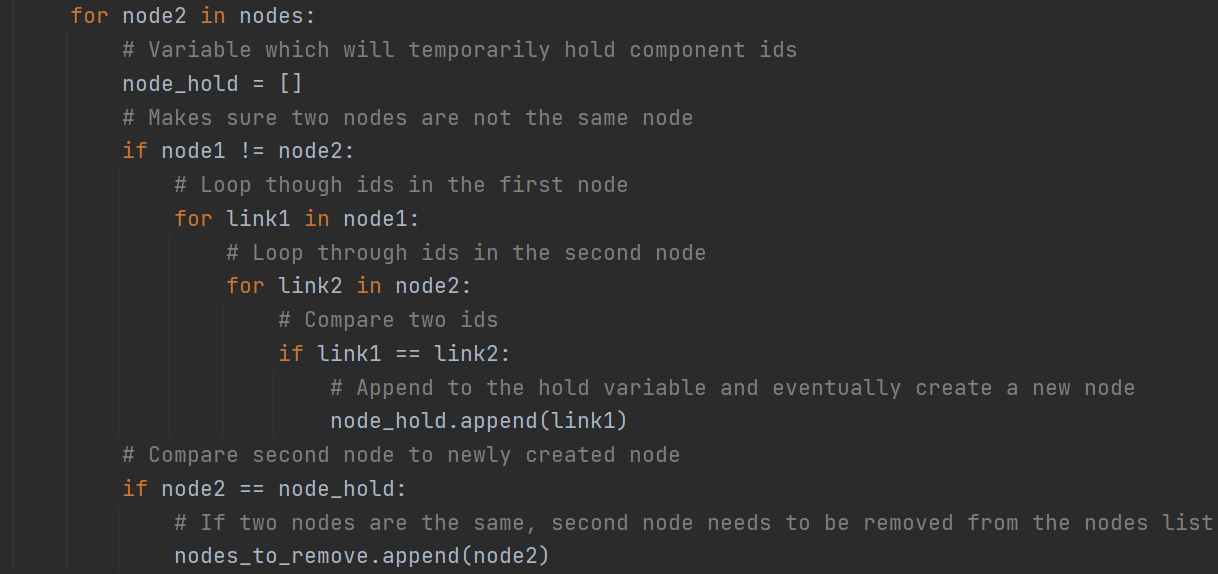
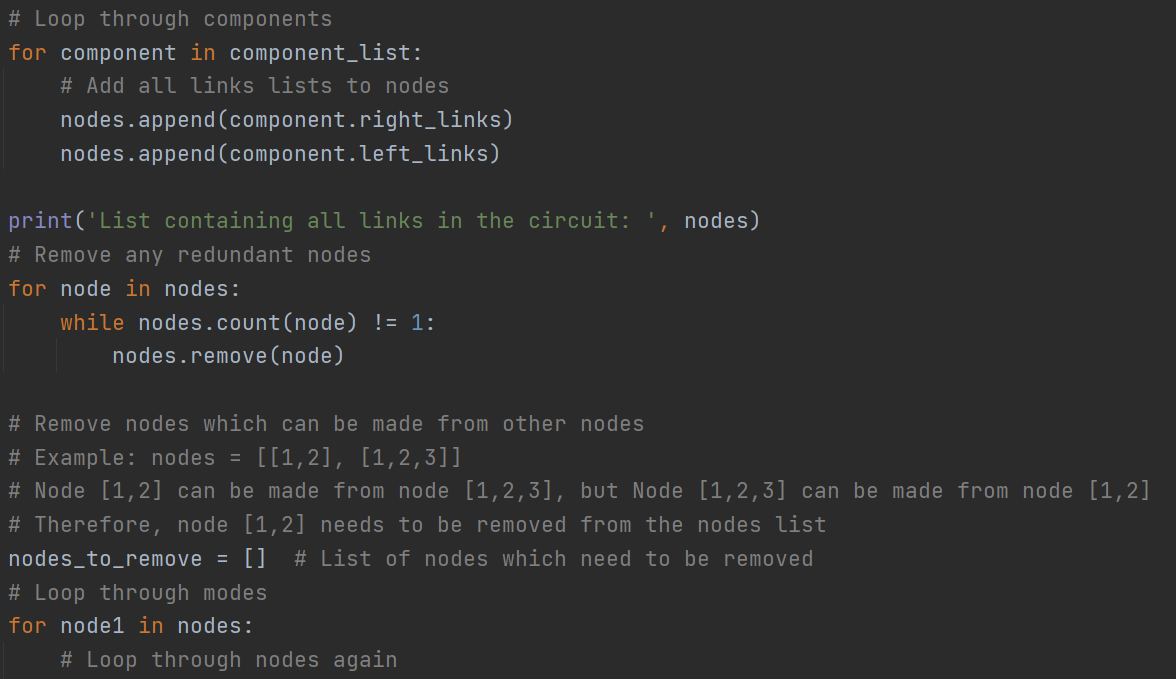
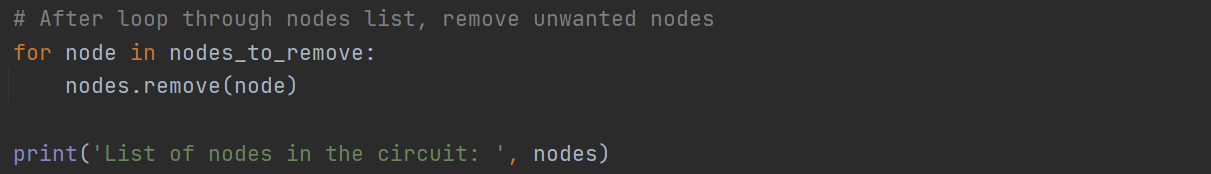
1. *Once a circuit is completed, it should be analysed and converted into a format which computer can understand.*
   1. *Once the analysis of the circuit is complete, a netlist in a text format should be created and stored as an external file.*
   2. *If the program detects an error, analysis should be stopped, and an appropriate message should be displayed informing the user that circuit was built in an incorrect way.*

First two tests show that the simulator takes data about links between components, converts it into a one list containing nodes and then makes a netlists file. To test the program’s validation of a circuit, a number of incorrect circuits will be built and analysed by the program.

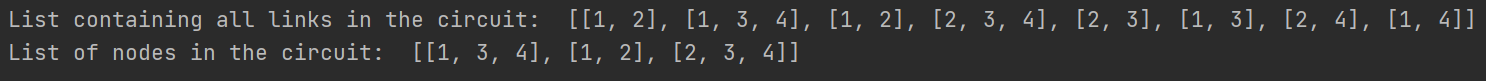
Test table is below.

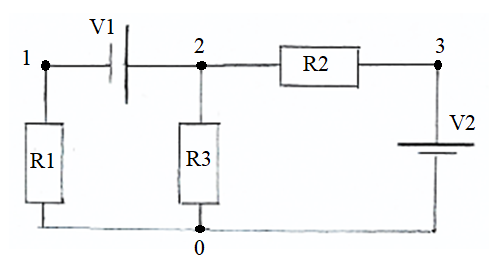
|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Objective number being tested** | **Test description** | **User input** | **Expected outcome** | **Actual outcome** | **Comment on the outcome**  **(Pass/Fail)** |
| 1 | 3.1 | From a list containing circuit links, a nodes list needs to be created | Create a network between components | List of nodes in that circuit. | List containing only circuit nodes | From a links list a nodes list was created.  **Pass** |
| 2 | 3.1 | Test the way a program creates a circuit netlist by creating an example circuit from the analysis section. | Create a circuit same as the example circuit in analysis. | Same netlist as the example’s circuit netlist. | Netlist with identical component names and values. Even though nodes are not identical, they are still both correct. Same nodes are represented with different numbers, e.g. node represented with ‘2’ in example circuit is represented as ‘0’ node in the netlist circuit. | Netlist names, nodes and values correspond to the example circuit.  **Pass** |
| 3 | 3.2 | Try analysing a circuit with no components in the main tab. | Press built button with empty main tab. | Appropriate message occurs warning the user about their mistake. | ‘Not enough components to construct a circuit.’ is displayed. | Circuit is rejected and appropriate warning is displayed.  **Pass** |
| 4 | 3.2 | Try analysing a circuit with only one component in the main tab. | Add one component and press built button. | Appropriate message occurs warning the user about their mistake. | ‘Not enough components to construct a circuit.’ is displayed. | Circuit is rejected and appropriate warning is displayed.  **Pass** |
| 5 | 3.2 | Try analysing a circuit with two components in the main tab. | Add two components and press built button. | Appropriate message occurs warning the user about their mistake. | ‘Not enough components to construct a circuit.’ is displayed. | Circuit is rejected and appropriate warning is displayed.  **Pass** |
| 6 | 3.2 | Try analysing a circuit with three components in the main tab and no connections between them. | Add three components and press built button. | Appropriate message occurs warning the user about their mistake. | ‘Not all components are connected in a circuit. Please connect all components.‘ is displayed. | Circuit is rejected and appropriate warning is displayed.  **Pass** |
| 7 | 3.2 | Try analysing a circuit with three components connected together and one component with no connections. | Connect three components and press built button. | Appropriate message occurs warning the user about their mistake. | ‘Not all components are connected in a circuit. Please connect all components.‘ is displayed. | Circuit is rejected and appropriate warning is displayed.  **Pass** |
| 8 | 3.2 | Try analysing a circuit with four components connected together except for component one terminal. | Connect four components and press build button. | Appropriate message occurs warning the user about their mistake. | ‘Not all components are connected in a circuit. Please connect all components.‘ is displayed. | Circuit is rejected and appropriate warning is displayed.  **Pass** |

Screenshot evidence for test 1:

Screenshot of code for creating and printing links list and nodes list: 

Code output:



Evidence for test 2:

Example netlist:

V1 2 1 9

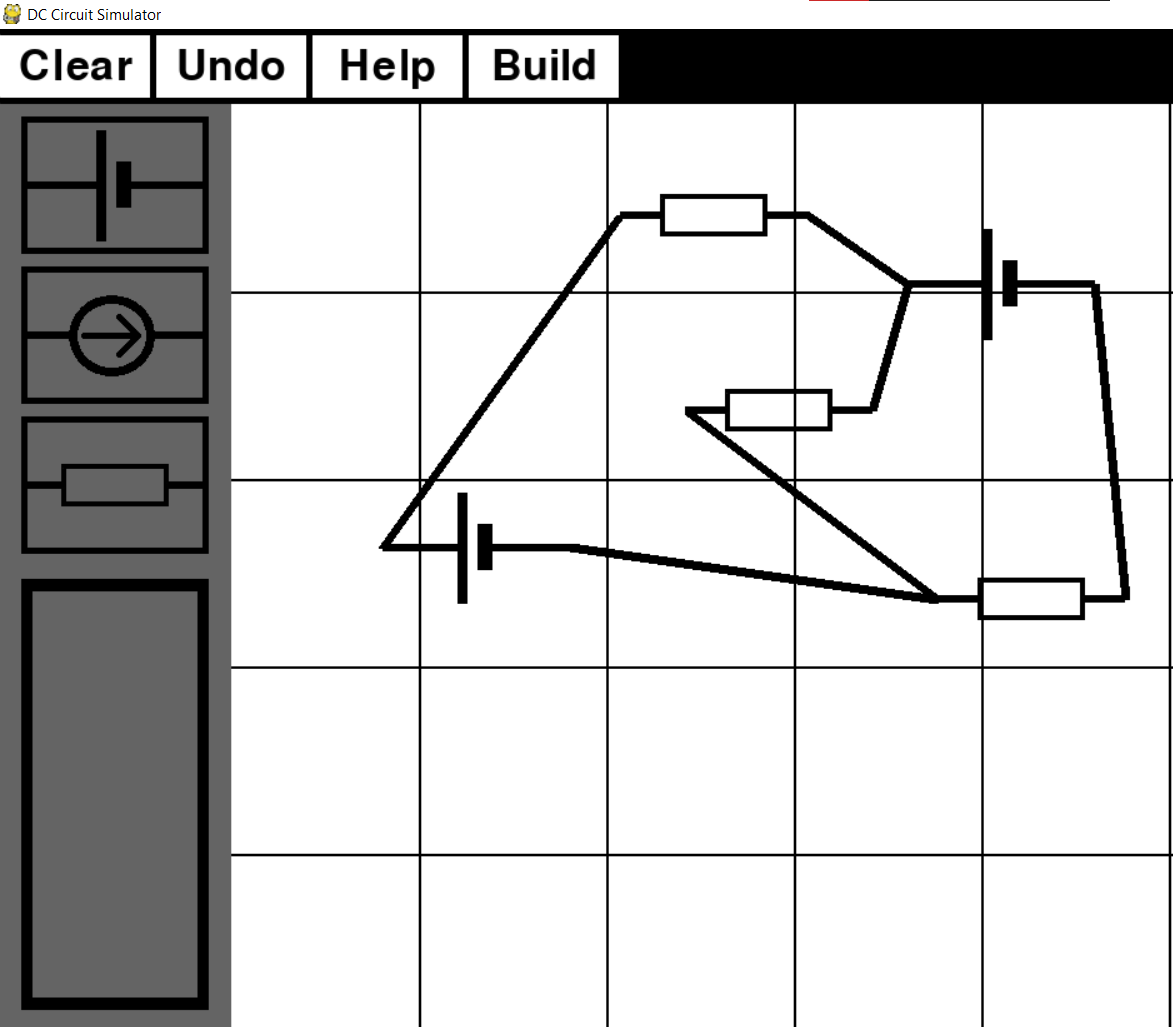
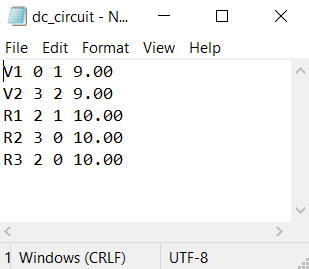
V2 0 3 9

