DC Circuit Simulator:

**Design**

Juraj Pavlović

2020./2021.

Content

[Design 5](#_Toc67390861)

[IPSO chart 5](#_Toc67390862)

[System hierarchy structure 7](#_Toc67390863)

[System flow chart 11](#_Toc67390864)

[Classes table 11](#_Toc67390865)

[Data dictionary 12](#_Toc67390866)

[Validation 16](#_Toc67390867)

[Data type check 16](#_Toc67390868)

[Presence check 16](#_Toc67390869)

[Length check 17](#_Toc67390870)

[Key algorithms 17](#_Toc67390871)

[Drag and drop system 17](#_Toc67390872)

[Creating a network of components 18](#_Toc67390873)

[Main program loop 18](#_Toc67390874)

[Parsing a file 19](#_Toc67390875)

[Filling MNA matrices 19](#_Toc67390876)

[Solving MNA system 20](#_Toc67390877)

[Data structure 21](#_Toc67390878)

[Lists 21](#_Toc67390879)

[Tuples 21](#_Toc67390880)

[Numpy Arrays 21](#_Toc67390881)

[Hash tables 21](#_Toc67390882)

[File structure 22](#_Toc67390883)

[User interface design 23](#_Toc67390884)

[Visibility 26](#_Toc67390885)

[Affordance 26](#_Toc67390886)

[Feedback 27](#_Toc67390887)

[Minimizing the user’s memory load 27](#_Toc67390888)

[Speaking the user’s natural language 27](#_Toc67390889)

[Consistency 27](#_Toc67390890)

[Python modules 28](#_Toc67390891)

[PyGame 28](#_Toc67390892)

[Tkinter 28](#_Toc67390893)

[NumPy 28](#_Toc67390894)

[SciPy 29](#_Toc67390895)

[Security and integrity of the data 29](#_Toc67390896)

# 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

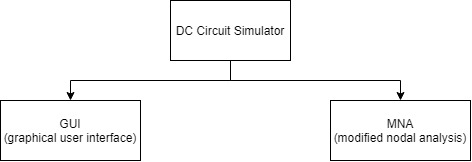
Below is a table showing systems main inputs, processes and outputs.

|  |  |  |  |
| --- | --- | --- | --- |
| **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 |

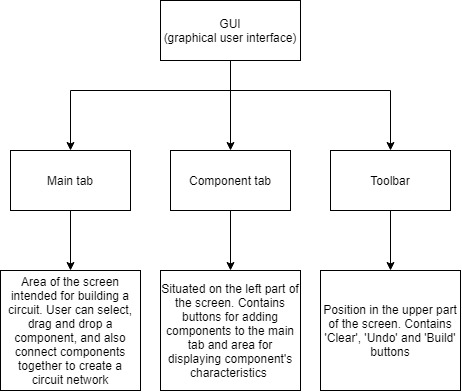
## System hierarchy structure

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

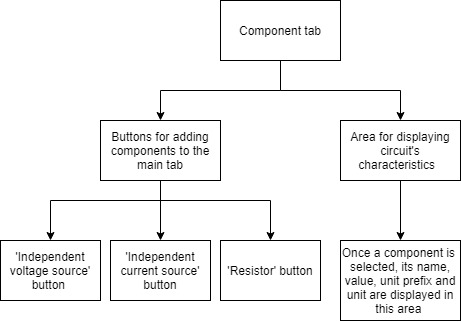
The simulator is split into two main parts: GUI and MNA. GUI gives the user an ability to construct a circuit and MNA then analyses the circuit and performs mathematical operations to obtain the final results.



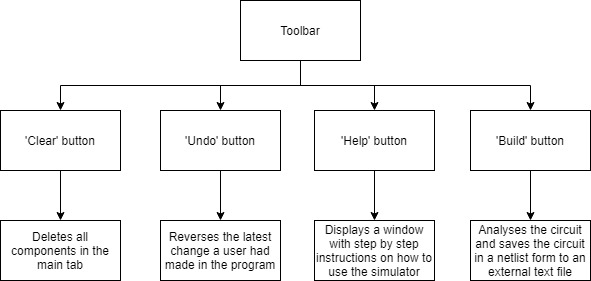
GUI is visibly split into three main parts on the screen.



