ELECTRICAL CIRCUIT SIMULATOR - THOMAS BLAIR

1 - Analysis

1.1 - INTRODUCTION TO THE PROBLEM

1.1.1 - PROBLEM DESCRIPTION

Due to physics being a challenging A-level, covering a wide array of topics in intricate detail, many students struggle with regards to carrying out work independently and understanding their teacher. Throughout the entire course and especially within the first year, electricity is considered one of the more difficult topics. This is because it requires a deep understanding of both theory and practical content as well as a great deal of critical thinking and visualisation of problems.

At Durham Sixth Form Centre (DSFC), physics teacher Mr Allan has noticed that students often struggle with understanding the electricity unit. He is interested in finding new methods to teach the topic to his 83 students across year 12 and year 13. During a meeting with Mr Allan, I was informed that he is looking to use computer aided learning tools to provide students with an interactive learning experience. With the problem clearly present, I began research on a solution.

1.1.2 - CURRENT SOLUTION

Currently, Mr Allan relies on past exam paper questions and handmade diagrams for teaching electrical theory. Practical demonstrations of circuits are only carried out when required by the A-level physics specification. As I was taught by Mr Allan for the first year of my physics course, I can comfortably say that (although past-paper and teacher-made questions were helpful) I and other students would have benefited from a more interactive learning experience. This would have been especially helpful whilst carrying out independent study.

To solve this ever-present issue, I have decided to develop an electrical circuit builder and simulator that can be downloaded onto a computer.

1.2 - USER IDENTIFICATION AND NEEDS

1.2.1 - STUDENT IDENTIFICATION AND NEEDS

The first type of end user would be a student at DSFC taking A-level physics in year 12 or year 13, in any class. More broadly, the software could be used by any physics student studying at a level lower than or equal to A-level. Students could use the software for a variety of purposes, such as:

- Checking over work during or outside of class when a mark scheme is unavailable. This ensures that students can learn from mistakes by understanding where they have gone wrong.
- Creating correct examples for notes that do not contain any errors. Since some examples made by students will not have a reliable worked solution, being able to validate their results with confidence is important.
- Gaining a better understanding of new topics within A-level electricity such as internal resistance, potential dividers and resistors in parallel. Seeing how different layouts and values affect a circuit will solidify the user's understanding and clarify any misconceptions they may have.
- Simulating practicals relating to circuits to comprehend the theory behind what they are investigating. In addition, this will allow a student to see how the results are expected to look, in order to identify any errors during their practical.

1.2.2 - TEACHER IDENTIFICATION AND NEEDS

The second type of end user is Mr Allan. However, as with the students, the software could be used by any teacher looking to teach electrical theory at a level lower than or equal to A-level. A teacher can use this software in ways such as:

- Demonstrating new electrical theory introduced at A-level, as well as solidifying understandings from GCSE physics. Visual learning techniques are known to be highly memorable and engaging, hence an interactive diagram will allow a teacher to get points across more reliably than a lecture or series of notes.
- Going through questions and examples with the class to better explain what is happening throughout a circuit. The ability to change a component's properties will let a teacher show students how adjustments to component variables will affect a circuit.
- Demonstrating practicals that the class are expected to carry out independently. This is detrimental, as it can prevent mistakes from students, saving time and resources.
- Preparing questions for a lesson by simulating a circuit and tasking students with finding currents, voltages or resistances across components or explaining why the circuit is behaving as it is.

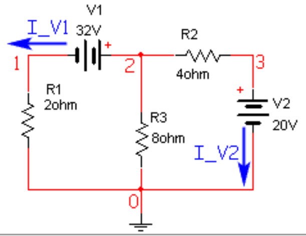
1.3 - RESEARCH METHODS

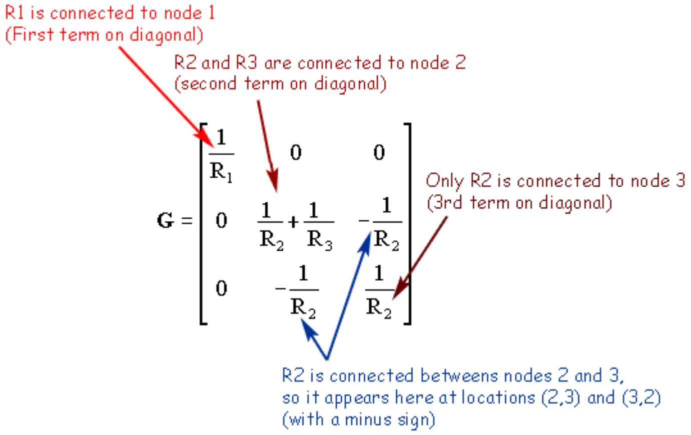
1.3.1 - RESEARCHING MATHEMATICAL THEORY

To simulate an electrical circuit, a computer must solve the simultaneous equations formed by Kirchoff's laws. My chosen method of doing so is Modified Nodal Analysis (MNA). As a brief introduction to this process, I have read and applied a guide published by Swarthmore College, the link and QR code for which is shown below:

https://lpsa.swarthmore.edu/Systems/Electrical/mna/MNA1.html

This guide details how a circuit can be converted into a series of simultaneous equations. This system can then be solved to find the current through each voltage source and the voltage at each node. To calculate the current through components once the equations are solved, I will develop an algorithm which will use the resistance of the component, and the voltage across it to do so. Shown on the following pages are screenshots of some examples detailed in the guide:

Case 1	
Circuit Diagram	MNA Equations
 V1 32V I_V1 + 1 2 R2 4ohm 3 R1 2ohm R3 8ohm + V2 20V I_V2 0	$\mathbf{G} = \begin{bmatrix} \frac{1}{R_1} & 0 & 0 \\ 0 & \frac{1}{R_2}+\frac{1}{R_3} & -\frac{1}{R_2} \\ 0 & -\frac{1}{R_2} & \frac{1}{R_2} \end{bmatrix}$



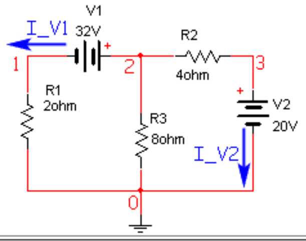
R1 is connected to node 1 (First term on diagonal)

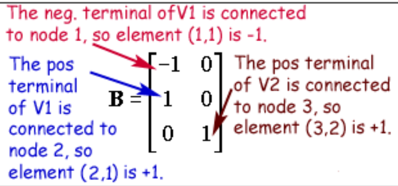
R2 and R3 are connected to node 2 (second term on diagonal)

$$\mathbf{G} = \begin{bmatrix} \frac{1}{R_1} & 0 & 0 \\ 0 & \frac{1}{R_2}+\frac{1}{R_3} & -\frac{1}{R_2} \\ 0 & -\frac{1}{R_2} & \frac{1}{R_2} \end{bmatrix}$$

Only R2 is connected to node 3 (3rd term on diagonal)

R2 is connected betweens nodes 2 and 3, so it appears here at locations (2,3) and (3,2) (with a minus sign)

Case 1	
Circuit Diagram	MNA Equations
 V1 32V I_V1 + 1 2 R2 4ohm 3 R1 2ohm R3 8ohm + V2 20V I_V2 0	$\mathbf{B} = \begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$

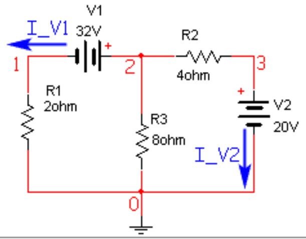


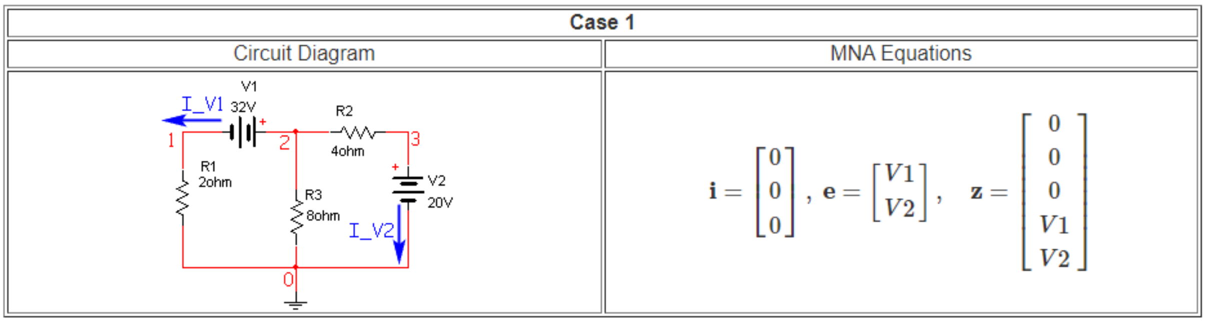
The neg. terminal ofV1 is connected to node 1, so element (1,1) is -1.

The pos terminal of V1 is connected to node 2, so element (2,1) is +1.

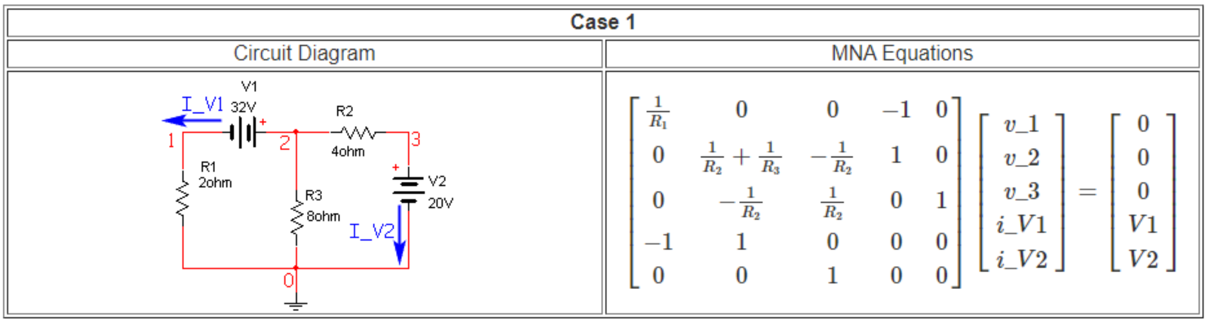
$$\mathbf{B} = \begin{bmatrix} -1 & 0 \\ 1 & 0 \\ 0 & 1 \end{bmatrix}$$

The pos terminal of V2 is connected to node 3, so element (3,2) is +1.

Case 1	
Circuit Diagram	MNA Equations
 V1 32V I_V1 + 1 2 R2 4ohm 3 R1 2ohm R3 8ohm + V2 20V I_V2 0	$\mathbf{v} = \begin{bmatrix} v\_1 \\ v\_2 \\ v\_3 \end{bmatrix}, \mathbf{j} = \begin{bmatrix} i\_V1 \\ i\_V2 \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} \mathbf{v} \\ \mathbf{j} \end{bmatrix} = \begin{bmatrix} v\_1 \\ v\_2 \\ v\_3 \\ i\_V1 \\ i\_V2 \end{bmatrix}$



Case 1	
Circuit Diagram	MNA Equations
	$\mathbf{i} = \begin{bmatrix}0\\0\\0\end{bmatrix}, \ \mathbf{e} = \begin{bmatrix}V1\\V2\end{bmatrix}, \quad \mathbf{z} = \begin{bmatrix}0\\0\\0\\V1\\V2\end{bmatrix}$



Case 1	
Circuit Diagram	MNA Equations
	$\begin{bmatrix} \frac{1}{R_1} & 0 & 0 & -1 & 0 \\ 0 & \frac{1}{R_2}+\frac{1}{R_3} & -\frac{1}{R_2} & 1 & 0 \\ 0 & -\frac{1}{R_2} & \frac{1}{R_2} & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v\_1 \\ v\_2 \\ v\_3 \\ i\_V1 \\ i\_V2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ V1 \\ V2 \end{bmatrix}$

It is important to note that the guide simply shows the rules for populating the matrix. Developing an algorithm to represent the circuit, populate the matrices and solve them is not included.