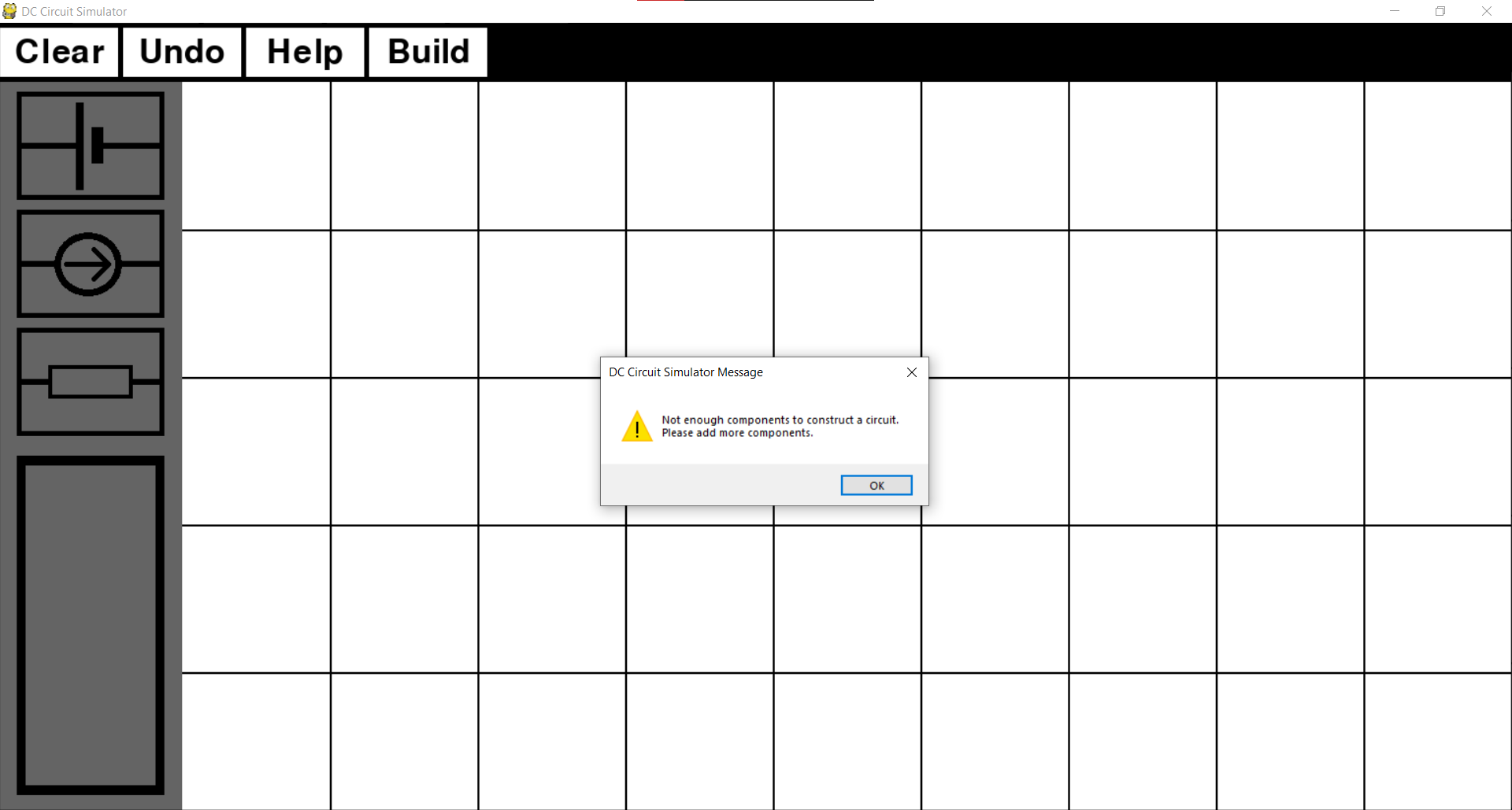
R1 1 0 10

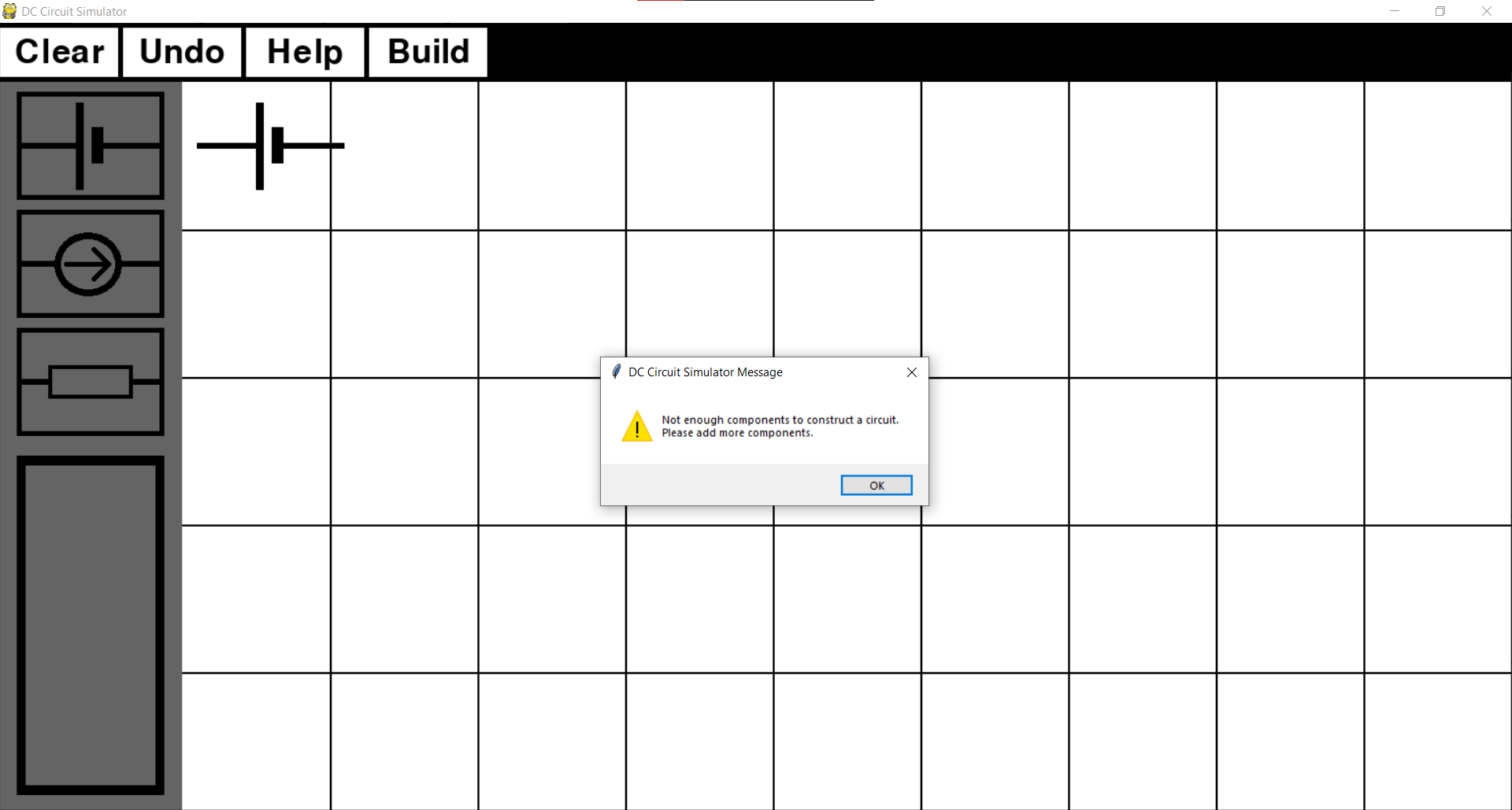
R2 2 3 10

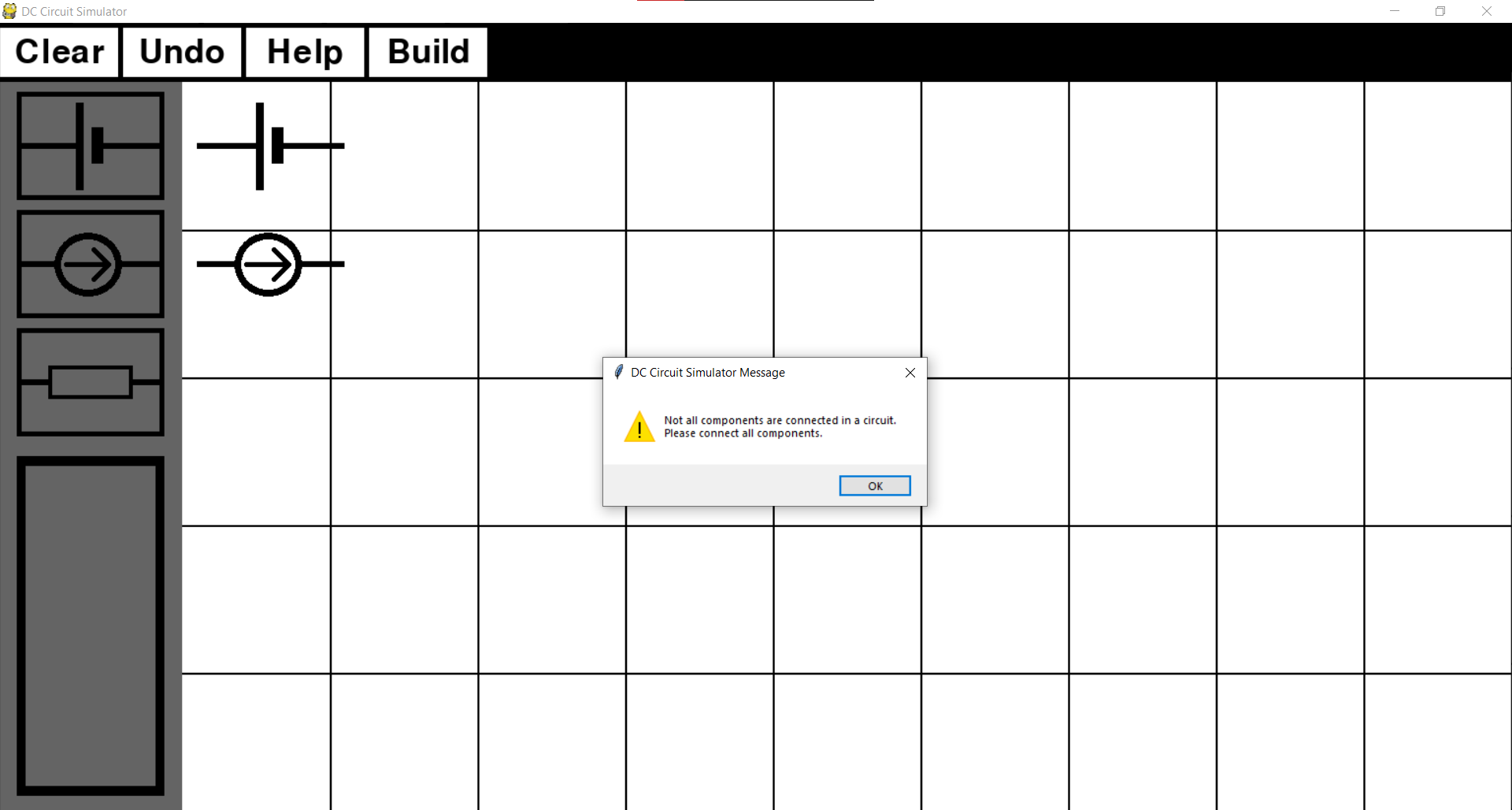
R3 2 0 10

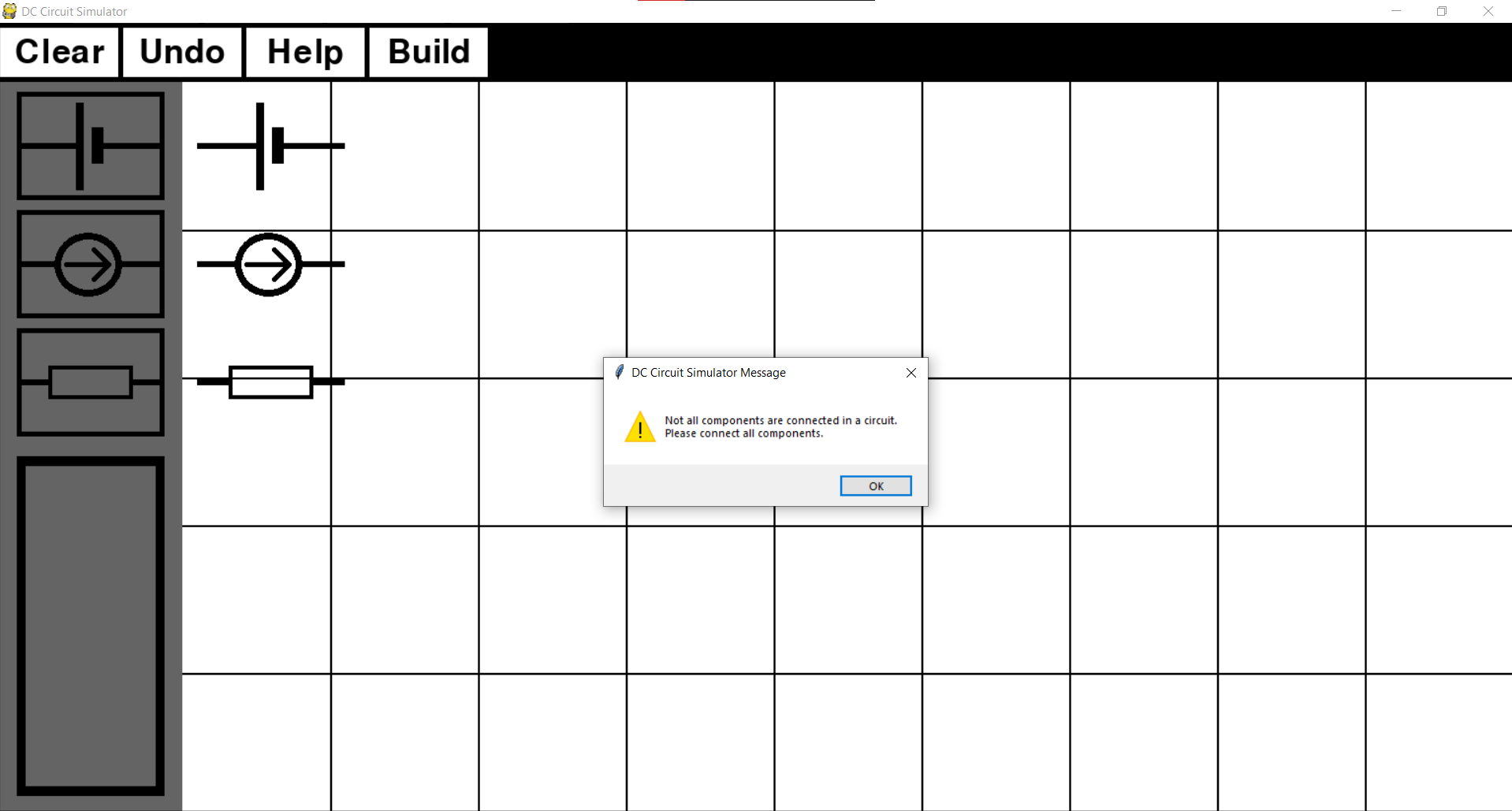
Screenshot of a circuit built in the simulator: Circuit netlist:

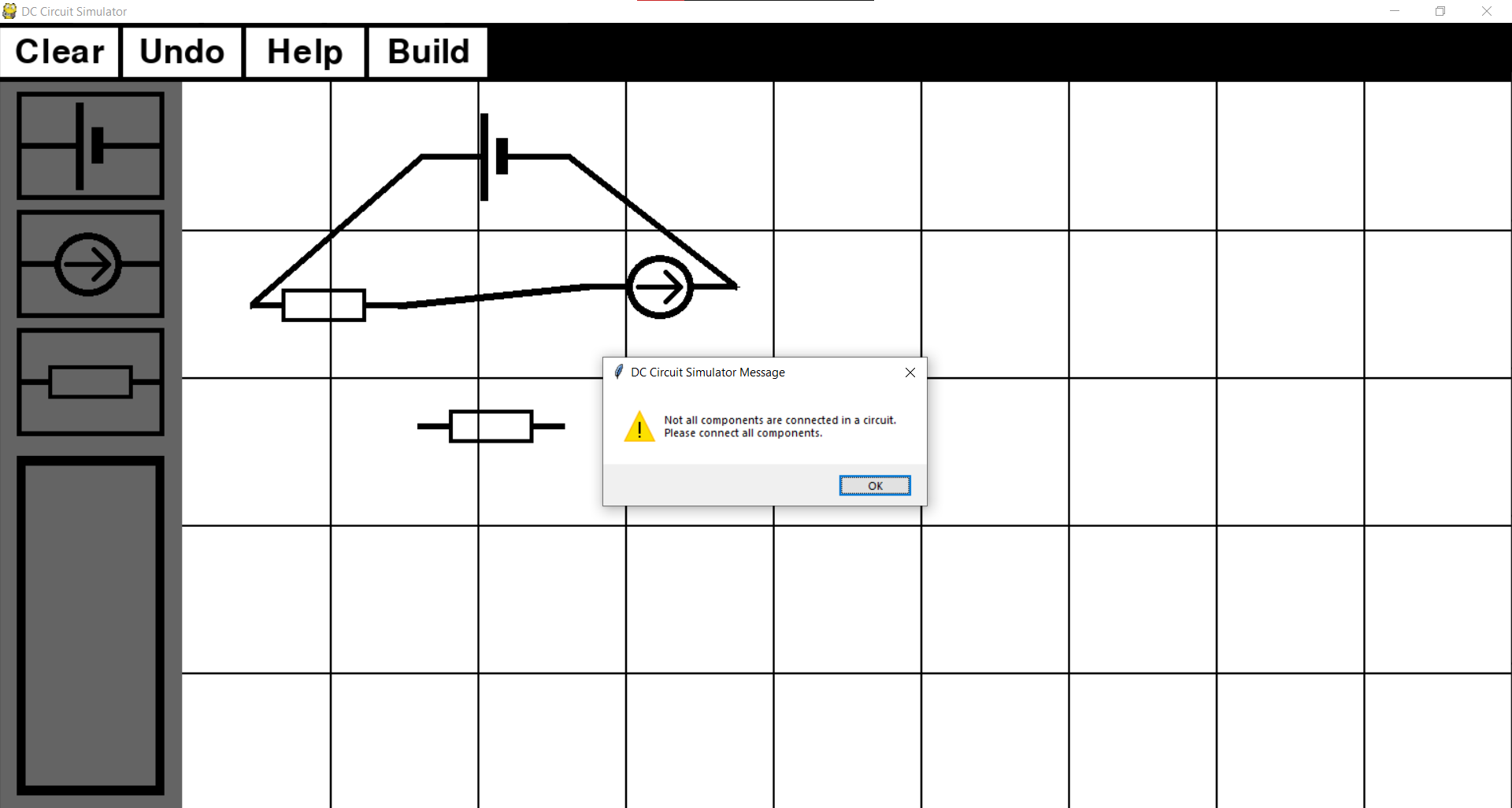
 

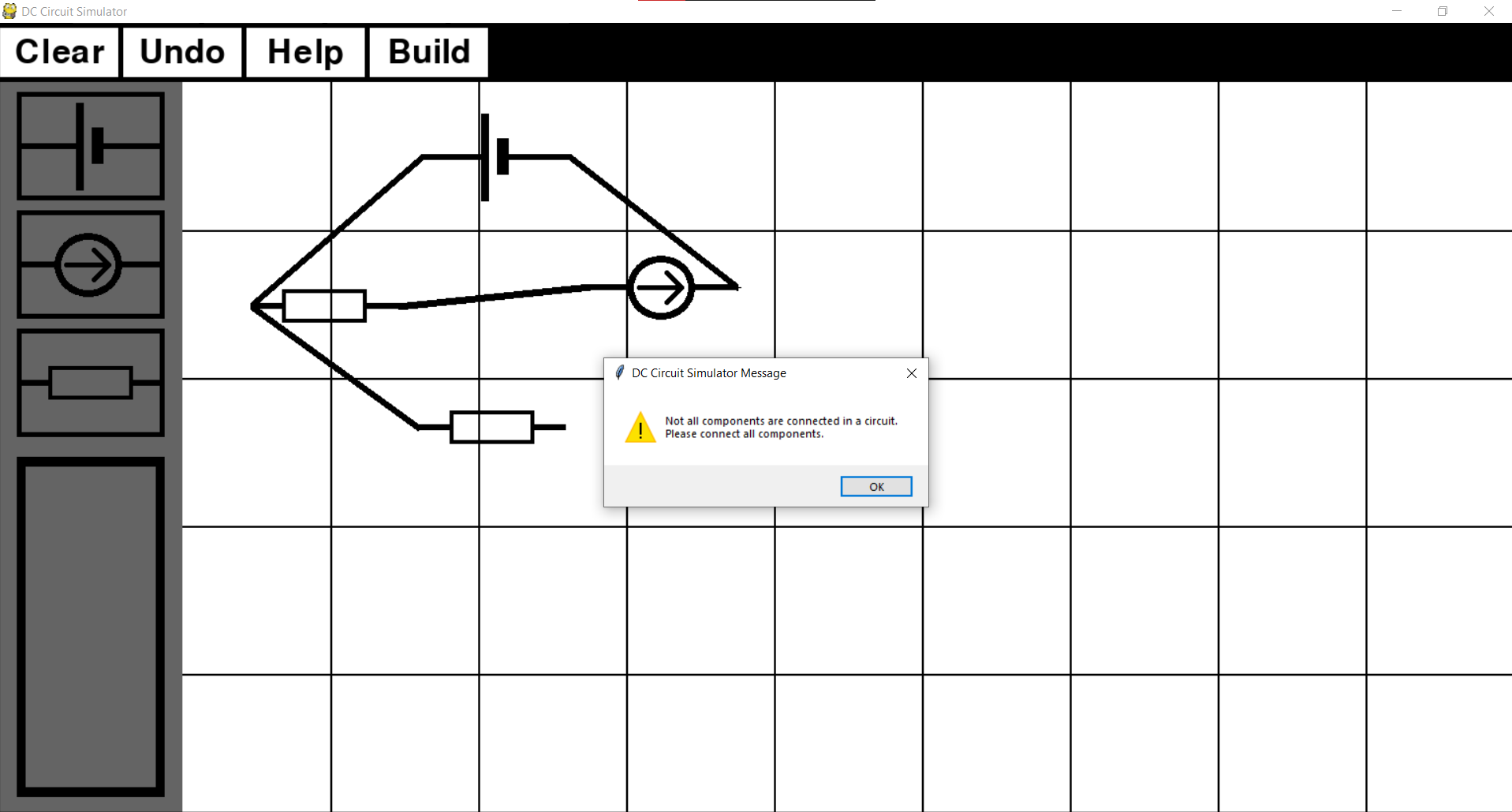
Screenshot evidence for test 3: 

Screenshot evidence for test 4: 

Screenshot evidence for test 5: 

Screenshot evidence for test 6: 

Screenshot evidence for test 7: 

Screenshot evidence for test 8: 

## Objective 4

My aim with objective 4 was to calculate voltages across each node and current flowing through every independent voltage source in the circuit. These are the most important values to know when solving a circuit question.

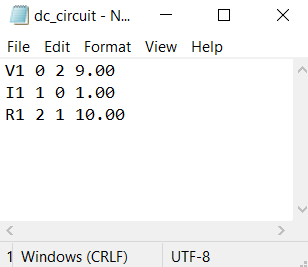
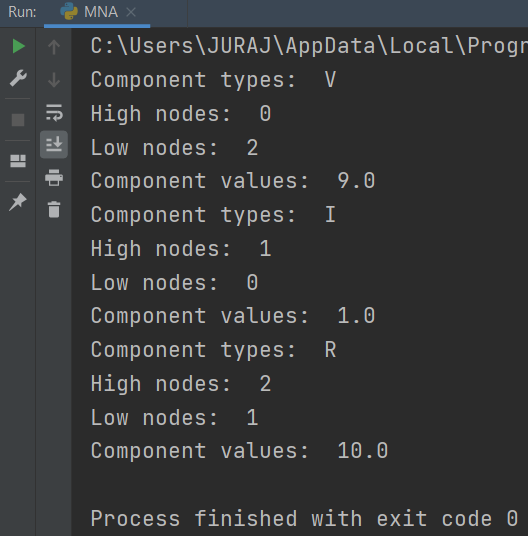
1. *Once a netlist is stored in a text file, MNA will be initialised and once it is complete, final results will be displayed to the user.*
   1. *The program should begin MNA analysis by processing the information of a netlist. This will be done by first parsing a netlist file and then mapping string nodes from netlist file into integer nodes which represent indices of the MNA matrix*
   2. *Construct A matrix by combining sub-matrices G, B, C and D*
   3. *Construct z matrix using the i matrix containing the voltages of independent voltage and e matrix containing the currents of the independent current sources.*
   4. *Solve the linear matrix equation to obtain x matrix*
   5. *Once x matrix is calculated, display MNA results to the user.*
   6. *If the circuit is shorted or for any other reason cannot be calculated so it produces and error, appropriate message should be displayed informing the user that circuit was incorrectly built.*

MNA will be tested by first checking if it passes the file in a correct way (do parsed values correspond to original values from the netlist). Then I will check if the results obtained by running the MNA and solving matrix equation, are correct by comparing it to an example circuit. Finally, I will test the MNA system by building circuit with normal boundary and erroneous data.

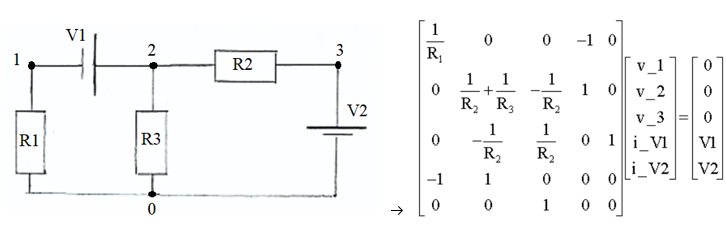
Test table is below

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Objective number being tested** | **Test description** | **User input** | **Expected outcome** | **Actual outcome** | **Comment on the outcome**  **(Pass/Fail)** |
| 1 | 4.1. | Check if MNA parses the netlist text file correctly. | User constructs a circuit and presses ‘build’ button | Parsed values corresponding to netlist file | Values correspond to each other | File gets parsed correctly.  **Pass** |
| 2 | 4.4 | Check if calculated x matrix corresponds to nodal voltages and currents flowing through voltage sources. | User constructs a circuit and presses ‘build’ button. | x matrix needs to be show correct circuit values. | x matrix has correct circuit values. | x matrix gives correct values.  **Pass** |
| 3 | 4.5 | User needs to see calculated results. | User constructs a circuit and presses ‘build’ button. | x matrix needs to be displayed to the user with explanation what each value represents. | Values displayed with using Tkinter message box appropriate explanation | User can see the circuit results.  **Pass** |
| 4 | 4.6 | Create a circuit with serial connections  (normal). | User constructs a circuit and presses ‘build’ button. | Output calculated circuit values. | Circuit values outputted. | Values calculated and displayed to the user.  **Pass** |
| 5 | 4.6 | Create a circuit with parallel connections (normal). | User constructs a circuit and presses ‘build’ button. | Output calculated circuit values. | Circuit values outputted. | Values calculated and displayed to the user.  **Pass** |
| 6 | 4.6 | Create a circuit with one incorrect parallel connection (boundary). | User constructs a circuit and presses ‘build’ button. | Display an appropriate warning to the user. | Program crashed. | A user should have been warned about their mistake.  **Fail** |
| 7 | 4.6 | Create a circuit which is shorted. (erroneous) | User constructs a circuit and presses ‘build’ button. | Display an appropriate warning to the user. | ‘Circuit was built in an incorrect way.’ is displayed. | Error detected, circuit is rejected, and user is warned about their mistake.  **Pass** |
| 8 | 4.6 | Create a circuit with 10 components where every component has a connection with every other component (erroneous). | User constructs a circuit and presses ‘build’ button. | Display an appropriate warning to the user. | ‘Circuit was built in an incorrect way.’ is displayed. | Error detected, circuit is rejected, and user is warned about their mistake.  **Pass** |
| 9 | 4.6 | Create a circuit with 20 components. | User constructs a circuit and presses ‘build’ button. | Output calculated circuit values. | Circuit values outputted. | Values calculated and displayed to the user.  **Pass** |

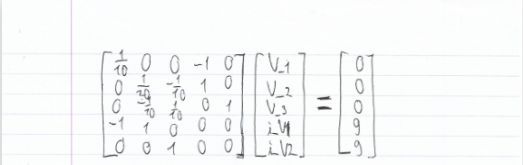
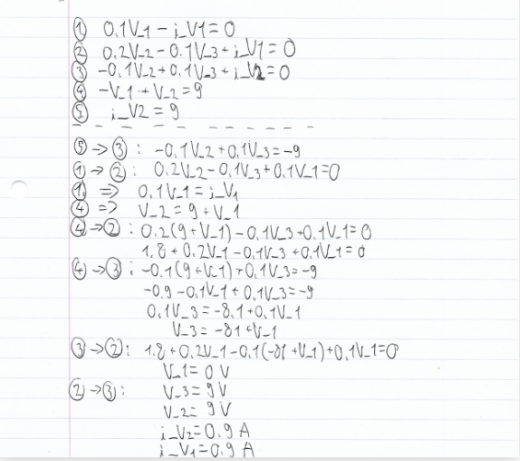
Evidence for test 1:

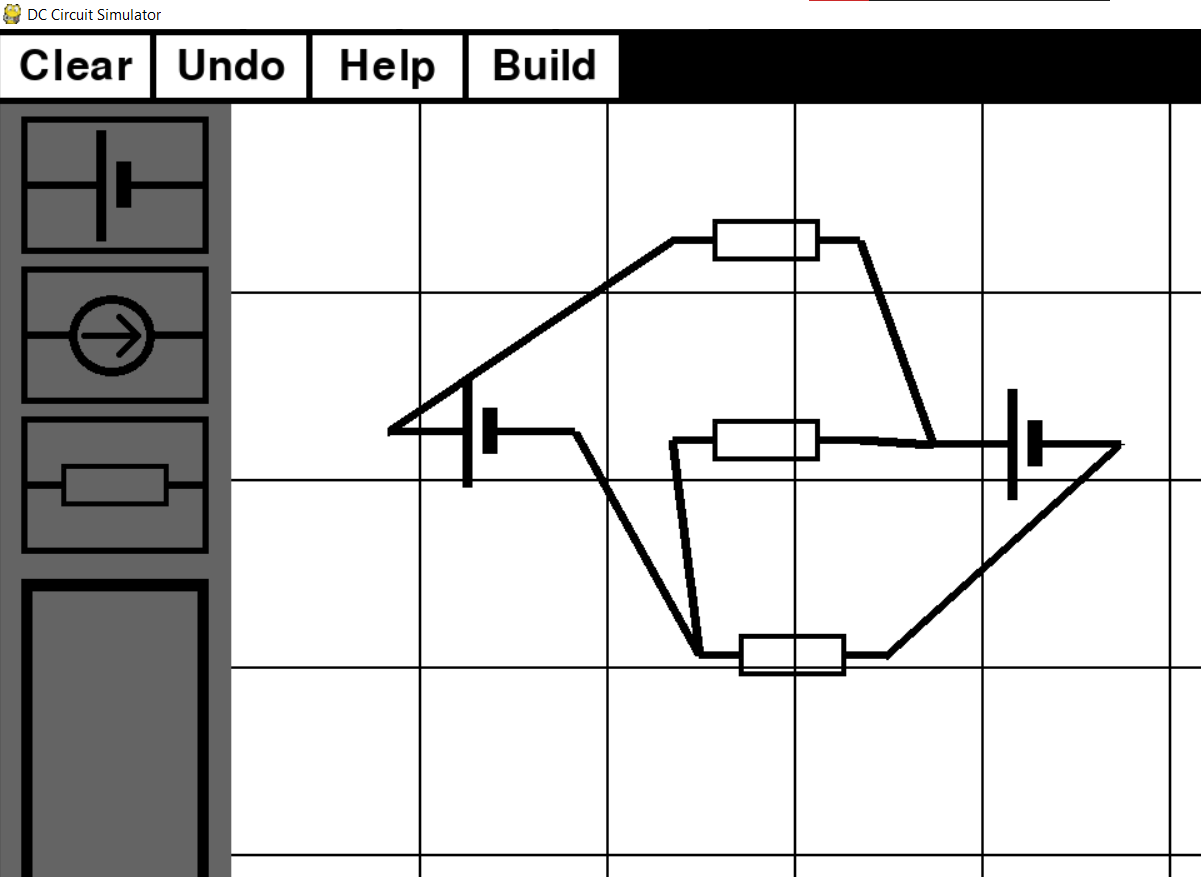
Netlist file: Parsed values: 