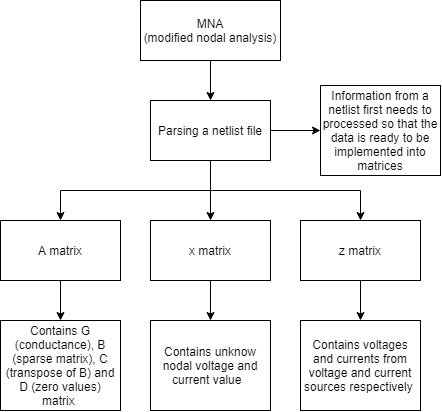
Component tab consists of two main parts both related to components which form a circuit. One part has buttons with icons of components. These are used to add components to the main tab. Other part is used to give the user more detailed information about each specific component.



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.

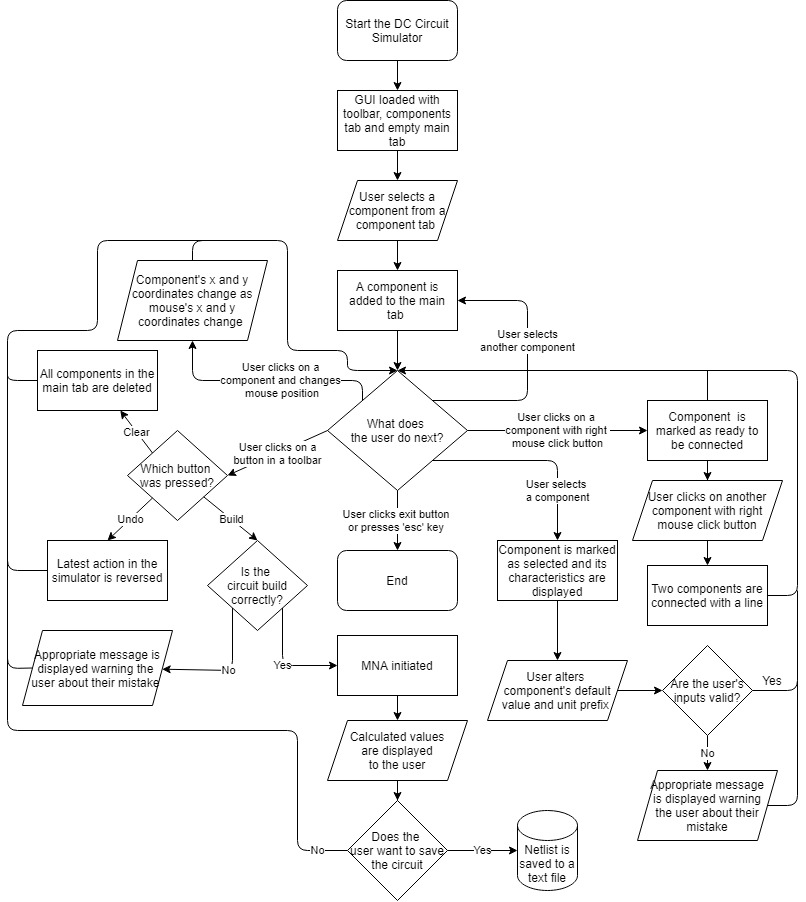


After building a circuit, MNA uses the netlist text file (file describing a circuit in a text format) to construct a linear equation with three matrices. Using the data from two matrices, two-dimensional A matrix and one-dimensional z matrix, a matrix with unknown values, x matrix, can be calculated.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