I have attempted to use MNA by hand to solve some circuits with loose connections and internal resistances, in order to plan how a computer could carry the operations out. From these tests I have discovered that the method fails when dealing with resistors with no resistance or incomplete circuits, and that I will have to keep this in consideration when designing my program. Furthermore, the guide only considers circuits without cells in parallel or multiple unconnected circuits. These must be considered when designing my algorithm in order to simulate any A-level circuit.

1.3.2 - RESEARCHING SOLVING OF MATRICES

In order to solve a linear system of equations, the entire system can be represented as a single matrix by concatenating the two matrices with no unknown variables. This matrix can then be manipulated to calculate the value of each unknown variable. To explore ways to solve this problem, I watched a short video made by ‘The Organic Chemistry Tutor’, the link and QR code of which can be found below:

https://www.youtube.com/watch?v=eDb6iugi6Uk

The video explains how a matrix representing a system of linear equations can be solved by a human through examples, whilst explaining the row operations that can be carried out. In the design section, I will develop an algorithm that can automate this task for a matrix with an equal number of unknowns and equations.

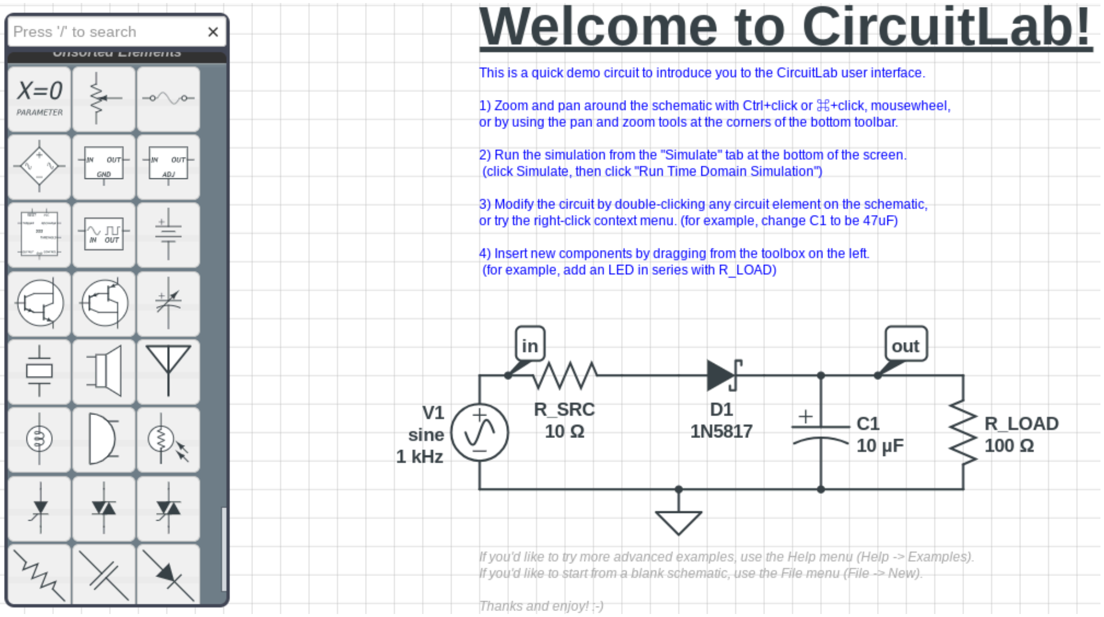
$$\begin{aligned} x + y - z &= -2 \\ 2x - y + z &= 5 \\ -x + 2y + 2z &= 1 \end{aligned} \longrightarrow \left[\begin{array}{ccc|c} 1 & 1 & -1 & -2 \\ 2 & -1 & 1 & 5 \\ -1 & 2 & 2 & 1 \end{array}\right]$$

Shown above is a matrix formed from a set of equations, an example from the video. From this, row operations such as multiplication by scalar values and addition with other rows can transform the matrix into row echelon form like so:

$$\left[\begin{array}{ccc|c} 2 & 1 & -1 & 1 \\ 0 & 1 & 5 & 17 \\ 0 & 0 & -13 & -39 \end{array}\right]$$

1.3.3 - RESEARCHING OTHER SOLUTIONS

Current simulators on the internet are not catered to A-Level physics due to the abundance of complex components and lack of information catered towards their studies. This makes them confusing and difficult to use. Shown below is the user interface for “CircuitLab”, an electrical circuit simulator which is free to use in a browser:



Welcome to CircuitLab!

This is a quick demo circuit to introduce you to the CircuitLab user interface.

1) Zoom and pan around the schematic with Ctrl+click or ⌘+click, mousewheel, or by using the pan and zoom tools at the corners of the bottom toolbar.

2) Run the simulation from the "Simulate" tab at the bottom of the screen. (click Simulate, then click "Run Time Domain Simulation")

3) Modify the circuit by double-clicking any circuit element on the schematic, or try the right-click context menu. (for example, change C1 to be 47uF)

4) Insert new components by dragging from the toolbox on the left. (for example, add an LED in series with R_LOAD)

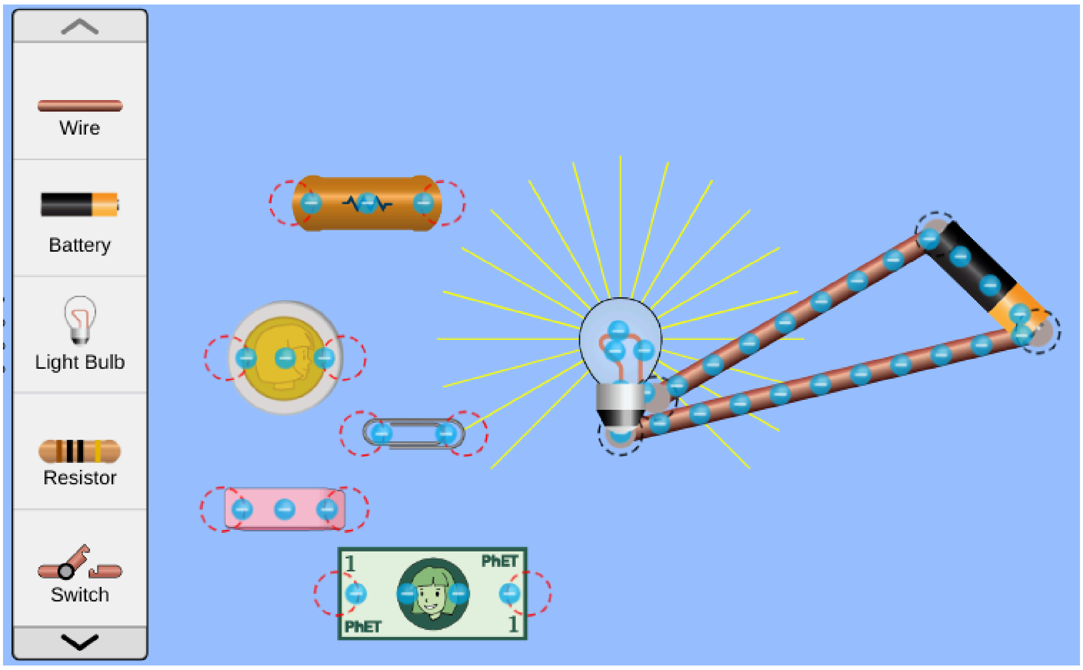
If you'd like to try more advanced examples, use the Help menu (Help -> Examples).
If you'd like to start from a blank schematic, use the File menu (File -> New).

Thanks and enjoy! :-)

As you can see, there is a plethora of complex components, most of which are only studied at an undergraduate level or even postgraduate level. CircuitLab is clearly catered to users with more complex requirements, making it too convoluted for users with a lower understanding of electric circuits. The addition of components related to electronics such as logic gates further clutters the user interface.

Furthermore, the user must add a ground source to the circuit, a feature of which we do not consider at our level. However, I plan to develop a system to add ground sources automatically, in order to prevent unnecessary information being shown to the user.

Another circuit simulator is “phET”, a free to use web application. Shown below is a screenshot of the user interface:



Unlike CircuitLab, phET does not contain complex components, meaning an A-level physics student could understand how each component works. However the lack of standardised symbols and pictures as well as the unnecessary components such as coins and paper clips make it difficult to focus on the theory behind them.

1.4 - ACCEPTABLE LIMITATIONS

The program would not be able to take the heating effect of components over time or internal resistance of wires into consideration, since the computations would have to be consistently carried out. This will not be relevant for most uses of the simulator, as exam style questions involving calculations mainly ignore these effects. Practicals will also be able to be simulated despite the lack of random error. Making the program needlessly complex to include these effects would distract users and affect the performance of the program. The focus of development should be on creating a GUI and an effective MNA algorithm.

1.5 - ANALYSIS TECHNIQUES

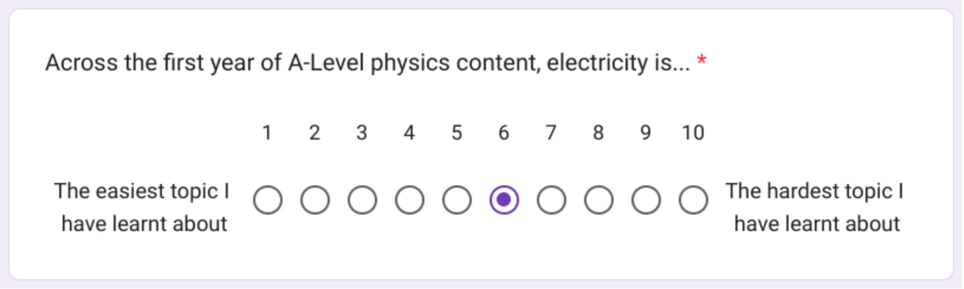
1.5.1 - COLLECTING DATA FROM USERS

To find out what students would want from an electrical circuit simulator, I will distribute a google form to physics students. I will also interview my physics teacher to understand what components we will cover over our course and any considerations involving the theory, such as diodes or capacitors. Collecting data from both sources will prove beneficial as I can tailor the software to a variety of user demographics as well as collect data on how my solution can fix the current problem.

The questionnaire will provide generalised, mass data which can be used to plot graphs and pie charts to visualise how students feel about electrical theory and the solution I am proposing. On the other hand, the interview with my teacher will provide some suggestions and thoughts from a reputable, highly educated source which I will use to model my design choices, as well as comments on the practicality of the program.

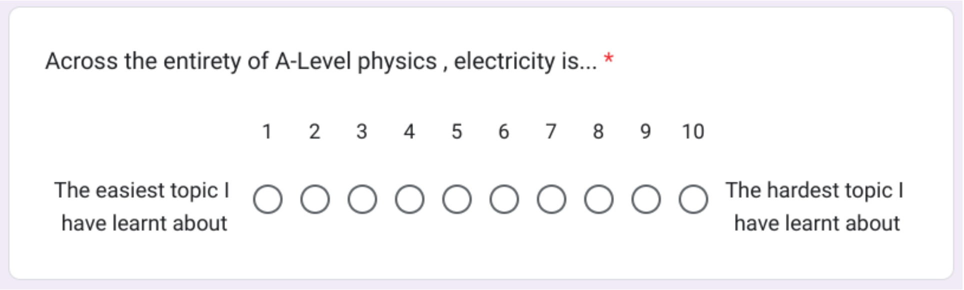
1.5.2 - QUESTIONNAIRE INTRODUCTION

To better understand the needs of the user, I have made a google form that can collect data from A-level students in my school and across the country; it consists of five questions as shown below:



Across the first year of A-Level physics content, electricity is... *

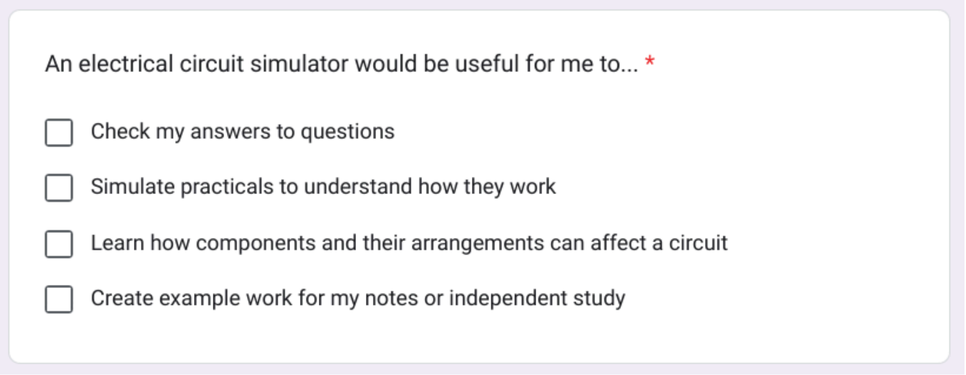
	1	2	3	4	5	6	7	8	9	10	
The easiest topic I have learnt about	○	○	○	○	○	●	○	○	○	○	The hardest topic I have learnt about



Across the entirety of A-Level physics , electricity is... *

	1	2	3	4	5	6	7	8	9	10	
The easiest topic I have learnt about	○	○	○	○	○	○	○	○	○	○	The hardest topic I have learnt about

The first and second questions can determine whether electricity is one of the more complex topics amongst students, to show that the simulator is solving a present, real world problem. The separate questions for AS and A-level content can prove if electricity is one of the harder topics in the 1st or 2nd year.