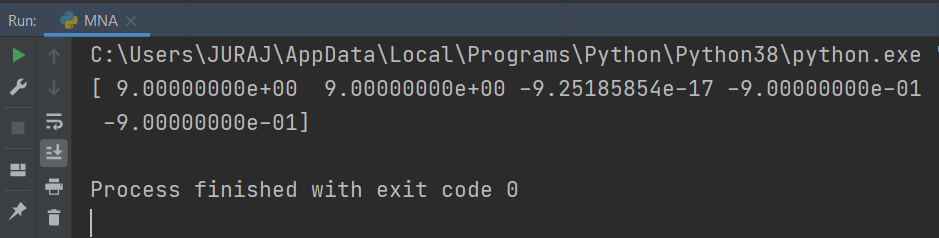
Evidence for test 2

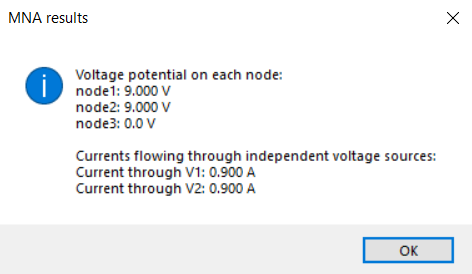
Example circuit with its matrix equation: 

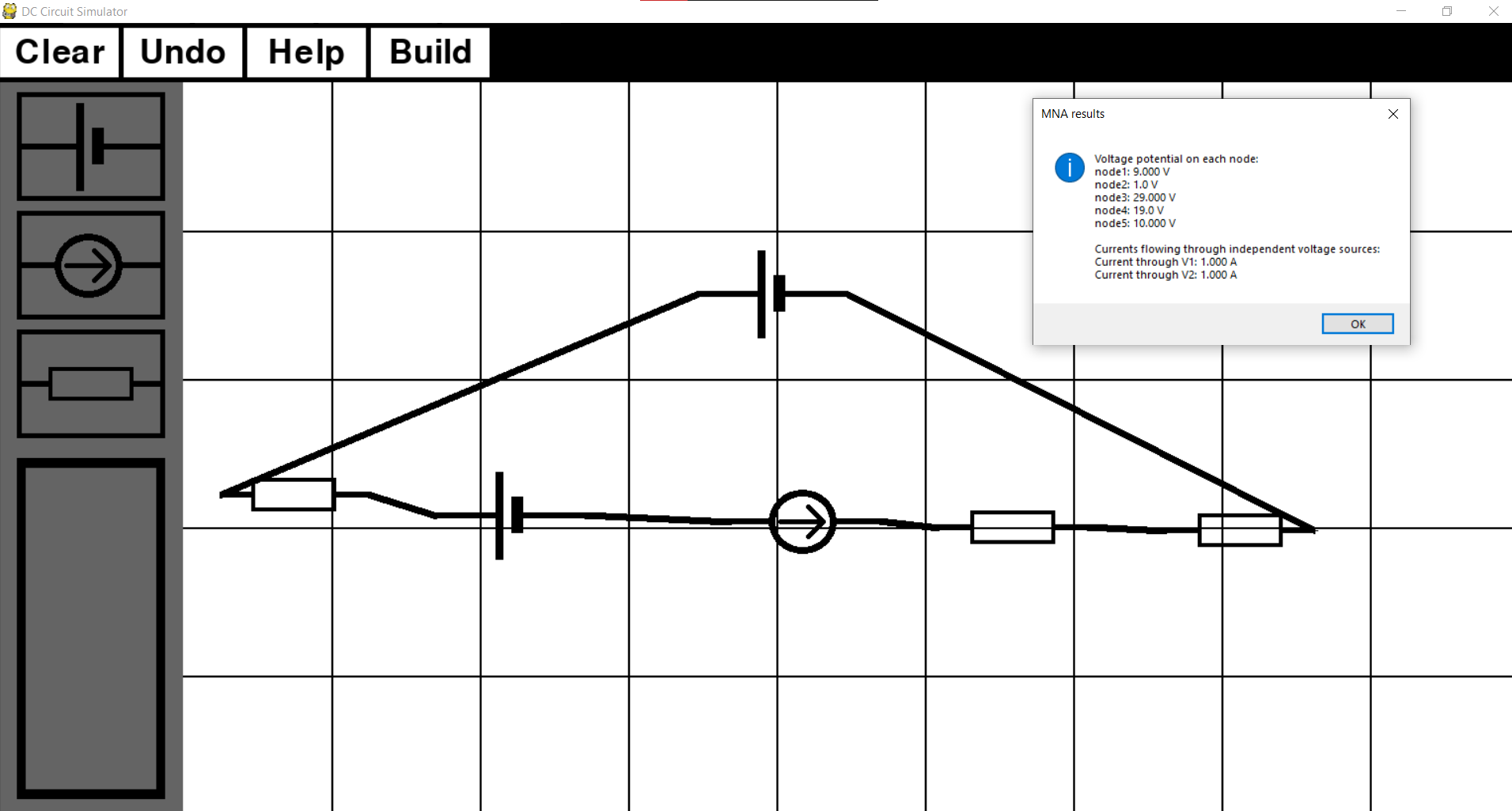
Lets us say that every resistor has value of 10 ohms and evrey independent voltage source has value of 9 volts. Below is a matrix calculation done by hand.

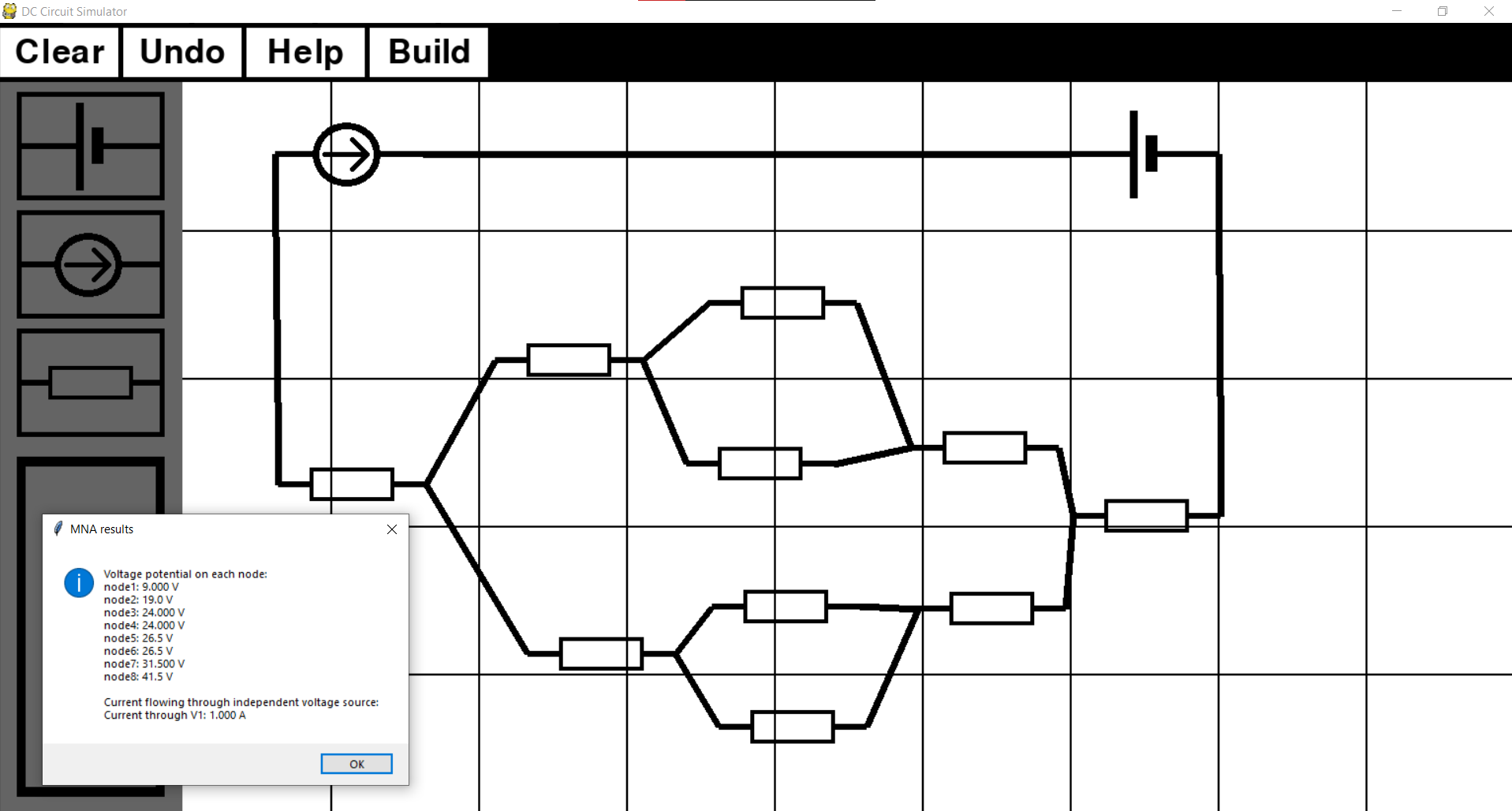
 

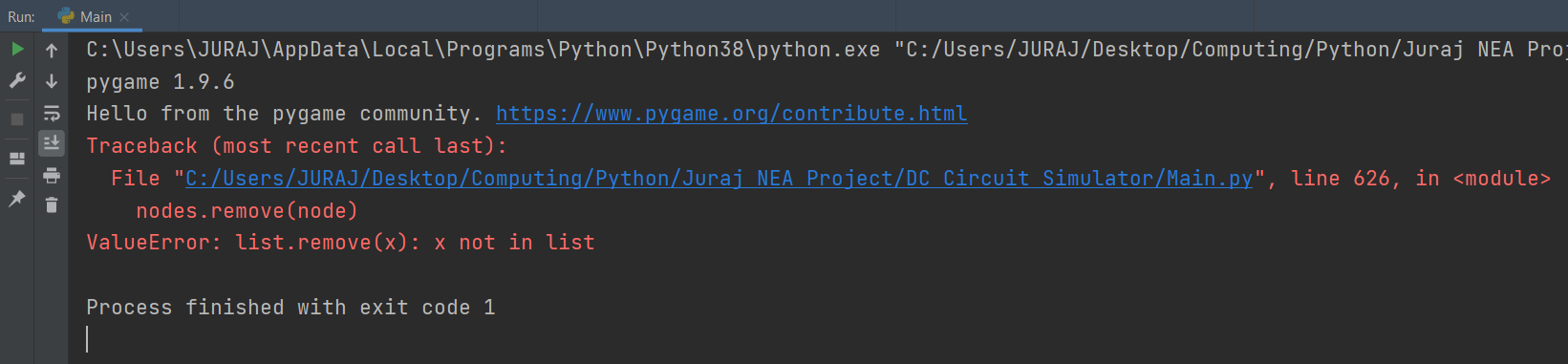
Screenshot of example circuit buit in the simulator: 