## Classes table

## 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 | 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 |
| G | Array | [0.25, -0.25, -0.25. 0.75] | Matrix containing conductance values in the circuit |
| B | Array | [1.0, 0.0] | Incidence matrix which is made up of 1s and 0s |
| C | Array | [0.0, 1.0] | Transpose of B matrix |
| D | Array | [0.0] | Matrix containing only zero values |
| z | Array | [0.0, 1.0, 9.0] | Matrix with values of voltage and current sources |
| x | 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

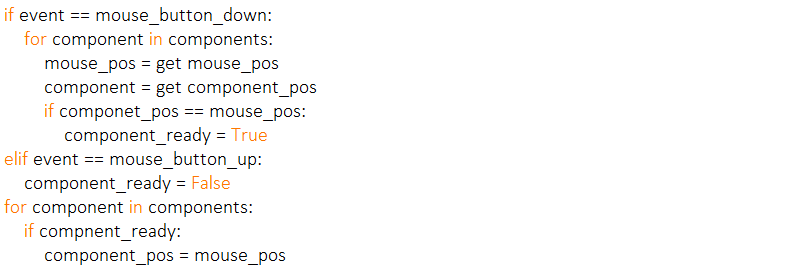
end if

end Function

## Key algorithms

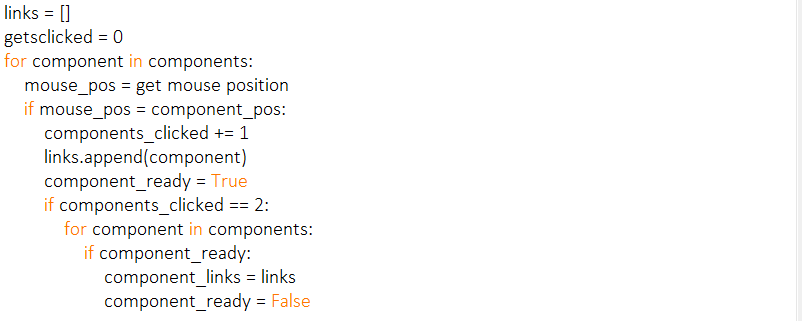
### Drag and drop system

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.



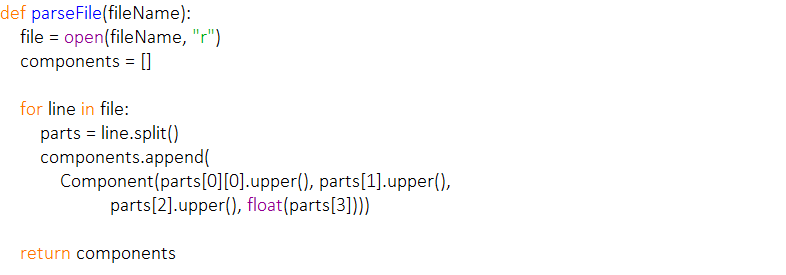
### Main program loop

This is how to game loop will be structured. Done is the stopping condition and will be set to true within the check inputs function if the user clicks the exit button. At this point the loop will end and so will the program.