An electrical circuit simulator would be useful for me to... *

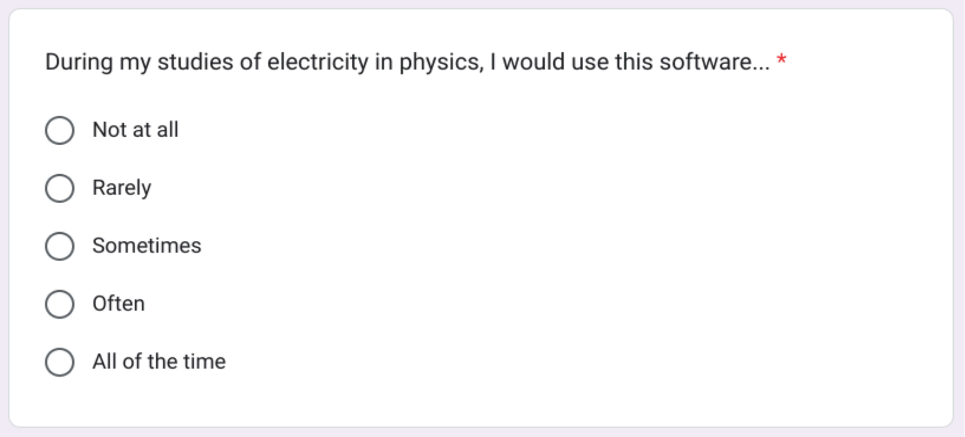
□ Check my answers to questions

□ Simulate practicals to understand how they work

□ Learn how components and their arrangements can affect a circuit

□ Create example work for my notes or independent study

The third question allows me to gain a better understanding of what the software would be primarily used for. Information from this question will reveal what the typical student would use the program for, allowing me to better cater the application to their needs.



During my studies of electricity in physics, I would use this software... *

○ Not at all

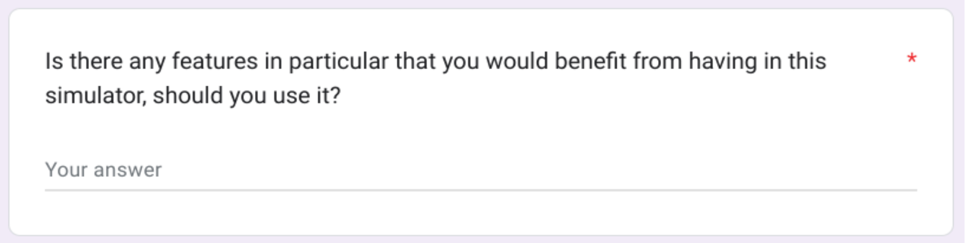
○ Rarely

○ Sometimes

○ Often

○ All of the time

Similar to the first questions, the fourth question will show how much of an impact the simulator would have on a user’s studies.



Is there any features in particular that you would benefit from having in this simulator, should you use it? *

Your answer

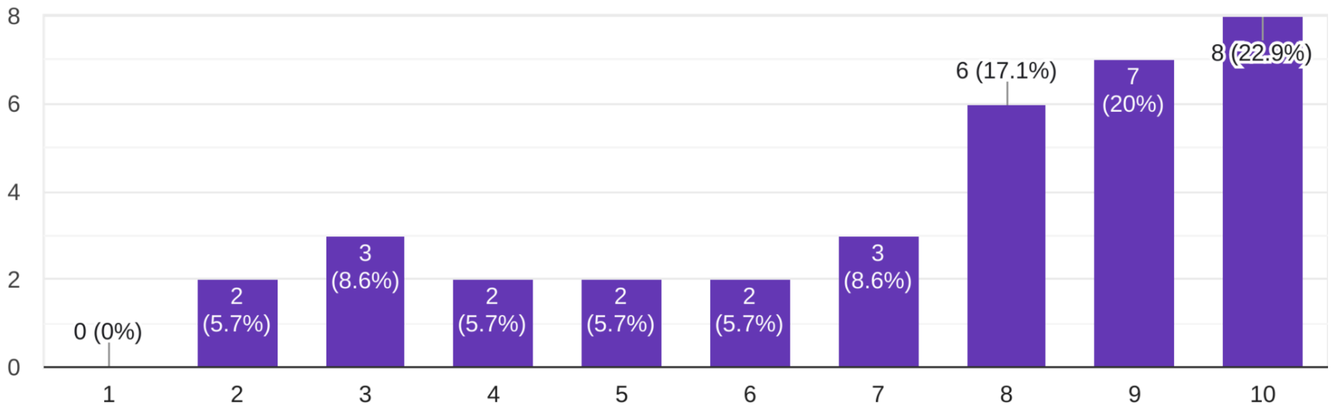
The final question will allow responders to provide any specific requests for the software, which will help in the case that any users have ideas on how the design can be improved.

1.5.3 - EXAMINATION OF RESPONSE DATA

After collecting the responses, I have converted the data into graphs to gain a visual representation of the results. These graphs are shown and explored below:

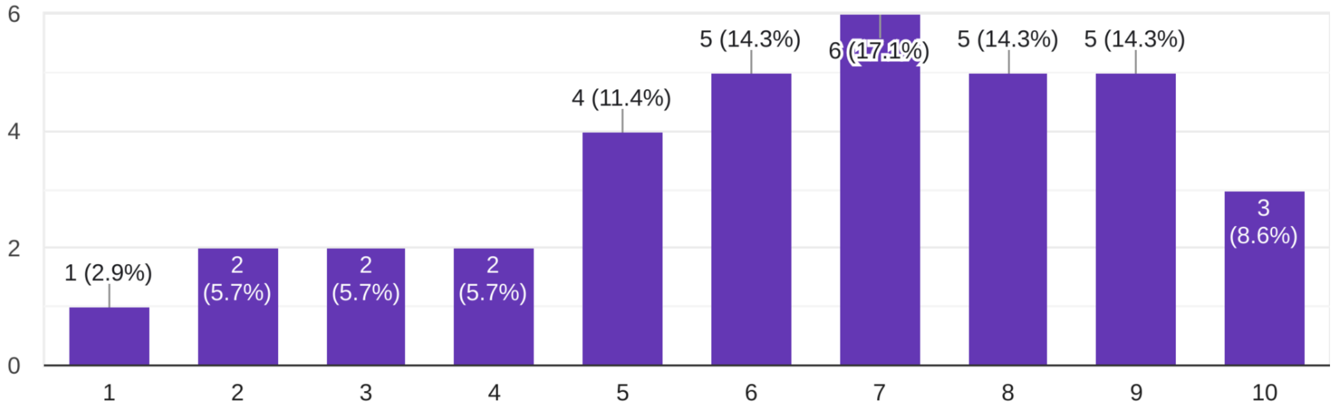
Across the first year of A-Level physics content, electricity is...

35 responses



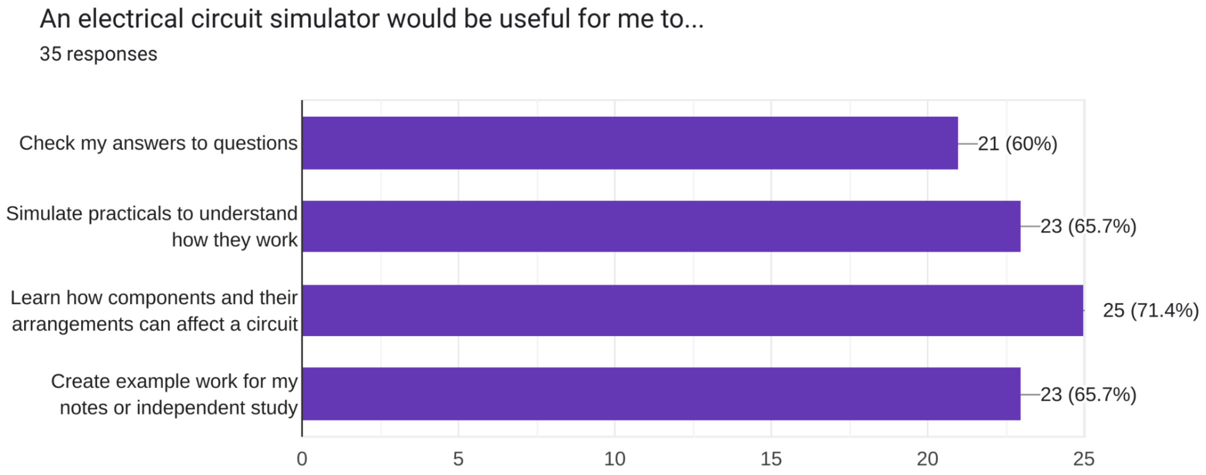
Across the entirety of A-Level physics , electricity is...

35 responses

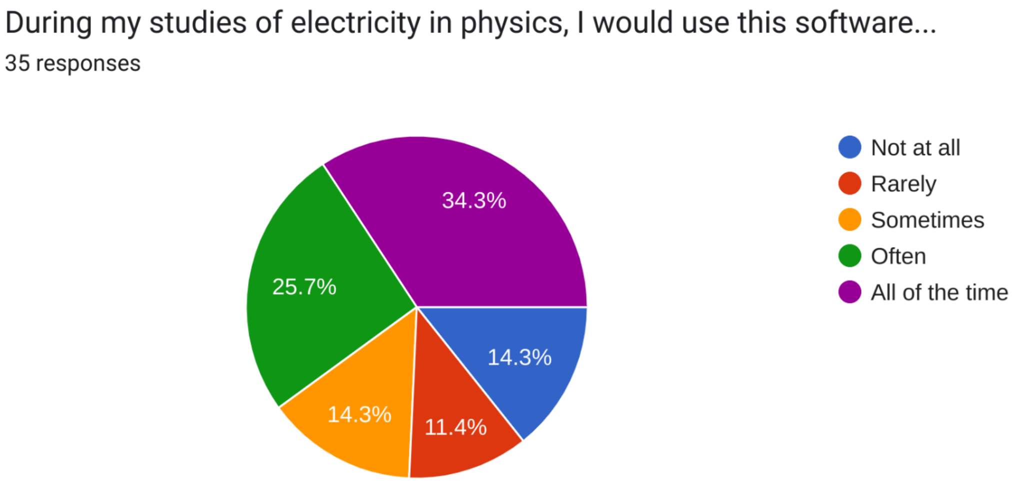


The data from these graphs implies that electricity has an average difficulty of 7.3/10 in the first year of A-level physics, however electricity has an average difficulty of 6.5/10 across the entirety of the course. This shows us that the new content covered in the second year is significantly harder than the content in the first year, bringing the relative difficulty of electricity down. From that, it would be reasonable to assume that my application would see far more usage in the first year of A-Level physics.

The lower difficulty rating for the second year could also be due to students gaining a better understanding of the topic over time. From this I can gather that it is important to provide resources to students in the first year to gain a solid grasp of the content and application. The simulator I develop will be designed to contribute towards this.



The third question was multiple choice and provides insight on what the simulator will be used for. All four options have a high percentage, suggesting that the software will be used for a variety of purposes. Since the most selected response was “Learn how components and their arrangements can affect a circuit”, I plan to implement convenient means of updating a components properties and connections.



During my studies of electricity in physics, I would use this software...