Screenshot of x matrix:

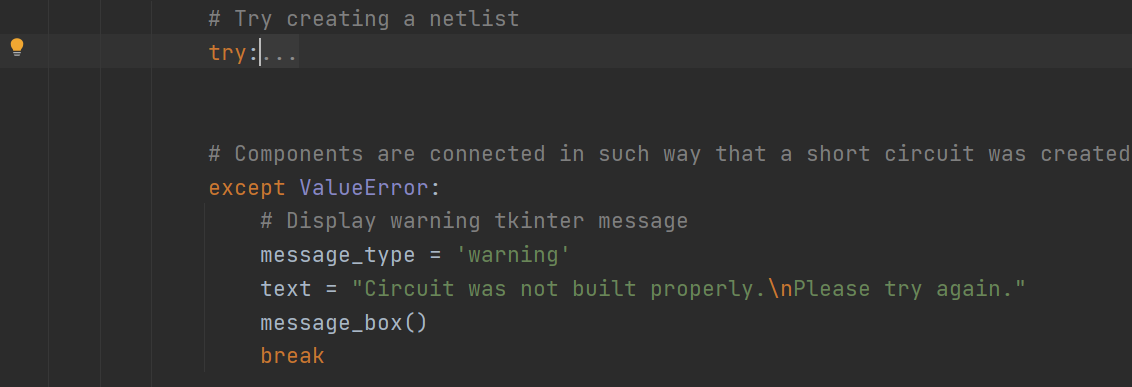
Screenshot evidence for test 3 (example circuit was used): 

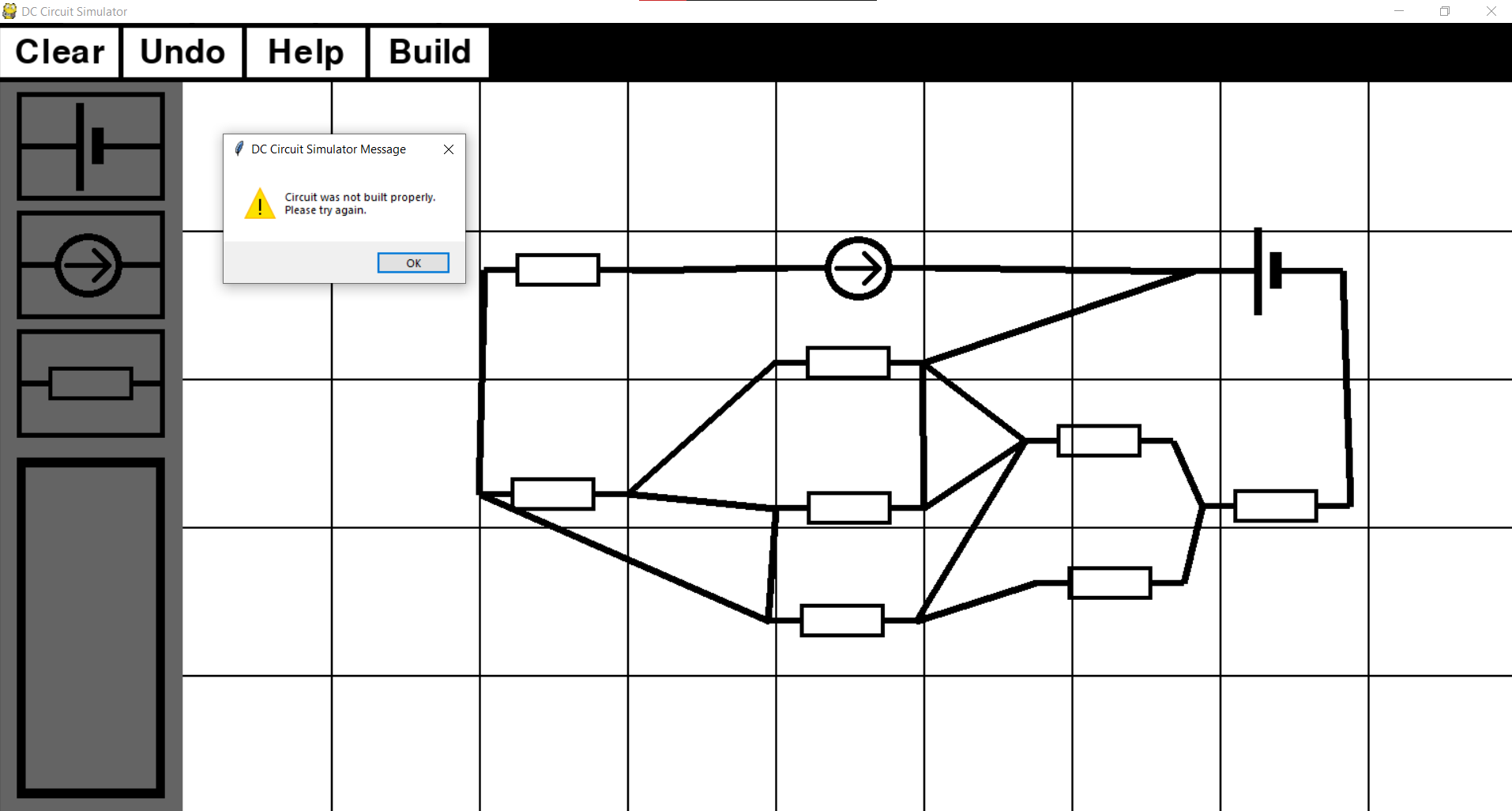
Screenshot evidence for test 4: 

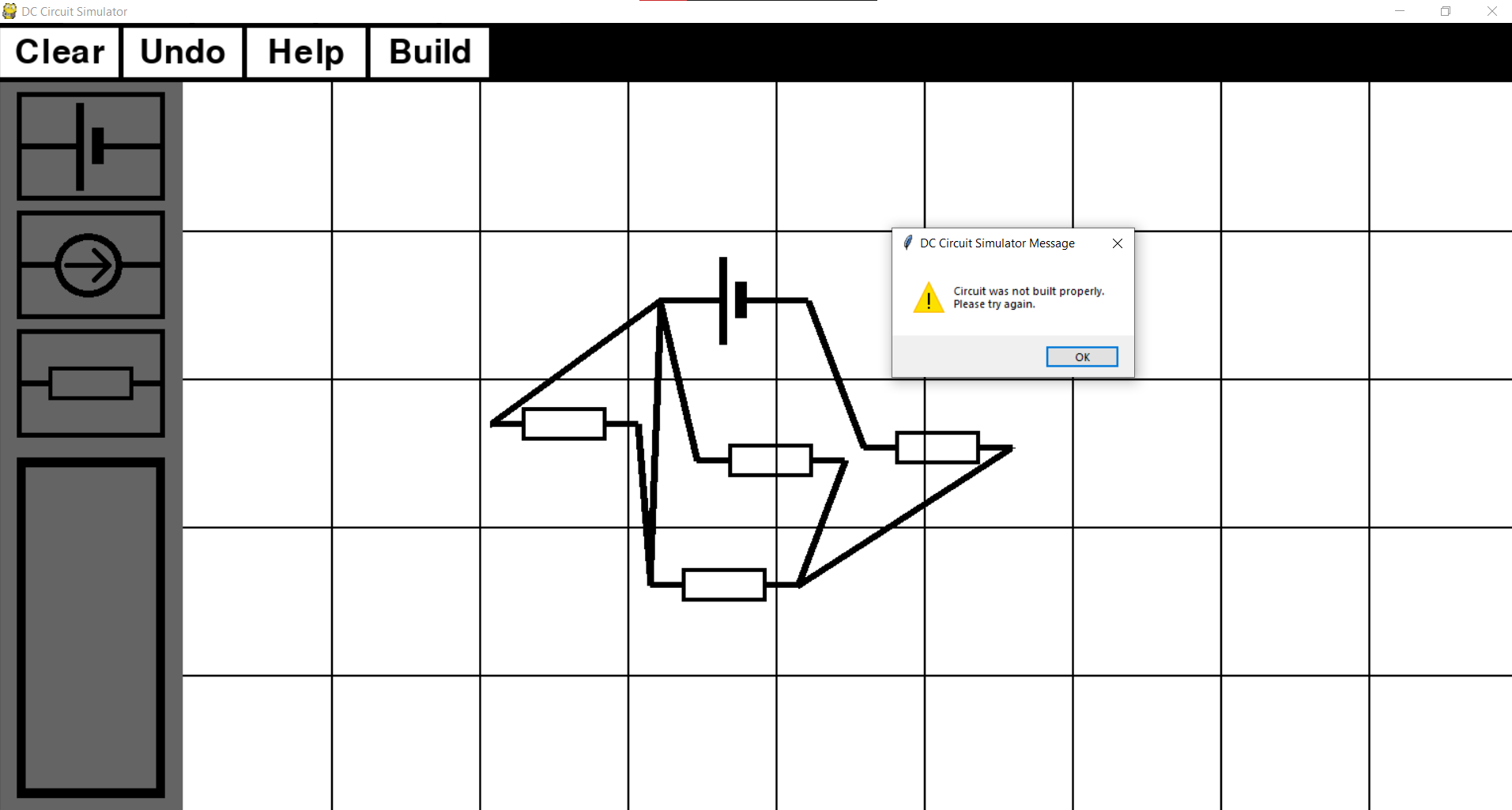
Screenshot evidence for test 5:

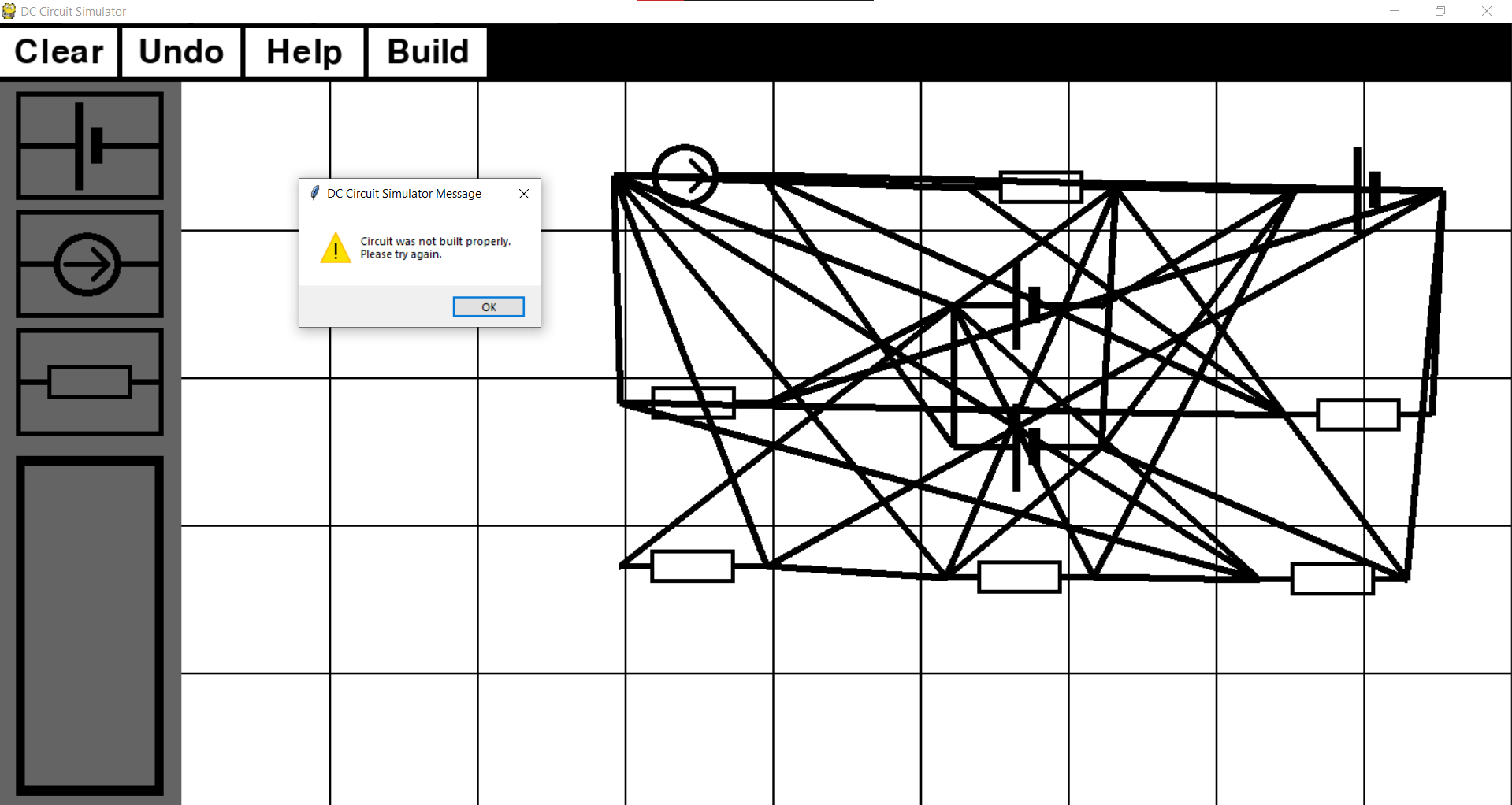
Screenshot evidence for test 6: 