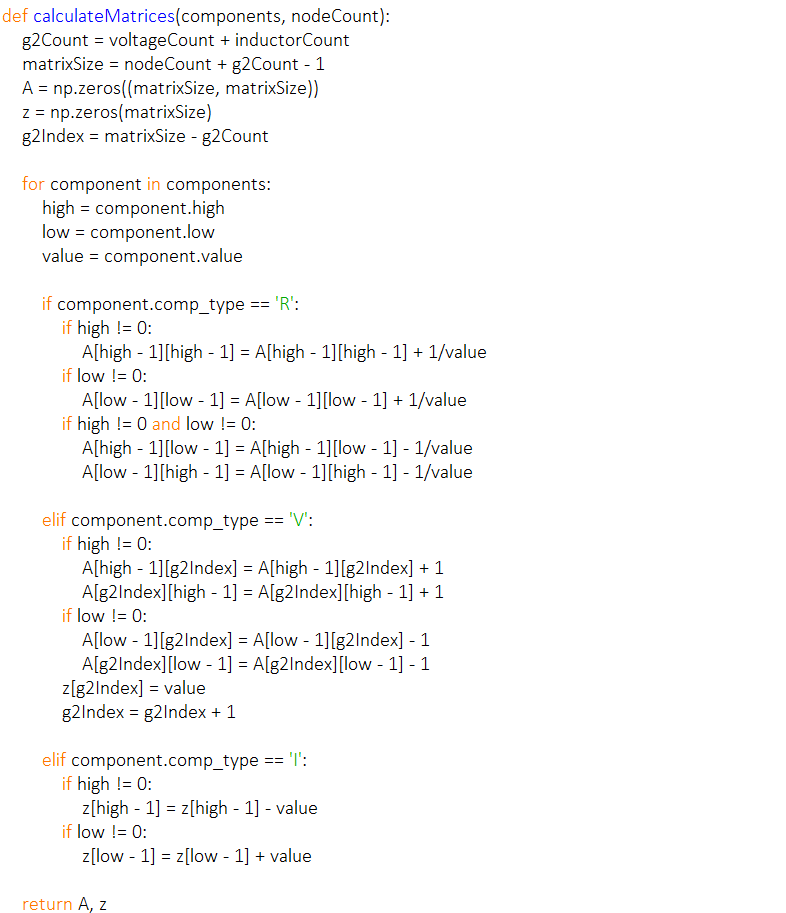
### Parsing a file

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.



### Filling MNA matrices

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 appropriate matrix with component’s values.



### Solving MNA system

Once the MNA matrix equation is obtained, it can easily be solved using the NumPy’s linalg.solve function which calculates unknown values of the system.



## 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.

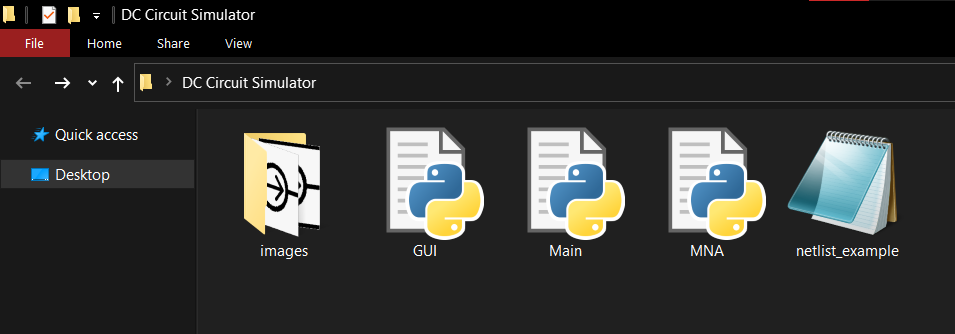
### Hash tables

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.

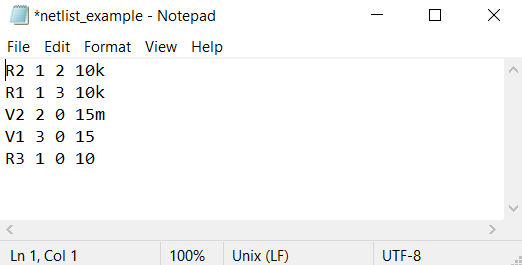
In the MNA system’s matrices each row and column correspond to a specific node, but in netlists these nodes are stored as strings. In order to use netlist’s string nodes, these strings need to mapped into specific integer. Since the computer does not know where exactly each node occurs in the netlist 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 consists of three python files with addition to extra text or image files. Python files will be called ‘Main’, ‘GUI’ and ‘MNA’. The Main.py file will include the main loop of the program. Running this file will start the program. GUI.py will consist of all information required to make the user interface. MNA.py will include all functions needed to carry out complex mathematical operations of the program.