35 responses

The data for the fourth question shows me that the software would be used regularly by 60.0% of students and occasionally by 25.7% of students. This further demonstrates the importance of the software and the impact it could have on a students education.

The final question was an optional open-response which allowed respondents to provide any suggestions or improvements that could be made to the application. Insightful suggestions included:

- Naming components.
- Standard form inputs for variables.
- Being able to view the IV graph for a component.

I will attempt to incorporate these requests into the final design to ensure that the program is catered towards the target audience. The standard form feature seems especially beneficial, as it will prevent errors when the user enters large values.

1.5.4 - INTERVIEW RESULTS

I have planned a series of questions to ask my physics teacher, Mr Allan, about the project to gain any advice on how it can be best suited for A-level students. He was my teacher for the first year of my A-level and often found creative and interactive ways to teach, something which I hope my software can further. Below is a transcript of the interview where said questions were asked:

1) **Do you think electricity is one of the harder topics to teach?**

   In the first year it can take a lot of time for students to grasp the theory, more so than other topics such as mechanics. The main difficulty comes from the way that variables can affect an entire circuit. Solving a circuit question is similar to solving a puzzle, requiring creativity alongside a deep understanding of the theory.

2) **Do you feel like visual demonstrations can help students understand a topic?**

   Absolutely. My lessons contain a lot of diagrams and occasionally I would use a website to simulate the photoelectric effect, and I can say it really clears things up for the students.

3) **How do you currently present questions on circuits with the class?**

   More often than not I'll hand sheets out to each student with a past paper question or one of my own. Once the class is done I'll explain what's on the mark scheme and answer any questions.

4) **Do you think a digital circuit builder and simulator would have much purpose in your classroom?**

   I could see myself using one quite often when going through questions or explaining some theory. If I could alter the values of components and see how that affects the circuit, it would really help with getting across some difficult concepts.

5) **What issues do you think I could encounter whilst developing the software?**

Since voltmeters have infinite resistance and ammeters have zero resistance, that could lead to complications when coding your program. You could instead just make the resistance relatively high or significantly small, that way it could represent a realistic circuit without making the program too complex. If you wanted to include photocells the theory would be unnecessarily difficult , so focusing on it actually working seems more important.

6) **Are there any features for the software that you think a student could benefit from during their own studies?**

I think just being able to experiment freely with different components and layouts will really allow a student to interact with the theory they're learning. I often encourage my students to make examples for their notes and being able to check over their answers with your software would be helpful to make sure they haven't made any mistakes.

7) **Are there any features that you think a student could benefit from during your lessons?**

Perhaps the currents, voltages and resistances could be displayed in standard form. That would be a great aid when we are dealing with millions of volts, amps or ohms within a circuit.

8) **What components would need to be included in the program?**

Of course you would need to add the basic cells, resistors, voltmeters and ammeters. Anything extra would only improve it, such as switches. I think photocells would be far too complex and I already have the means to teach that.

9) **Do you feel like any security measures should be in place for the software?**

I can’t think of a reason to have a password for anything since there isn’t any sensitive information being stored.

10) **Do you have any suggestions for the user interface?**

It would be great if it could be as uncluttered as possible so students aren’t distracted from the circuit we’re going through. A nice, non-distracting colour scheme would be beneficial. That’s all I can think of.

11) **Do you have any final comments or suggestions?**

All I can say is that a circuit simulator sounds really interesting and I can see myself using it, as long as it’s convenient and allows me to do the things I need to do with it.

From the interview I have learnt that the circuit simulator would still be useful with only cells and resistors. However adding, ammeters and voltmeters would be beneficial to help students accurately recreate A-level scenarios. Furthermore, the addition of standard form has been suggested by both a teacher and students; therefore I will make sure to implement the feature in my final product.

1.6 - OBJECTIVES FOR PROPOSED SYSTEM

1. **User Interface** - The visible menu that the user will have access to. This will be used to view and modify circuits. The user interface must:
   1.1. Be well designed to provide a lot of room for the user to place components on their circuit as well as room for a toolbar to add components and such.
   1.2. Have a user-friendly colour scheme that does not prevent visually impaired users from accessing the software. To do so the colour scheme must adhere to the Web Content Accessibility Guidelines (WCAG) by having a contrast ratio of 4:5:1.
   1.3. Contain a list of components that can be added to the current circuit. This list must only contain what is necessary for A-level physics for the purpose of simplicity. Adding components will utilise dynamic generation of objects from a user-defined use of an OOP model, part of group A for technical skills.
   1.4. Be visually similar to standard A-level exam papers by using the same symbols for components and units to provide familiarity with the software and prevent confusion.
   1.5. Clearly present the results of the simulation for each component. This must include the current and voltage (as well as the resistance for resistors).

2. **Algorithm** - The algorithm used to simulate the circuit. It operates mainly via Modified Nodal Analysis. The algorithm should:
    2.1. Be designed to handle erroneous data such as components with zero or infinite resistance and nodes that only have one connection forming a loose circuit.
    2.2. Utilise a depth-first-search for the purpose of grounding nodes in each circuit on the board. This will include Graph-traversal and recursive algorithms, both of which are part of group A for technical skills.
    2.3. Be optimised for performance to ensure that results can be calculated quickly and accurately. However since operations are not repeatedly carried out in real-time, a computer with relatively low processing power could run the program.
    2.4. Use advanced matrix operations in order to solve the system of equations, falling under group A for technical skills. This should use Gauss-Jordan elimination, graph theory and laws of electricity such as Kirchoff’s laws and Ohm’s law. This will allow the creation of a system of equations representing any circuit and the solving of said system.
3. **Object Oriented Programming Model** - The model used to provide data structures and methods to carry out the modified nodal analysis algorithm as well as provide classes relating to the user interface. The OOP Model should:
    3.1. Represent the circuit as a graph with edges being components and vertices being nodes, using Java and its object oriented programming paradigm. Inheritance and polymorphism will be used to create an efficient model of the problem. Graphs and complex OOP models fall under group A for technical skills.
    3.2. Provide classes and methods allowing creation of a user-interface through the Java Swing and AWT packages.

4. **User Interactivity** - The features that are available to the user to customise their circuit and more. These features will include:

    4.1. The ability to modify the properties of any component to copy questions a user has seen or to see how changing properties will affect the circuit. This should be convenient and easily accessible for the user, since most components will have to be modified to suit their needs.

    4.2. Having a convenient way to create circuits by selecting components from the list and dragging them onto the circuit, then being able to connect components with wires. The process should be simple and easy.

1.7 - PROPOSED SOLUTION

To summarise, my proposed solution for the current struggle of learning the fundamentals of electrical theory is an electrical circuit simulator catered towards A-level students. It should contain only the components needed for their A-level and contain features that will enhance their learning through diagrams and interactive learning.

The simulation would use modified nodal analysis to calculate the current passing through each cell and the voltage at each node to further calculate the current passing through each component. The algorithm would use matrices and solve them using gaussian elimination. The use of this algorithm will ensure that the program can handle complex circuits with voltage sources in separate branches.

Java will be an ideal programming language for this task due to its object oriented programming paradigm, allowing for effective and efficient modelling of the circuit as described below. Furthermore, its performance is superior compared to many alternatives, making it suitable for the calculations required. In addition, I am most proficient in Java which will decrease development time and ensure that the code is efficient and well written. To code the GUI I have decided to use Swing and AWT as opposed to JavaFX due to a larger library of components, which could help creating an interactive GUI through toolboxes, sliders and more. Furthermore, I have used Swing for some personal projects and I know how to layout and create panels, labels and buttons, all of which will be needed for the project. The program itself will be downloaded onto a computer so it can be accessed without internet connection.

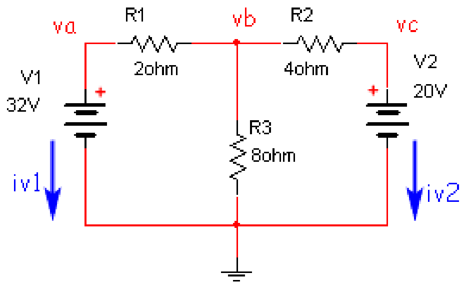
Another benefit of using Java is that I gain access to the EJML (Efficient Java Matrix Library) which can carry out basic and advanced operations on matrices such as addition, multiplication and inversion. However if the library doesn't suit my needs I can code functions to run alongside the library or write my own functions for advanced matrix operations.

1.8 - MODELLING

1.8.1 - REPRESENTING CIRCUIT

The circuit itself will be modelled as a graph of ‘Node’ objects as the vertices connected by ‘Component’ objects representing the edges. Each component can have a resistance value or an emf (voltage across the terminals of a power source). Voltmeters will have to be taken as a resistor of zero conductance and an ammeter would have infinite conductance. The program can iterate through each node, using the connected components to populate a matrix. Another matrix will contain the emf’s of each power supply. These matrices can be combined and solved through gaussian elimination.

1.8.2 - DEMONSTRATION OF COMPUTATIONS



$$\begin{bmatrix} \frac{1}{R_1} & -\frac{1}{R_1} & 0 & 1 & 0 \\ -\frac{1}{R_1} & \frac{1}{R_1}+\frac{1}{R_2}+\frac{1}{R_3} & -\frac{1}{R_2} & 0 & 0 \\ 0 & -\frac{1}{R_2} & \frac{1}{R_2} & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} v_a \\ v_b \\ v_c \\ i_{v1} \\ i_{v2} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ V1 \\ V2 \end{bmatrix}$$

The equation above shows an equation of Ax=z formed from the circuit diagram where A and z have known values and x is unknown (representing the currents and voltages to be calculated. The z matrix can be simply populated by iterating through each voltage source; however the A matrix can be more complex. This matrix can be divided into 4 sub-matrices labelled G,B,C and D. The G matrix is comprised of the conductance of components between each node and the conductance of all components connected to a node. The B matrix is populated with only 0, 1 and -1 values for each node and voltage source pair. A zero is placed when the node and cell are not connected, a 1 is placed if the node is connected to the positive terminal and a -1 is used for the negative terminal. The C matrix is a transpose of the B matrix (identical values but rotated 90 degrees). The D matrix is filled entirely with zeros. The generation of the matrices can be more detailed if complex components such as dependent voltage sources are included, however the reduced complexity of my program makes such features redundant, improving performance.

The x and z matrices are far less complex than the A matrix, since both are composed of two submatrices that are easily generated by entering the voltage of each source.

Once the matrices are populated, A system of linear equations is represented and able to be solved, the algorithm for which will be explained in detail in the design chapter.

2 - Design

2.1 - OVERALL SYSTEM SUMMARY

As stated in the analysis section, the aim of my project is to develop a program which allows a user to create a circuit and update the attributes of components. The program must then calculate the current and voltage across all components and output this to the interface. The purpose of this is to aid in teaching A-level physics students about the fundamentals of electrical circuits by allowing them to check their answers to a question, simulate practicals and more. To develop the software, I will be using the Eclipse IDE for java on account of its user-friendly interface and wide array of libraries should they be needed.

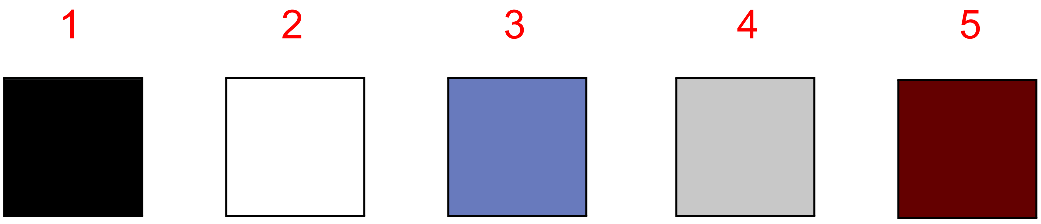
The user-interface will be made using both the java swing and AWT libraries. AWT provides the Graphics2D class which allows painting of complex shapes. This is beneficial as I plan to create the main user-interface using drawn shapes as opposed to text boxes and images. Doing so will save memory as no images have to be stored alongside the program and the images are simple enough that they can be represented as a vector graphic quickly and efficiently. Furthermore, Using painted shapes will allow me to implement dynamic features that are not included in the packages such as dragging and dropping components whilst having them snap to a grid. The main window must have a grid where circuits can be represented, a list of components which can be dragged onto this grid and a section where the user can view and alter the properties of any component.