To fix this error I will add try and except statement to the code:

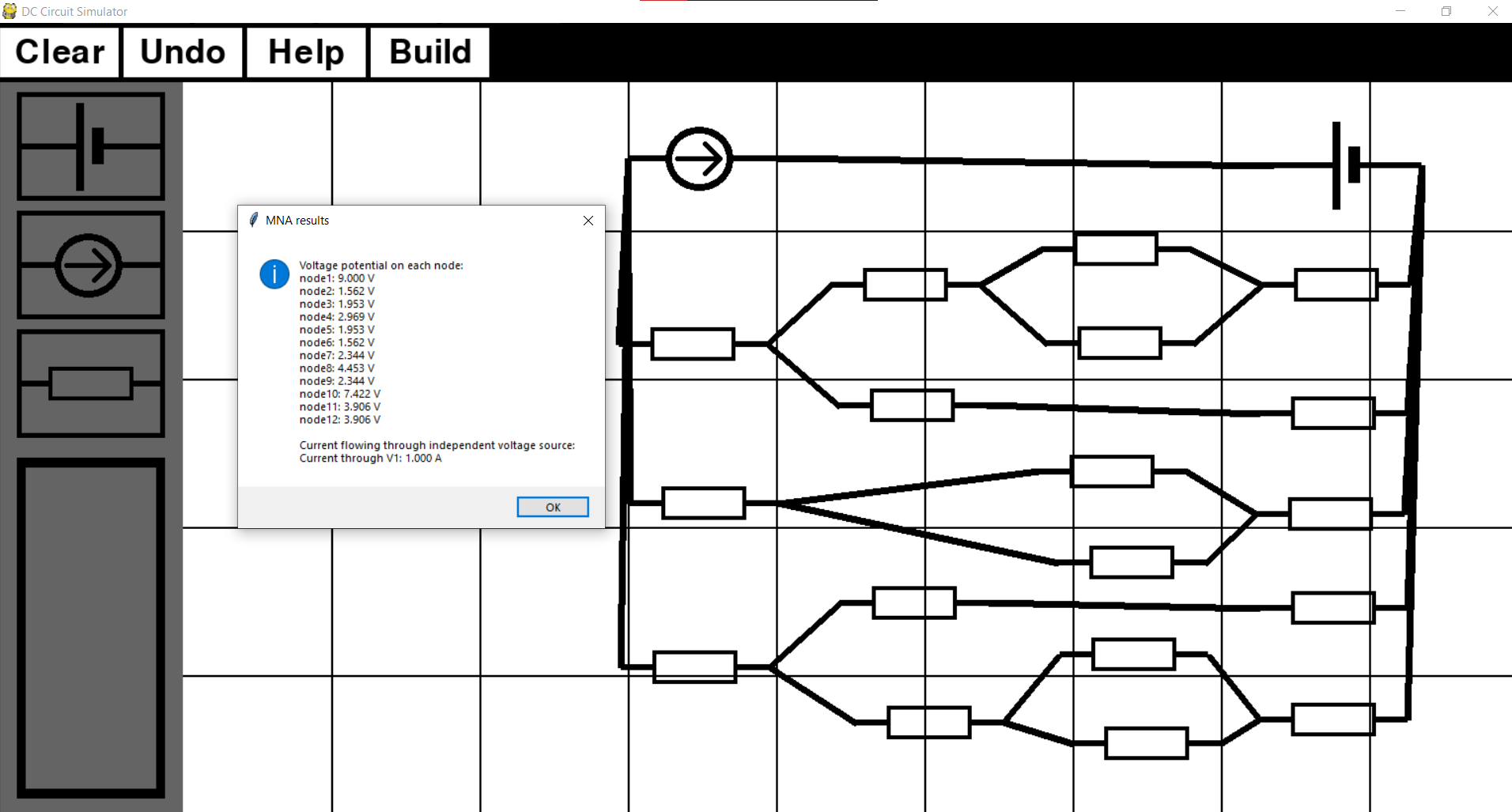


Screenshot with the new code: 

Screenshot evidence for test 7: 

Screenshot evidence for test 8: 

Screenshot evidence for test 9:



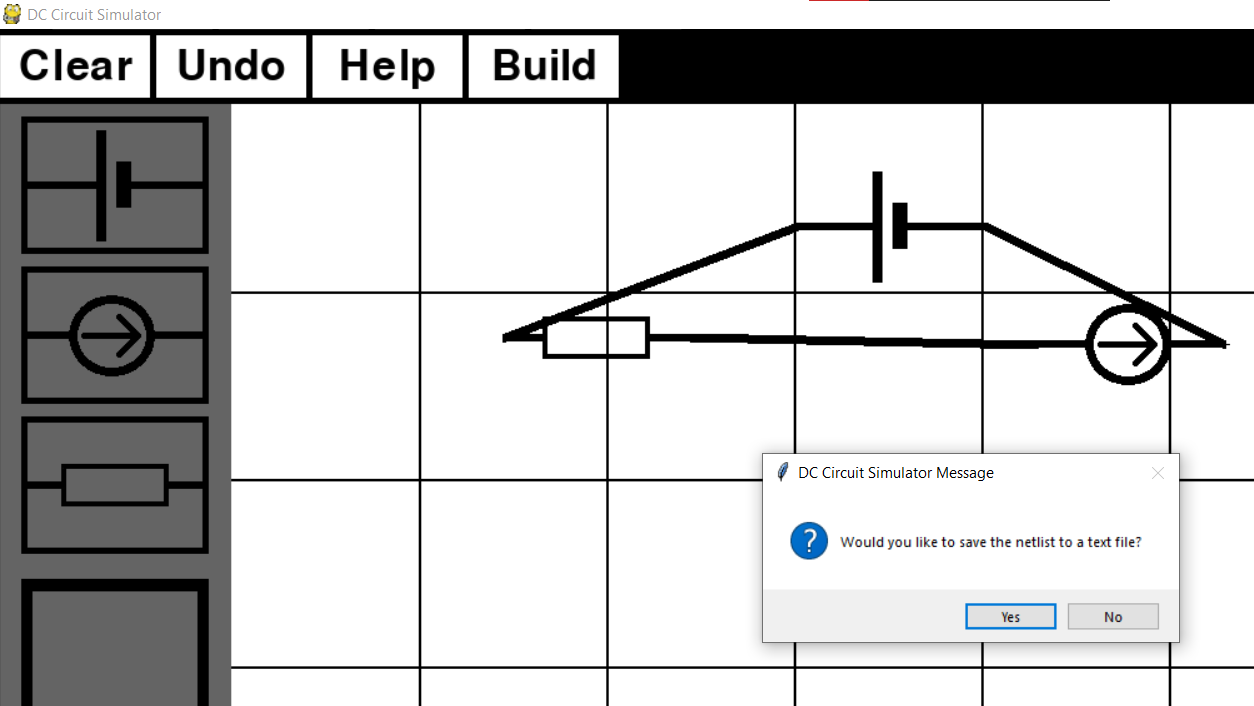
## Objective 5

1. *User will have an option to save the circuit once it was analysed. After the circuit’s netlist was created and stored in the external file, user will be asked if they would like to save the circuit. If the user chooses not to save it, the file containing the circuit’s information will be deleted after appropriate calculations have been made (MNA) and final results have been displayed to the user. If the user decides to save it, netlist information will be saved.*

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Test No.** | **Objective number being tested** | **Test description** | **User input** | **Expected outcome** | **Actual outcome** | **Comment on the outcome**  **(Pass/Fail)** |
| 1 | 5. | User needs to be asked if they would like to save the netlist file | No | File does not get saved | File was not saved | No saved file.  **Pass** |
| 2 | 5. | When user decides to save a file, file should be saved and they should be informed about the name of the file. | Yes | File gets unique title and gets saved | File was saved as ‘dc\_circuit1.txt’ | File saved.  **Pass** |

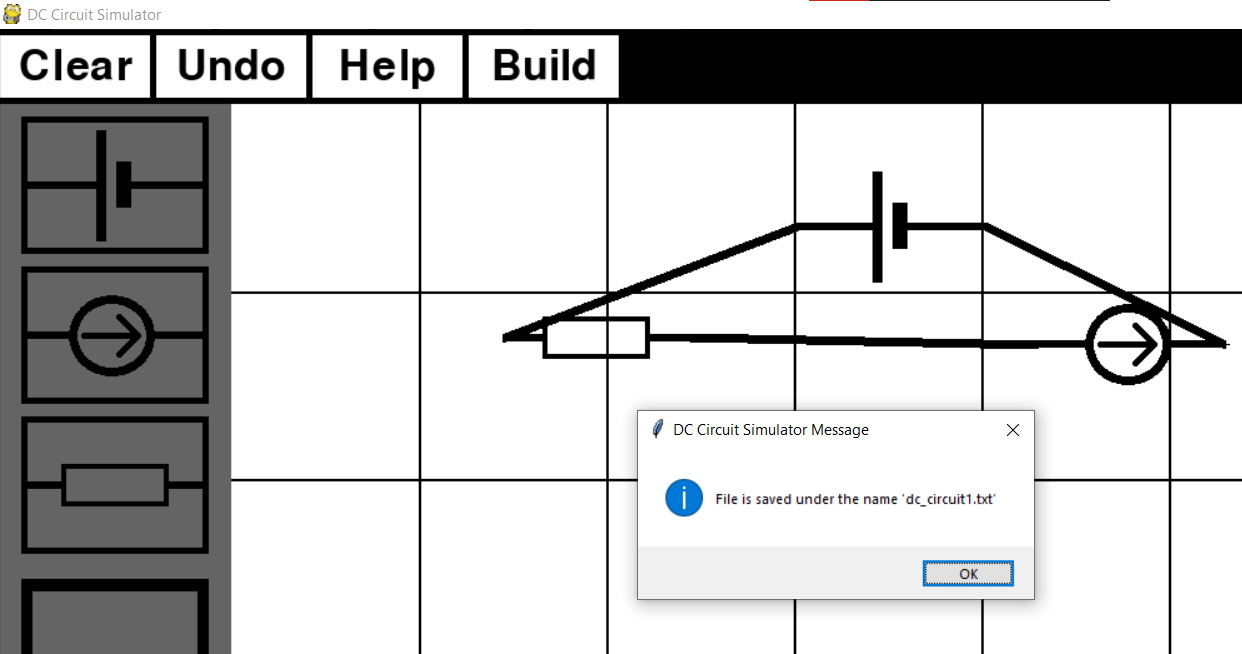
Screenshot evidence for test 1:

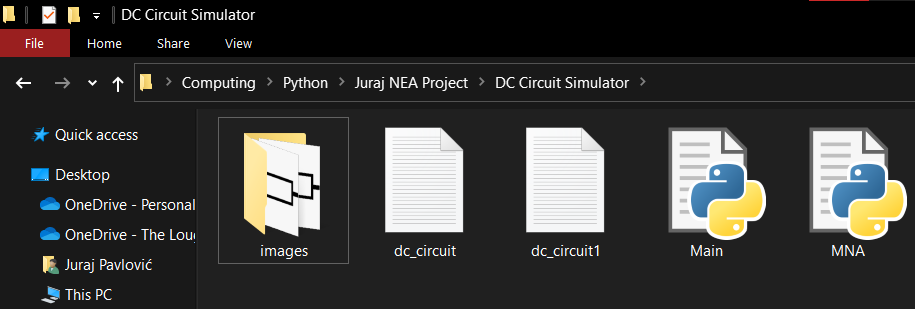
User was asked if they would like to save a file.