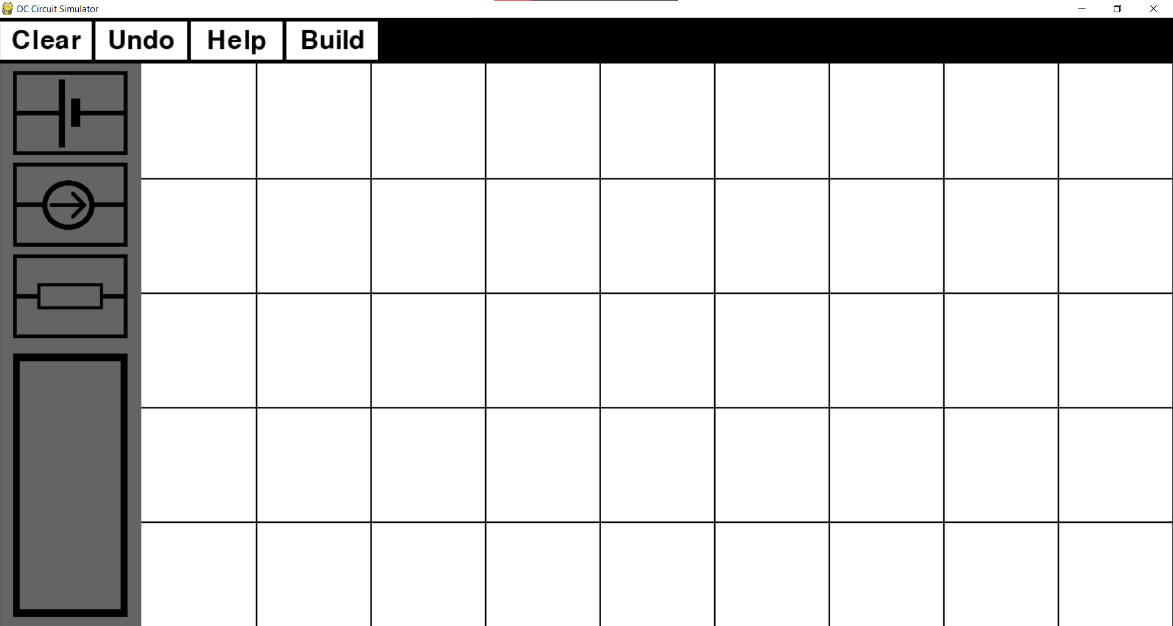
When a user decides to save a circuit layout, a new text file will be created which will contain the circuit’s netlist. An example of how this file will look like is below.



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.