In addition, AWT provides means of recognising mouse inputs, from which I can write functions for when the mouse is pressed, clicked, released and moved at specific coordinates. With this and the Graphics2D class I can create a dynamic interface with features tailored to the project as well as means for the user to interact with it.

However, a series of text boxes will be implemented for the separate window used to edit components. This window must have boxes for the name, resistance or voltage of a component that can be typed into by the user, in order to change its properties.

2.2 - USER INTERFACE DESIGN

The user interface is composed of two separate windows, the main window and the editor window. The main will be open constantly whilst the editor is a pop-up, opened to edit the properties of a component. The colour scheme across the program will consist of the following colours:



1) With an RGB value of 0,0,0, black will be used for fonts, borders and components. Since A-level exam questions use black for circuits, this will provide a sense of familiarity to a student using this software.

2) With an RGB value of 255,255,255, white will be used for the background of text boxes and buttons as well as the background of the circuit grid. This will contrast the black foreground making text easy to read whilst ensuring the software follows WCAG colour contrast guidelines.

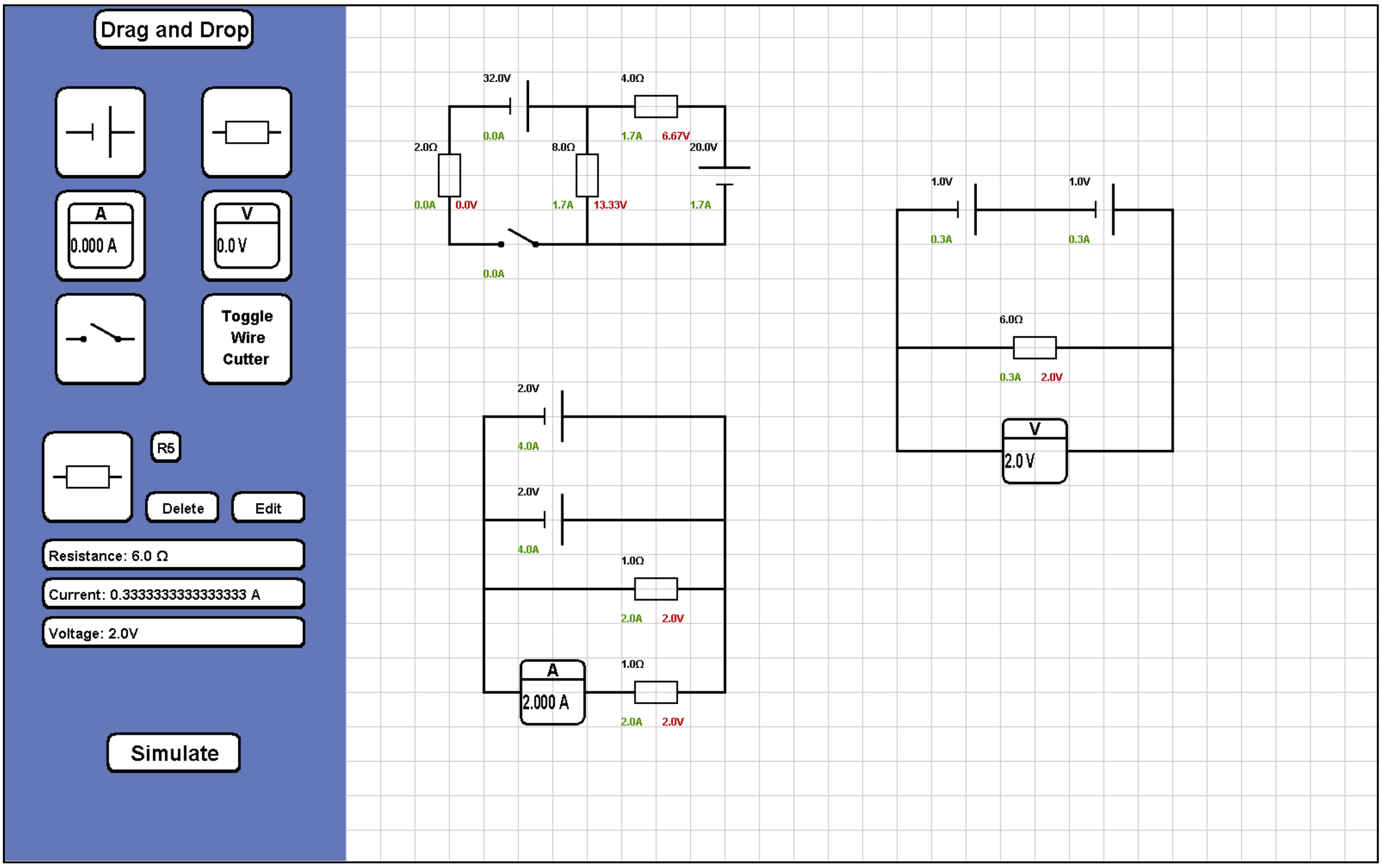
3) With an RGB value of 104, 122, 189, this faint sky blue will be used for the background of the program, as it is easy on the eyes yet provides a much needed splash of colour. In addition to this, a study from the University of Columbia has shown that blue can induce creativity and focus, both of which are essential for an A-level physics student.

4) With an RGB value of 200,200,200, light grey will be used for the grid on the circuit board. This particular shade has been chosen since it is not so dark as to clutter the main window and remove focus from the circuits, yet dark enough that the grid can be easily seen by the user.

5) With an RGB value of 100,0,0, this dark red will be used as a background with white text for the error messages within the “Editor” window shown in 2.2.2. The red and white combination have a contrast ratio of 13.61:1, exceeding the WCAG guidelines.

2.2.1 - MAIN WINDOW DESIGN

Below is my design for the main user interface. The white, gridded, right-hand portion of the screen is the board where circuits can be constructed and visualised. The smaller, blue left-hand portion is the menu where the user can drag new components from the tools section (or select the wire cutter), as well as view and edit the properties of selected components from the board. The “Simulate” button executes the MNA algorithm and updates the values of all components on the board, whilst the “Edit” button will open the editor window for the selected component. When dragging a component, it will appear red when it cannot be placed due to not being on the board or overlapping with other components; whereas a dragged component in an acceptable location will be green. The component must also snap to the grid when it has reached the board.



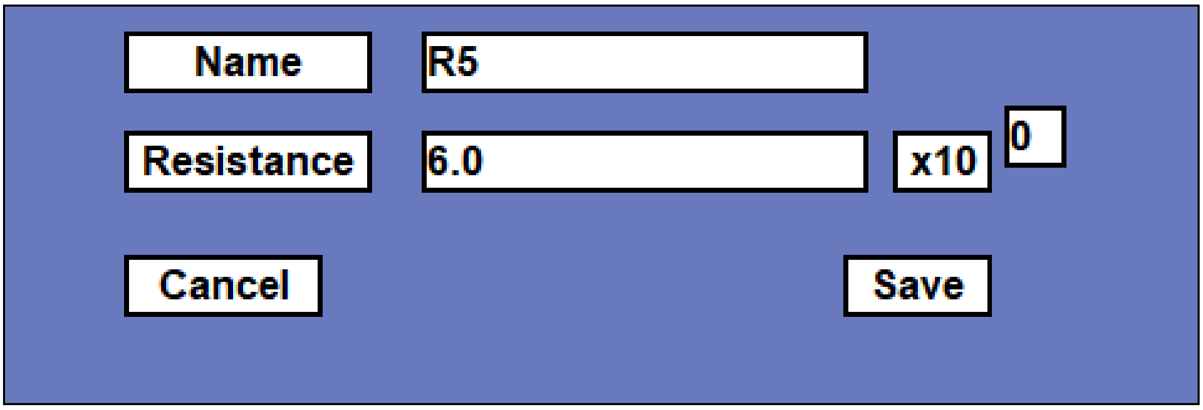
Resistance: 6.0 Ω

Current: 0.3333333333333333 A

Voltage: 2.0V

2.2.2 - COMPONENT EDITOR WINDOW DESIGN

Briefly mentioned in 2.2.1, the “Edit” button opens a new window with data linked to the component that was displayed in the main window, the design for this is shown below:



Name R5

Resistance 6.0 x10 ^{0}

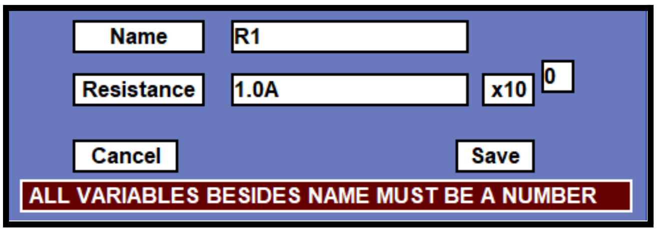
In this window, fields can be directly typed in to change the variables of a component. The second row with resistance will only appear for a cell or resistor; however, a cell will replace the “Resistance” label with “Voltage” as that is the changeable property of that component. For the user’s convenience, the property of the component is converted to standard form, allowing the user to modify the mantissa and exponent of the value. This will prove useful when a user is simulating circuits on large or small magnitudes, which is a common scenario in A-level physics.

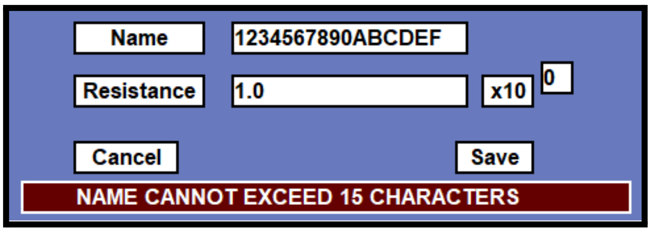
The cancel button will close the editor window, disregarding any changes made by the user, whereas the save button will update the component’s variables to the values entered and close the editor window.

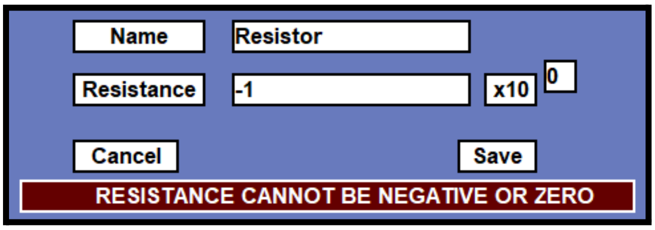
When the save button is pressed, the program must ensure the data entered is valid. The rules are as follows:

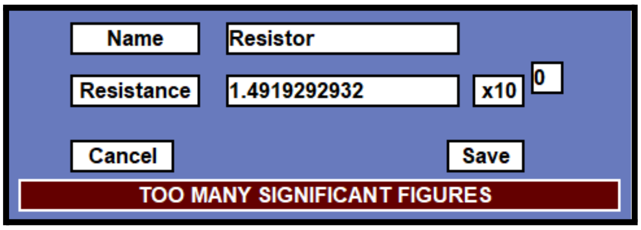
- The name may not exceed 15 characters, nor be less than 1 character.
- The resistance or voltage cannot be negative or zero.
- The exponent must be an integer between -15 and 15
- The resistance / voltage and exponent must be numbers
- The resistance / voltage must be no longer than 7 characters

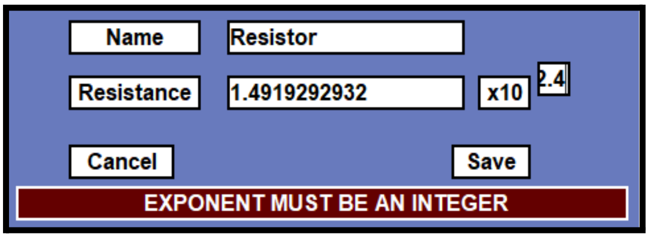
If the save button is pressed whilst any of these conditions are not met, an error box will appear and will remain until the save or cancel button is pressed again. Some examples of these error boxes are shown below:

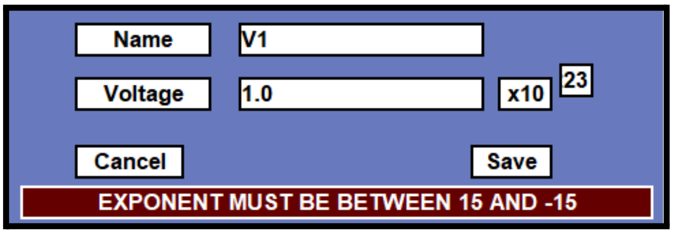

Name: R1
Resistance: 1.0A x10 0
ALL VARIABLES BESIDES NAME MUST BE A NUMBER


Name: 1234567890ABCDEF
Resistance: 1.0 x10 0
NAME CANNOT EXCEED 15 CHARACTERS


Name: Resistor
Resistance: -1 x10 0
RESISTANCE CANNOT BE NEGATIVE OR ZERO


Name: Resistor
Resistance: 1.4919292932 x10 0
TOO MANY SIGNIFICANT FIGURES


Name: Resistor
Resistance: 1.4919292932 x10 2.4
EXPONENT MUST BE AN INTEGER


Name: V1
Voltage: 1.0 x10 23
EXPONENT MUST BE BETWEEN 15 AND -15

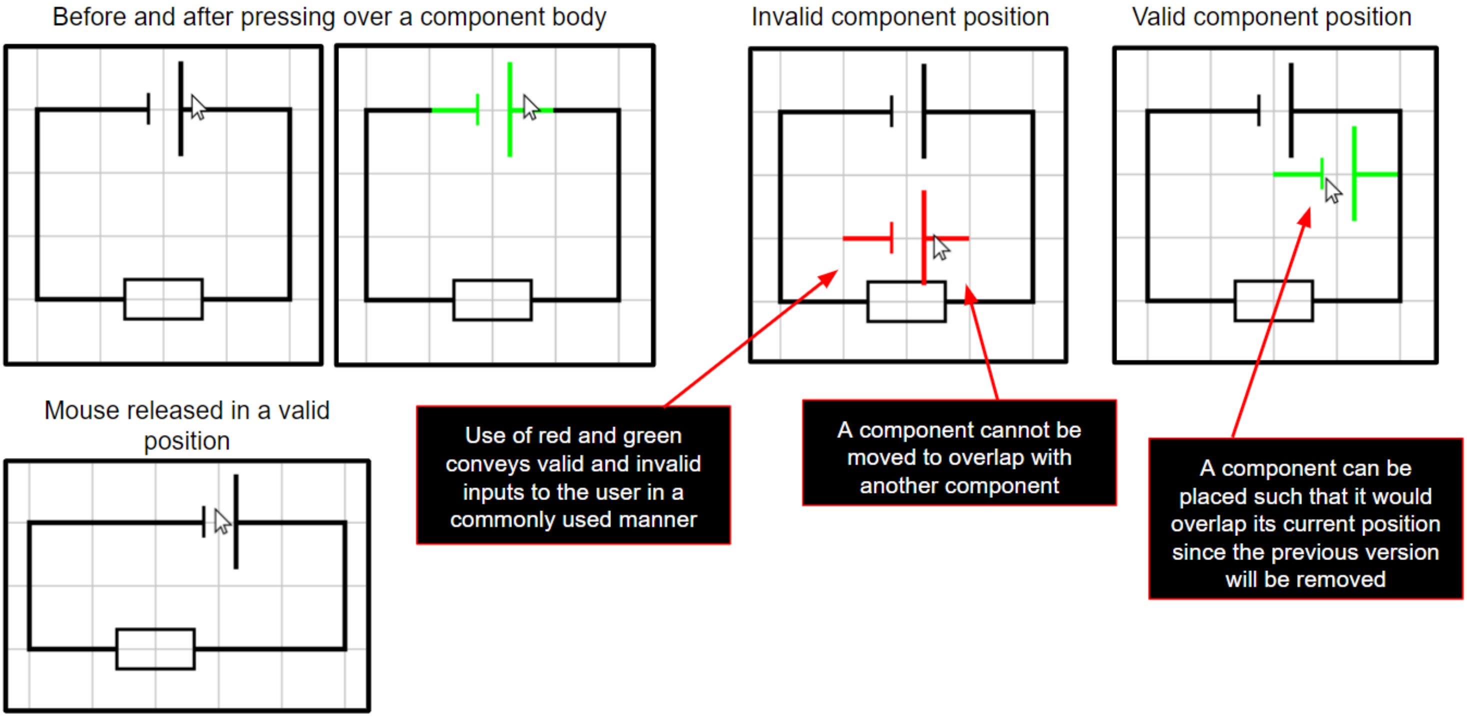
2.2.3 - USER INPUT DESIGN

To allow the user to build and alter a circuit I plan to implement recognition of mouse inputs, which will track the location of the mouse and when it is pressed and unpressed. In order to create a dynamic and interactive user-interface, the user must be able to:

- Drag and drop components that are currently on the board to change their position.
- Add new components to the circuit by dragging and dropping from the menu.
- Rotate components to be facing up, down, left or right.
- Draw wires across any two ports of two separate components.
- Remove connections using the wire cutter feature.
- Click any button on the menu.
- Update component properties using the editor window.

2.2.3.1 - DRAG AND DROP

To move components that are currently on the board as well as add new components from the menu, the user must be able to click and drag on components. The designs for moving a component and adding a new component are both shown below:



Before and after pressing over a component body

Invalid component position

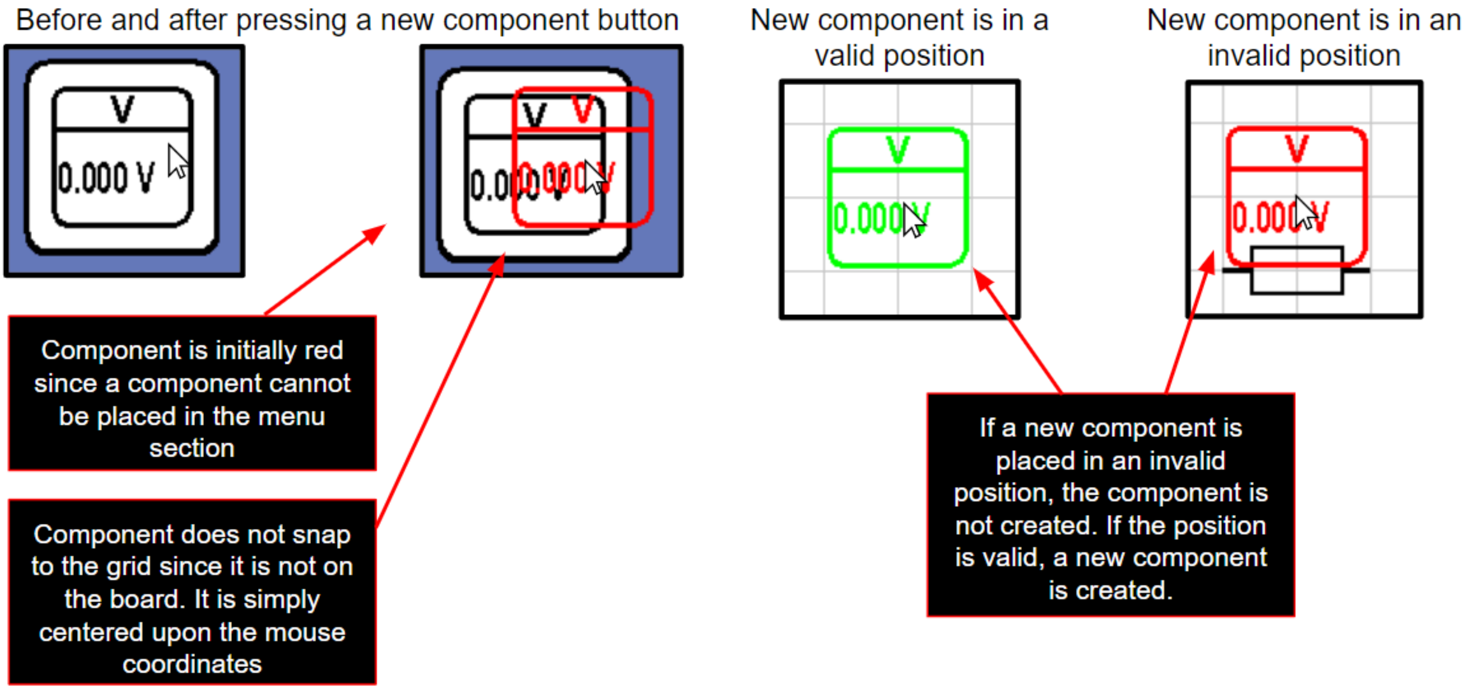
Valid component position

Mouse released in a valid position

Use of red and green conveys valid and invalid inputs to the user in a commonly used manner

A component cannot be moved to overlap with another component

A component can be placed such that it would overlap its current position since the previous version will be removed



Before and after pressing a new component button

New component is in a valid position

New component is in an invalid position

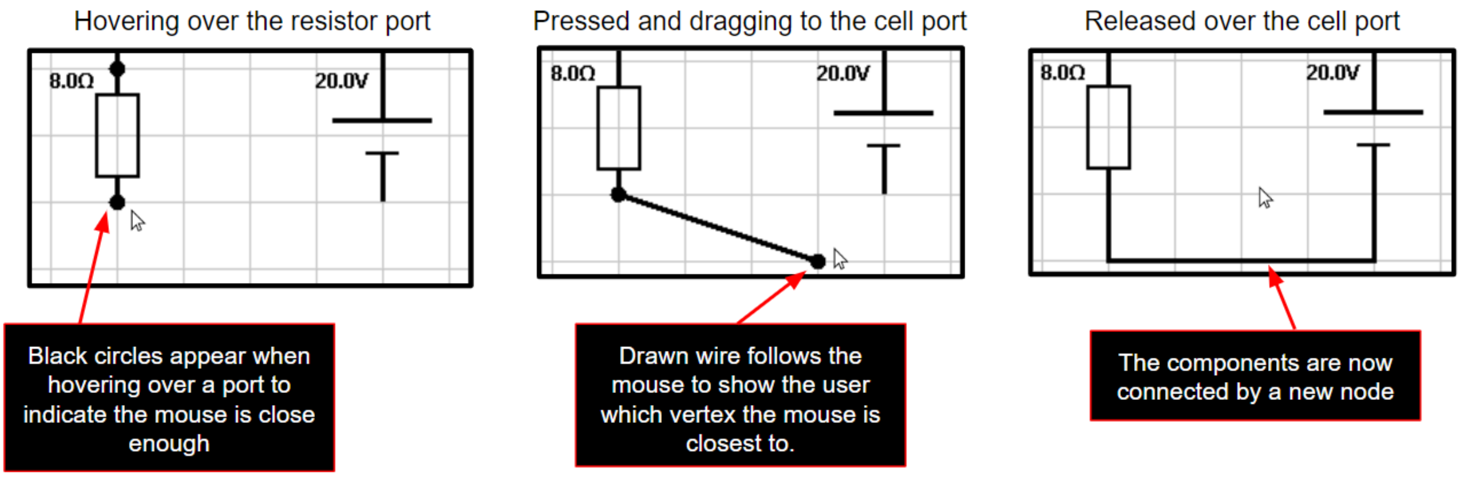
Component is initially red since a component cannot be placed in the menu section

Component does not snap to the grid since it is not on the board. It is simply centered upon the mouse coordinates

If a new component is placed in an invalid position, the component is not created. If the position is valid, a new component is created.