They answered ‘No’ so file didn’t get saved.

Screenshot evidence for test 2:

This time user answered ‘Yes’ so they were informed about file’s name. 

Screenshot of the saved file: 

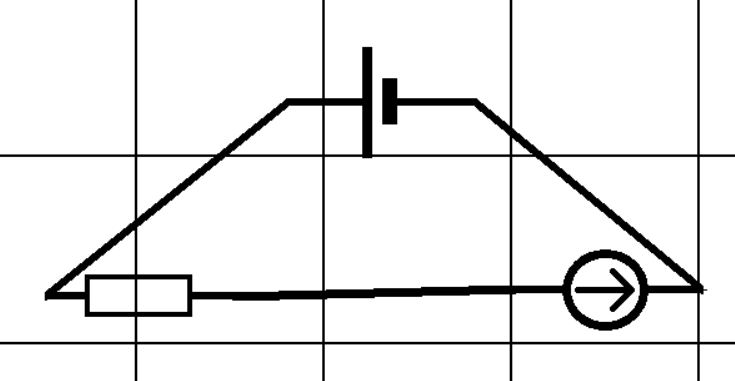
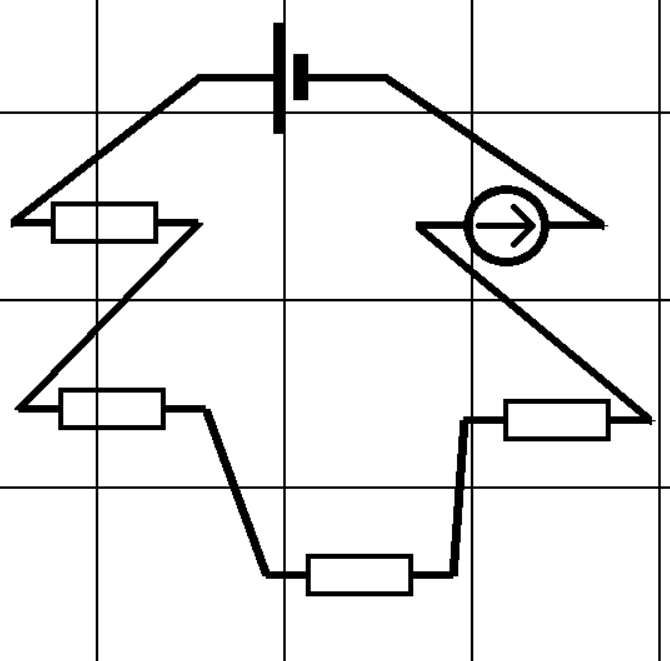
## Objective 6

1. *During MNA, once the matrices are constructed, they need to be stored using the sparse matrix optimisation techniques. This needs to reduce the overall time needed to perform all the calculations. To test it, I will be timing the program’s time to calculate the circuit outputs with and without the use of spare matrices.*

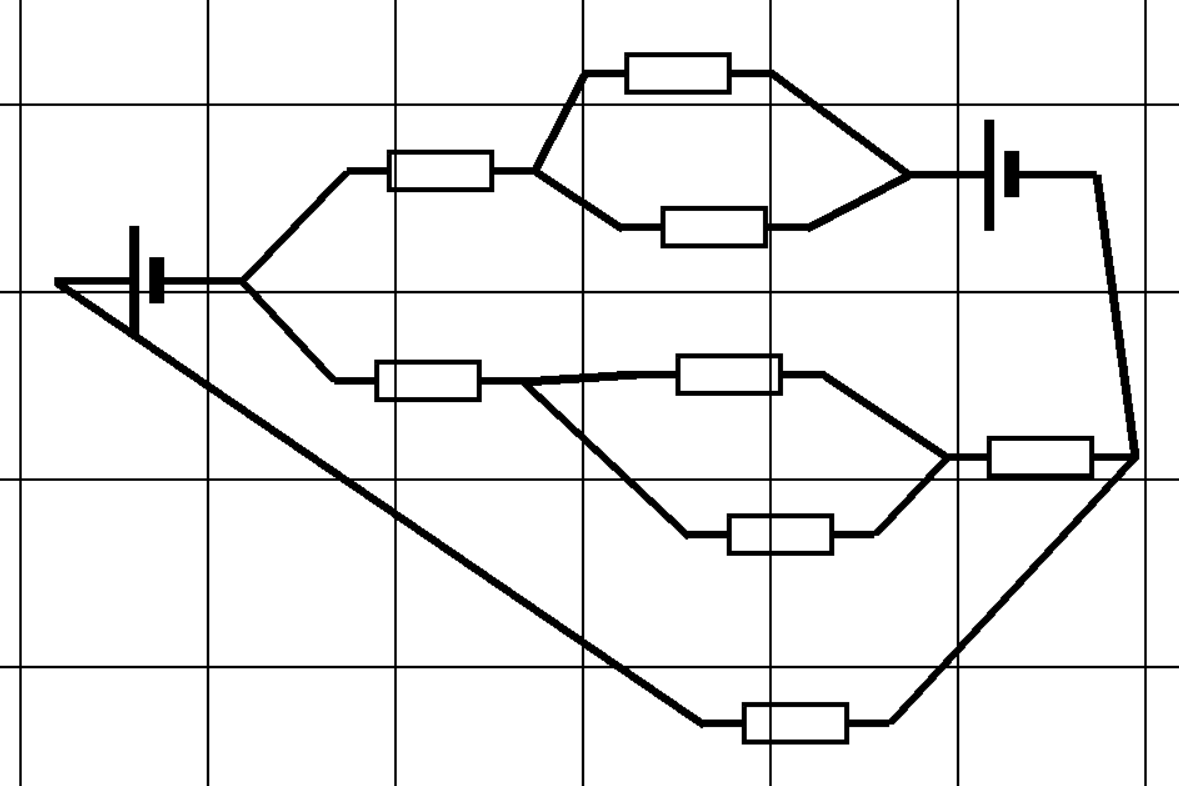
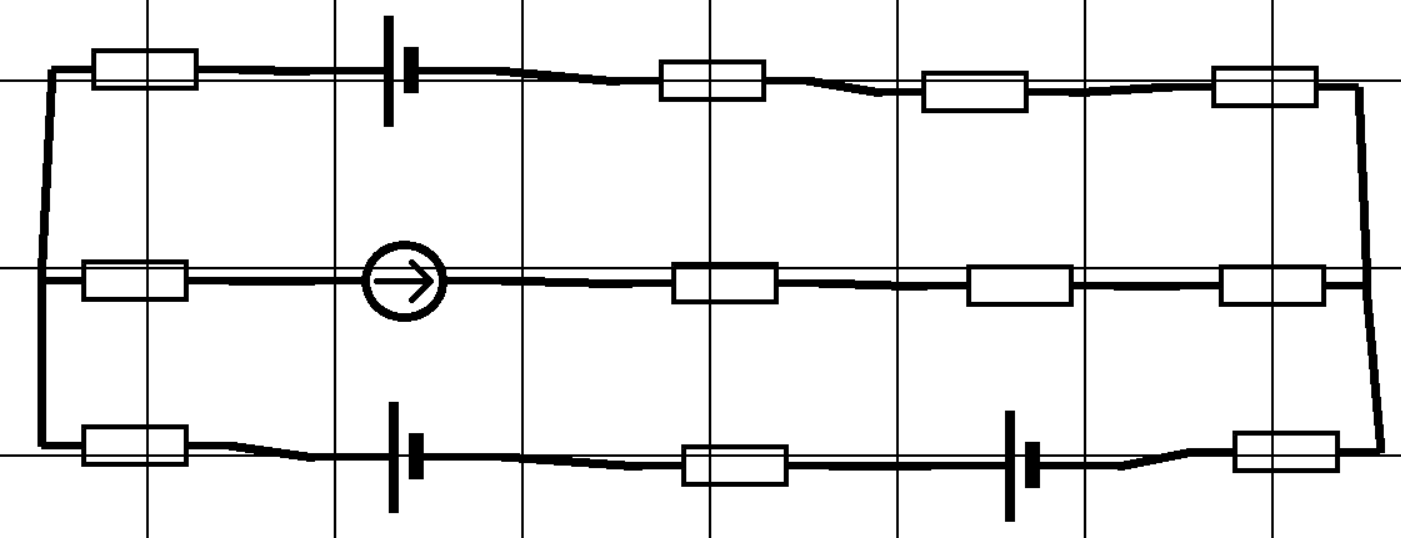
To check if using the sparse matrix optimalisation actually saves time when solving a circuit, I will create five different circuit and solve them first with the sparse matrix optimalisation and then without out. Both times, time the program took to finish will be measured.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Test No.** | **Objective number being tested** | **Test description** | **User input** | **Outcome** |
| 1 | 6. | Time taken to solve an optimised circuit with 3 components. | User creates a circuit and presses build button |  |
| 2 | 6. | Time taken to solve a normal circuit with 3 components. | User creates a circuit and presses build button |  |
| 3 | 6. | Time taken to solve an optimised circuit with 6 components. | User creates a circuit and presses build button |  |
| 4 | 6. | Time taken to solve a normal circuit with 6 components. | User creates a circuit and presses build button |  |
| 5 | 6. | Time taken to solve an optimised circuit with 10 components. | User creates a circuit and presses build button |  |
| 6 | 6. | Time taken to solve a normal circuit with 10 components. | User creates a circuit and presses build button |  |
| 7 | 6. | Time taken to solve an optimised circuit with 15 components. | User creates a circuit and presses build button |  |
| 8 | 6. | Time taken to solve a normal circuit with 15 components. | User creates a circuit and presses build button |  |
| 9 | 6. | Time taken to solve an optimised circuit with 25 components. | User creates a circuit and presses build button |  |
| 10 | 6. | Time taken to solve a normal circuit with 25 components. | User creates a circuit and presses build button |  |

Circuit 1: Circuit 2:

Circuit 3: Circuit 4:

Circuit 5