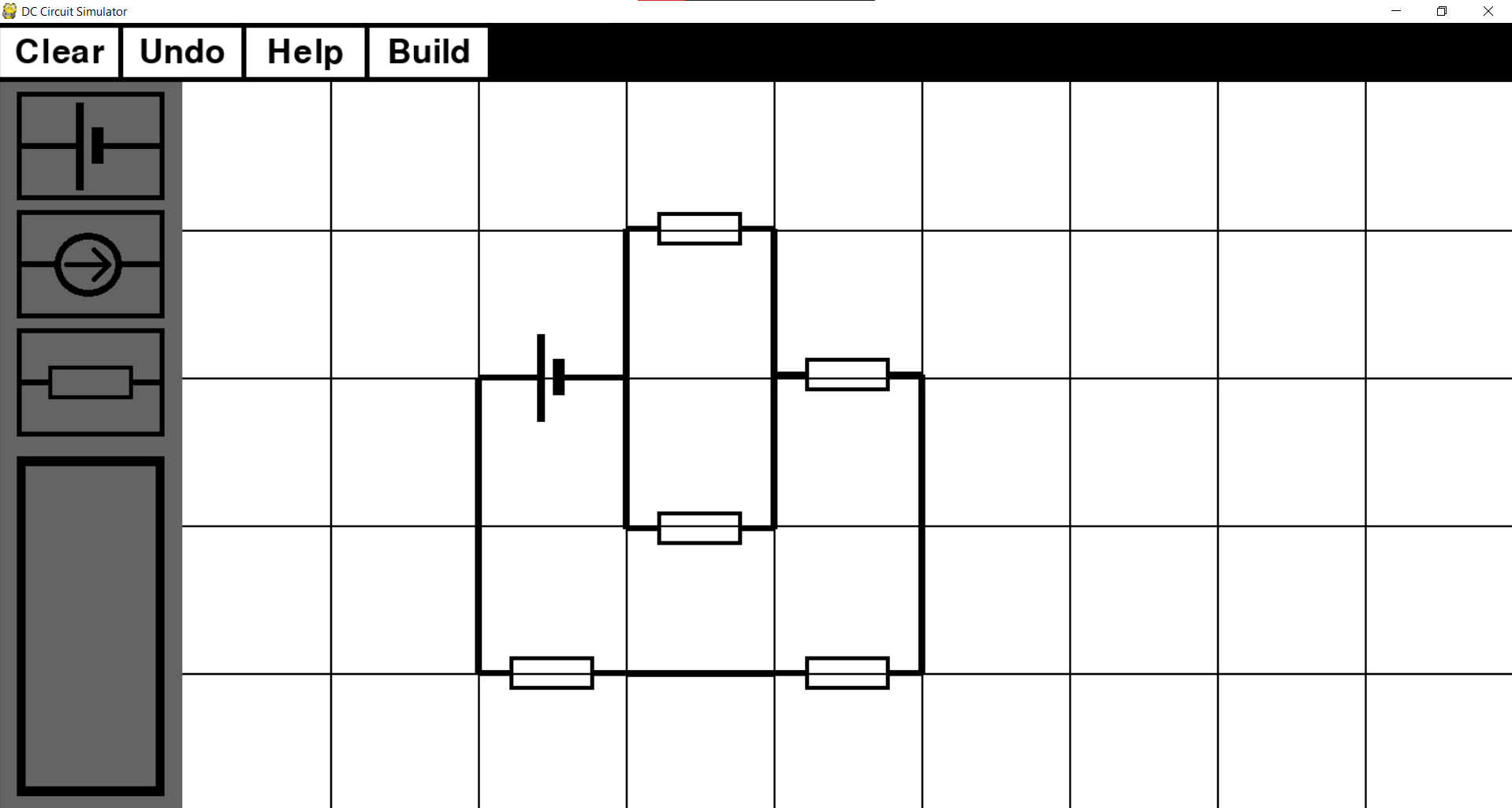


Toolbar

Component’s tab

Main tab

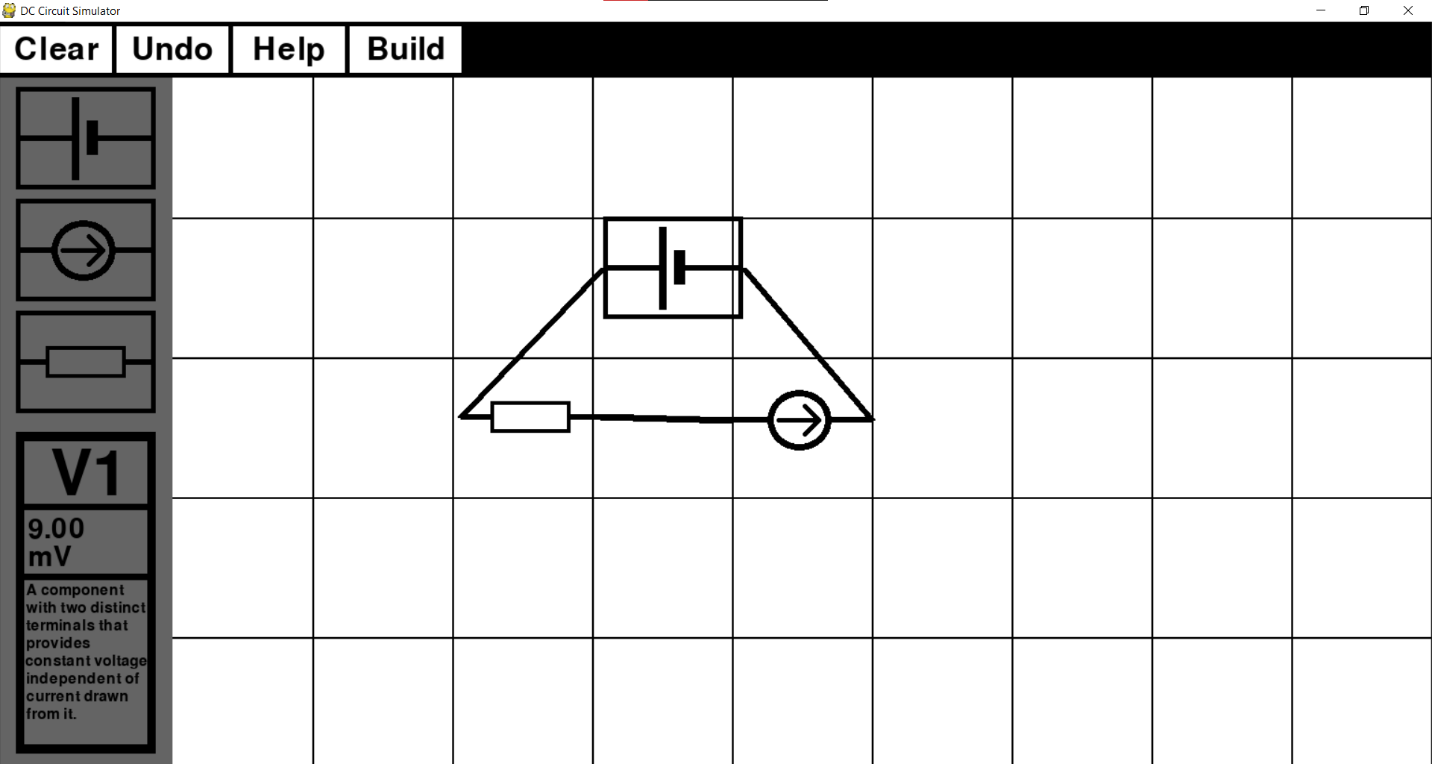
Main tab is a place on the screen where user can add components to, change their position, and connect them into a network. Below is a picture showing a GUI with a completed circuit in the main tab. The circuit consists of independent voltage source and resistors connected both in series and parallel.



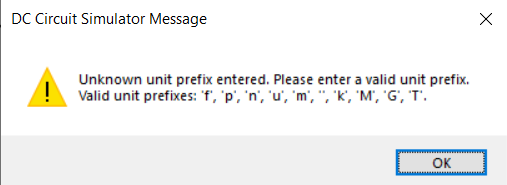
Toolbar includes four buttons on top of the screen:

* ‘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.

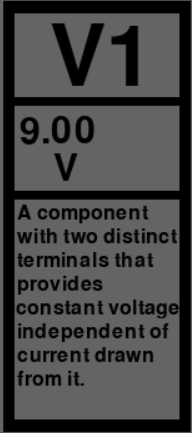
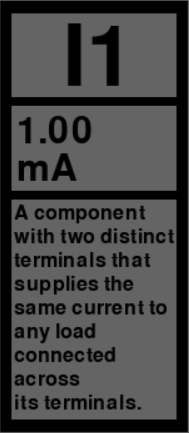
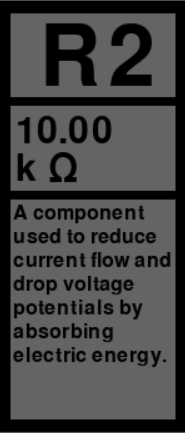
A grey area on the left side of the screen is where user can add components to the main tab by clicking on their icons. User can also see component’s characteristics which will appear in the grey rectangle with black sides. Below is an example of what a user will see if they select a component in the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

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.

### 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.

### 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.

## 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.