2.2.3.2 - CREATING AND CUTTING WIRES

In order to connect the components on the board to form a closed circuit, the user must be able to allocate connections themselves. The general idea for this feature is shown below:



Hovering over the resistor port

Pressed and dragging to the cell port

Released over the cell port

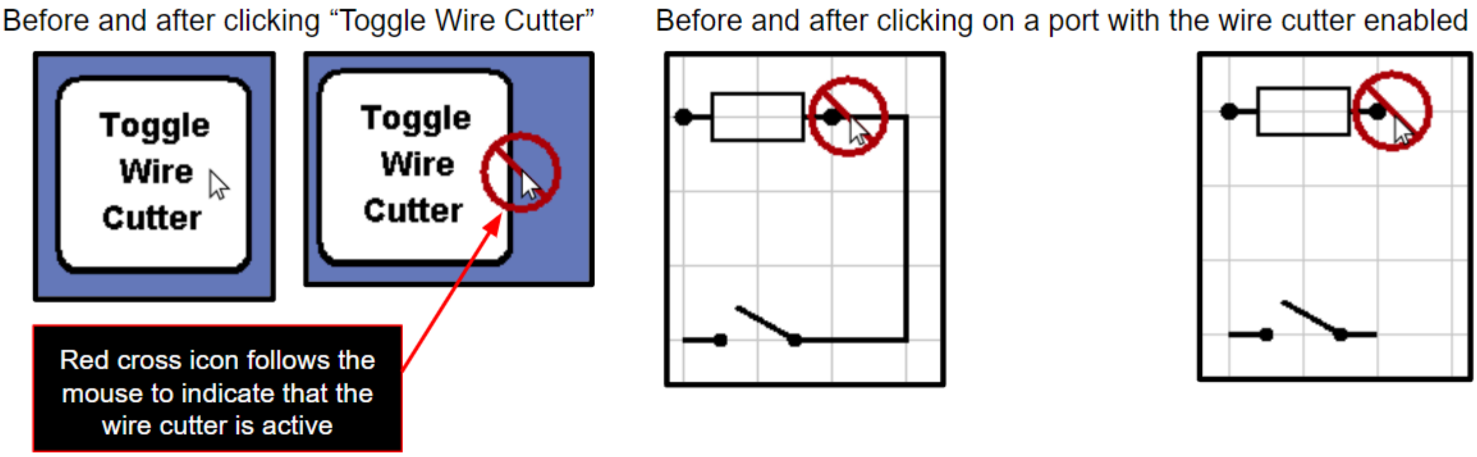
Black circles appear when hovering over a port to indicate the mouse is close enough

Drawn wire follows the mouse to show the user which vertex the mouse is closest to.

The components are now connected by a new node

The further design and implementation of this feature, with respect to the nodes created between connected components, is detailed in section 2.3.4.

As for cutting wires, when the wire cutter tool is selected, clicking on a port will remove the connection of that port. An example of this process is shown below:



Before and after clicking “Toggle Wire Cutter”

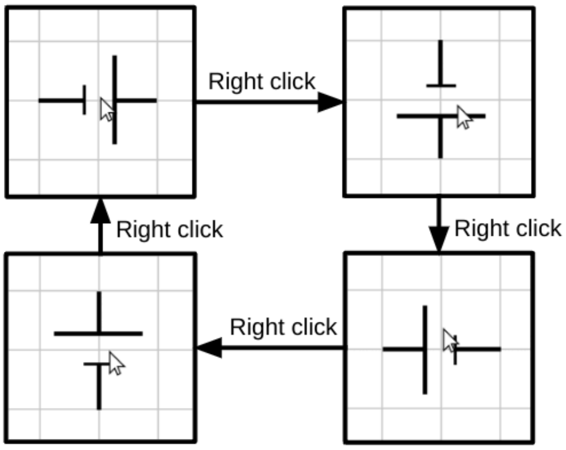
Before and after clicking on a port with the wire cutter enabled

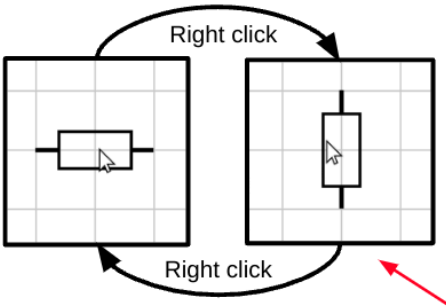
Red cross icon follows the mouse to indicate that the wire cutter is active

Clicking on the “Toggle Wire Cutter” button again will disable the wire cutter and remove the mouse-following icon.

2.2.3.3 - ROTATING COMPONENTS

In order to create a circuit with an understandable layout, the user must be able to rotate a component through increments of 90°. This will also allow a relatively accurate representation of any A-level physics circuit. This feature would work as shown below:



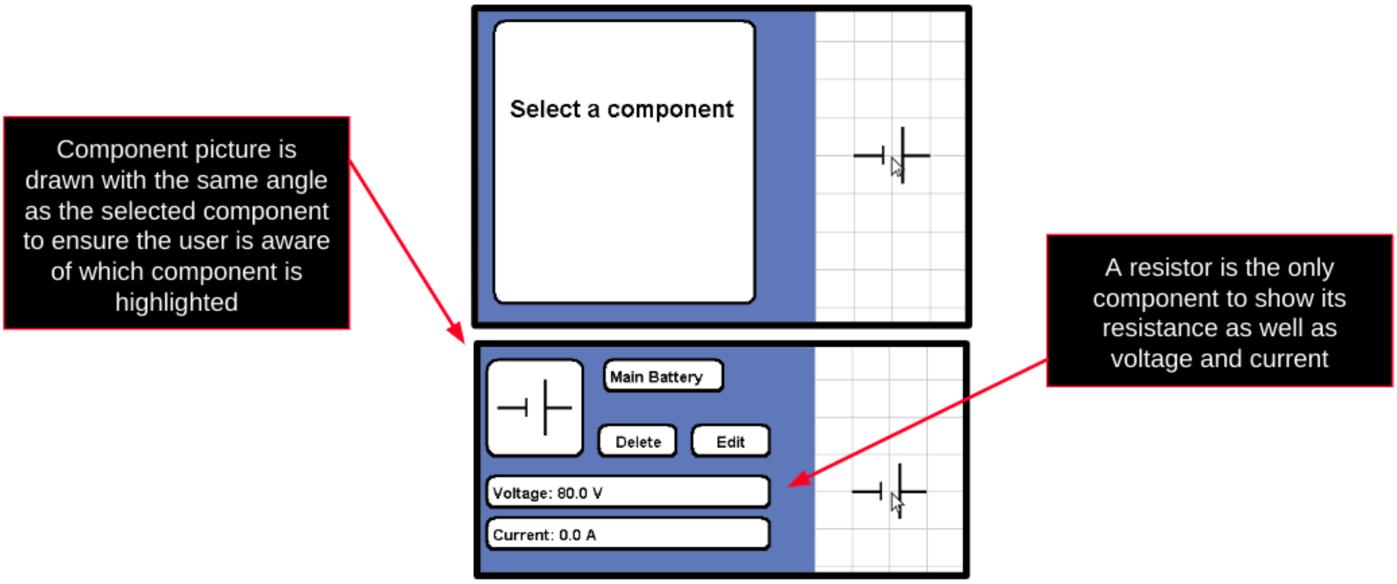


A symmetrical component such as a resistor or switch will only need two appearances for vertical and horizontal. However they must still have four potential angles so wires connecting to the ports are drawn the right way around

2.2.3.4 - MODIFYING THE HIGHLIGHTED COMPONENT

In order to delete or edit a component, the user must assign the component object to the "highlighted" variable by left clicking a component on the board. This will work like so:

Before and after left clicking on a component



Voltage: 80.0 V

Current: 0.0 A

Component picture is drawn with the same angle as the selected component to ensure the user is aware of which component is highlighted

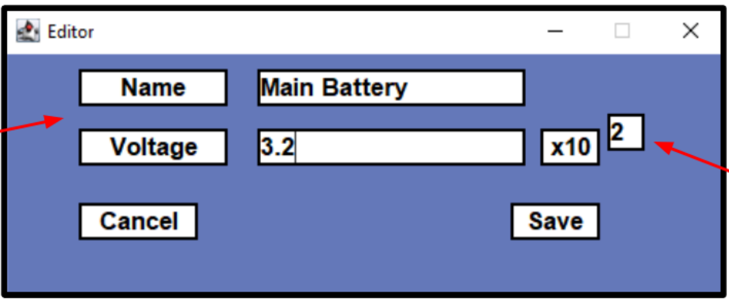
A resistor is the only component to show its resistance as well as voltage and current

For a switch, clicking the "Toggle open/closed" button that appears will call the "toggleSwitch" method on the highlighted component, setting the "highlighted" variable to the component the method returns. This will effectively replace the closed/open switch with an open/closed switch with the same position, connections, angle and name.

Upon pressing the "Delete" button, the highlighted menu reverts to its previous "Select a component" state and the component is removed from the circuit by calling the "delete" method from the "Component" class.

When the "Edit" button is pressed, the editor window is opened by calling the "setComponent" method upon the "Editor" object. The design for this window is shown on the subsequent page:

Resistors will have a “Resistance” box as opposed to “Voltage”. A switch or meter will only have a “Name” box since its intrinsic properties cannot be altered.



Name: Main Battery

Voltage: 3.2 x10 2

The user-interface uses standard for notation A x 10 ^{B} as opposed to the scientific notation of A E B. This is because the first notation is most commonly used in A-level specifications.

Unlike the main window, the editor window utilises the swing “JTextField” class to create text boxes that can be typed into by the user, as well as the “JButton” class to create the “Cancel” and “Save” buttons. When the editor window is opened, the values within the boxes are already filled to match the current values of the selected component. For the example shown above, the voltage of the cell was 320 volts, however this has been converted to standard form to make the value more readable and easier to modify. In addition, the exponent applied to the voltage can be changed by the user to avoid errors when entering large values (such as adding additional zeroes).

2.2.4 - USER INPUT IMPLEMENTATION

To allow the user’s mouse inputs to be recognised, I have imported the “mouseListener” interface from the AWT package. This provides the methods:

- mousePressed
- mouseReleased
- mouseClicked
- mouseDragged
- mouseMoved

At the end of any of these methods, the “repaint” method is called to show the user any changes made.

2.2.4.1 - MOUSE PRESSED

The “mousePressed” method is called when the mouse transitions from unpressed to pressed. This, alongside the coordinates where the press occurs, can be used to determine if a component is being dragged or a wire is being drawn from it. The method consists of if and else-if-statements with the mouse X and Y coordinates as the conditions. Firstly, there is an if-statement splitting the board from the menu, as well as ending the method if the “cutting” variable is set to true.

If a point on the menu was pressed, a series of if-statements determine if any of the new component buttons were pressed. In any of these cases, the Component variable “Selected” is set to a new “Component” with an Y coordinate of -2 to indicate it has not yet been placed.

If the mouse has been pressed within the board, a for-loop iterates through each component to determine if the press was on top of one. If so, the “Selected” variable is set to the pressed component. The integer “drag” is set to 3 if either of the component’s ports are pressed. The boolean “port1” is set to true if the port selected is the anode, and set to false for the cathode. This is used to draw a wire from that port to the port of another component. If the main body of the component is pressed, “drag” is set to 1 to show that the component itself is being dragged.

2.2.4.2 - MOUSE RELEASED

The “mouseReleased” method is called when the mouse transitions from pressed to unpressed. This method is called after “mousePressed” and uses the “drag” value to determine what happens when the mouse is released. If “drag” is 3 (indicating a wire being drawn from the selected component) then a for-loop iterates through each component to check if the mouse has been released upon any of the ports. If this is the case, the “Connect” method from the “Component” class is called on the “Selected” component, with the parameters being the component the mouse was released on, the previously assigned “port1” variable and a boolean stating which other port the wire is drawn between. As further explained in 2.3.4, this will connect the two components through a “Node” object.

If “drag” is set to 1, then the “Selected” component is released from being dragged. If the boolean “placeable” is true, the component’s X and Y variables are updated to match the grid coordinates closest to the mouse, whereas if “placeable” is false, nothing happens. However, if “Selected” is a new component from the menu (indicated by its unique Y coordinate of -2) the component is deleted as it was never created in the first place. The “placeable” variable is set during the paint method, being true if the dragged component is on the board and does not collide with another component and false otherwise.

2.2.4.3 - MOUSE CLICKED

The “mouseClicked” method is called when “mousePressed” and “mouseReleased” are called sequentially at the same point on the screen. This is used for the five buttons in the main window as well as the wire cutter tool, rotating feature and highlighting a component. Within this method, an if-statement first splits clicks on the board with clicks on the menu.

If the click is on the menu, further if statements determine which button was clicked. When the “Edit” button is clicked, the “setComponent” method is called on the editor object with the highlighted component being the parameter. This makes the editor window visible allowing the user to change a component’s properties.

If the “Delete” button is clicked, the highlighted component is removed from the circuit using the “delete” method within the component class (explained in more detail in 2.3.2).

When the “Simulate” button is clicked, the MNA method from the main class is called again, updating the voltage and current of every component present. This is surrounded by a try-catch in case the circuit simulated is invalid, due to short circuits or incompletion. If an error is returned within the MNA method, an error window opens informing the user that the circuit is invalid.

If the “Toggle open/closed” button is clicked and the highlighted component is a switch, the “toggleSwitch” method is called on the highlighted component. To prevent confusion, this button is only visible if the highlighted component is a switch.

If the “Toggle Wire Cutter” button is clicked, the “cutting” boolean variable is reversed, becoming true if false and false if true.

Finally, if the user clicks on the board, a for-loop iterates through each component, to determine if any components were clicked. If a component was left clicked, the highlighted component is set to the clicked component. However, right clicking on a component will rotate it by 90° via incrementing the “angle” variable and using a modulo operator to only allow angle values of 1,2,3 and 4. If a port is clicked whilst the “cutting” variable is true, then the “sever” method is called on the node at the port.

2.2.4.4 - MOUSE MOVED AND MOUSE DRAGGED

The “mouseMoved” method is called when the mouse coordinates are changed due to the user moving their mouse. This method simply changes the “mouseX” and “mouseY” variables to the current position of the mouse and calls the “paint” function.

The “mouseDragged” method is called instead of “mouseMoved” if the mouse is pressed whilst it is moving. Despite this, “mouseDragged” carries out the same, previously mentioned operations as “mouseMoved”.

2.3 - JAVA CLASS SYSTEM

The class system for the program must include the following classes:

- A “Main” class where the main window object is initialised. This class will contain the “nodeList”, “cellList” and “resList” variables which are ArrayList objects holding each node, cell or resistor on the board. The Main class must also contain the methods relating to the modified nodal analysis algorithm.

- A “Builder” class which extends the “JFrame” class provided by the Swing package with no methods besides a constructor. The constructor must update the Builder object to have the desired title, properties and dimensions upon instantiation. This object will act as the window that is visible on the user’s device.

- A “Canvas” class which extends the “JPanel” class provided by the Swing package. This will be instantiated once and added to the instance of the “Builder”. The “Canvas” object can be drawn on using the inherited “paint” method to provide a user-interface within the window.

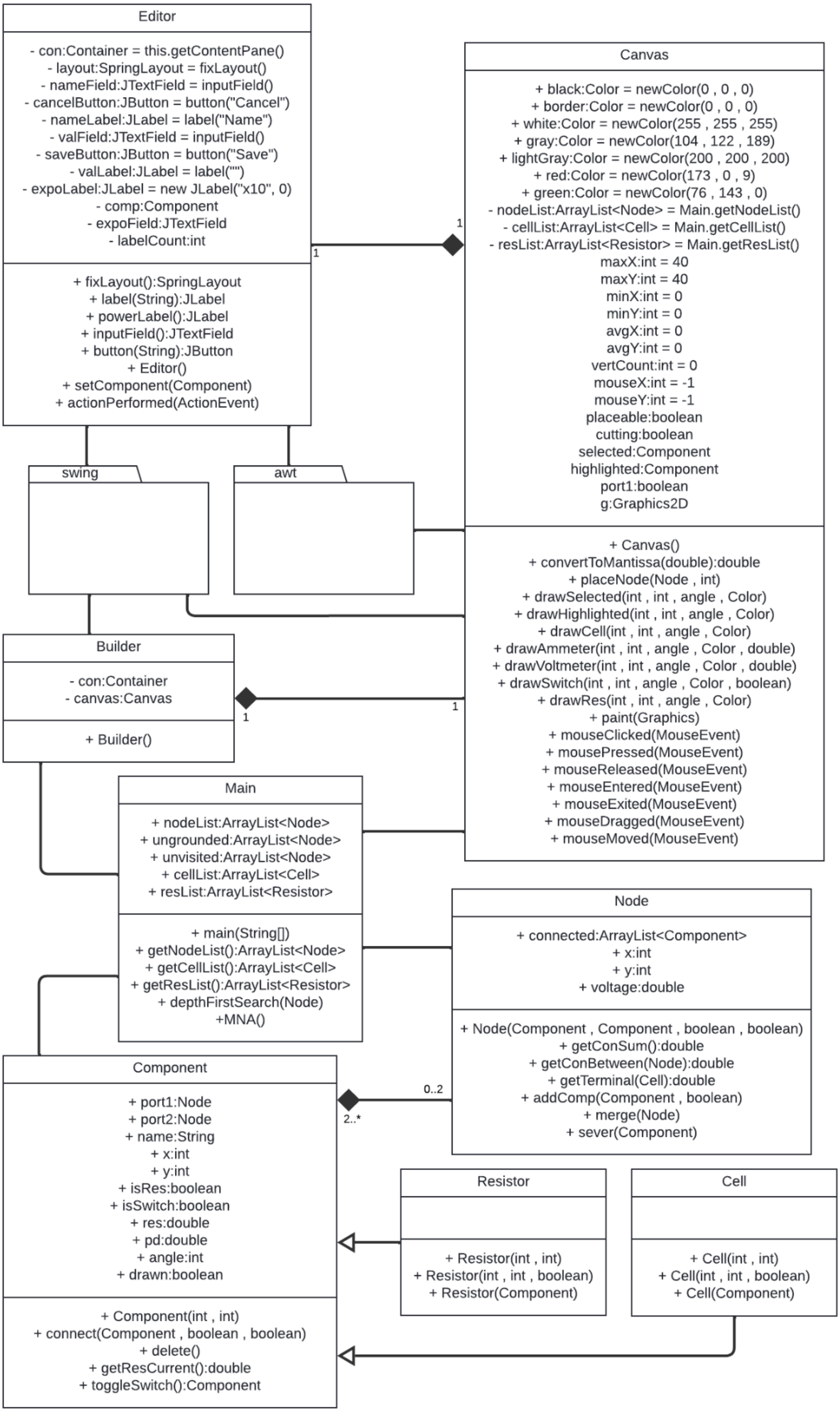
- An “Editor” class, extending the “JFrame” class just as the “Builder” does. The instance of this class will provide the editing window described in 2.2.2.

- A “Component” superclass with “Resistor” and “Cell” subclasses to represent the various components that can be created by the user. These can be used to encapsulate the data relating to each component to aid with the modified nodal analysis algorithm and output the results to the user-interface.

- A “Node” class which will represent the connections between the components on the board. This class will be used to simulate the circuit as well as show connections of components on the user-interface.

2.3.1 - UML CLASS DIAGRAM

I have constructed a UML class diagram to represent the program in terms of classes, packages and their relationships. This will aid development of the program as making use of Java's object oriented programming style will lead to effective troubleshooting and reusable functions on account of its modularity and encapsulation, since any issues are isolated within classes. This diagram is shown on the next page and will be explained throughout this section:


Editor
- con:Container = this.getContentPane()
- layout:SpringLayout = fixLayout()
- nameField:JTextField = inputField()
- cancelButton:JButton = button("Cancel")
- nameLabel:JLabel = label("Name")
- valField:JTextField = inputField()
- saveButton:JButton = button("Save")
- valLabel:JLabel = label("")
- expoLabel:JLabel = new JLabel("x10", 0)
- comp:Component
- expoField:JTextField
- labelCount:int
+ fixLayout():SpringLayout
+ label(String):JLabel
+ powerLabel():JLabel
+ inputField():JTextField
+ button(String):JButton
+ Editor()
+ setComponent(Component)
+ actionPerformed(ActionEvent)
1
1
Canvas
+ black:Color = newColor(0 , 0 , 0)
+ border:Color = newColor(0 , 0 , 0)
+ white:Color = newColor(255 , 255 , 255)
+ gray:Color = newColor(104 , 122 , 189)
+ lightGray:Color = newColor(200 , 200 , 200)
+ red:Color = newColor(173 , 0 , 9)
+ green:Color = newColor(76 , 143 , 0)
- nodeList:ArrayList<Node> = Main.getNodeList()
- cellList:ArrayList<Cell> = Main.getCellList()
- resList:ArrayList<Resistor> = Main.getResList()
maxX:int = 40
maxY:int = 40
minX:int = 0
minY:int = 0
avgX:int = 0
avgY:int = 0
vertCount:int = 0
mouseX:int = -1
mouseY:int = -1
placeable:boolean
cutting:boolean
selected:Component
highlighted:Component
port1:boolean
g:Graphics2D
+ Canvas()
+ convertToMantissa(double):double
+ placeNode(Node , int)
+ drawSelected(int , int , angle , Color)
+ drawHighlighted(int , int , angle , Color)
+ drawCell(int , int , angle , Color)
+ drawAmmeter(int , int , angle , Color , double)
+ drawVoltmeter(int , int , angle , Color , double)
+ drawSwitch(int , int , angle , Color , boolean)
+ drawRes(int , int , angle , Color)
+ paint(Graphics)
+ mouseClicked(MouseEvent)
+ mousePressed(MouseEvent)
+ mouseReleased(MouseEvent)
+ mouseEntered(MouseEvent)
+ mouseExited(MouseEvent)
+ mouseDragged(MouseEvent)
+ mouseMoved(MouseEvent)
swing
awt
Builder
- con:Container
- canvas:Canvas
+ Builder()
1
1
Main
+ nodeList:ArrayList<Node>
+ ungrounded:ArrayList<Node>
+ unvisited:ArrayList<Node>
+ cellList:ArrayList<Cell>
+ resList:ArrayList<Resistor>
+ main(String[])
+ getNodeList():ArrayList<Node>
+ getCellList():ArrayList<Cell>
+ getResList():ArrayList<Resistor>
+ depthFirstSearch(Node)
+MNA()
Node
+ connected:ArrayList<Component>
+ x:int
+ y:int
+ voltage:double
+ Node(Component , Component , boolean , boolean)
+ getConSum():double
+ getConBetween(Node):double
+ getTerminal(Cell):double
+ addComp(Component , boolean)
+ merge(Node)
+ sever(Component)
0..2
2..*
Component
+ port1:Node
+ port2:Node
+ name:String
+ x:int
+ y:int
+ isRes:boolean
+ isSwitch:boolean
+ res:double
+ pd:double
+ angle:int
+ drawn:boolean
+ Component(int , int)
+ connect(Component , boolean , boolean)
+ delete()
+ getResCurrent():double
+ toggleSwitch():Component
Resistor
+ Resistor(int , int)
+ Resistor(int , int , boolean)
+ Resistor(Component)
Cell
+ Cell(int , int)
+ Cell(int , int , boolean)
+ Cell(Component)

2.3.2 - COMPONENT CLASS

A “Component” object will be used to represent a resistor, cell, switch, ammeter or voltmeter created by the user. Using Java’s inheritance I have designed a “Component” class with the variables and methods shown in the UML diagram.

To further separate cells and resistors, two subclasses named “Cell” and “Resistor” will be implemented, both of which extend the “Component” class. Since the modified nodal analysis algorithm uses the conductance (the reciprocal of resistance) of each resistor, a resistor with no resistance will be unnecessarily complex to represent (as the program would be dividing by zero). Therefore components with no resistance such as cells, ammeters and closed switches will be implemented as a “cell” object, despite ammeters and closed switches having a supply voltage of zero volts.

On the other hand, any component with resistance such as resistors, voltmeters and open switches can be represented as a “resistor” object since they have no supply voltage yet a considerable resistance. Ideally, voltmeters and open switches will have an infinite resistance, not allowing any current to flow through them. To implement this, I will utilise an otherwise invalid value of -1 for its resistance to indicate that, when retrieving the component’s conductance, a zero should be returned.

To link components to the nodes and therefore other components, each component will have “Node” variables “port1” and “port2” which store the node connected to the anode and cathode respectively (positive and negative terminals). The component will also need a “current” variable to be updated at the end of the modified nodal analysis routine.

For the purposes of a user interface, each component must have a String “name” variable and an “angle” being 1,2,3, or 4 to show its rotation. The “Component” class will contain the following methods:

Constructor - This method is never used to create a “Component” object, as all components are part of the subclasses “Cell” or “Resistor”. However, this method is used within the constructors for these subclasses, assigning an x and y coordinate as well as an angle to the component.

connect - This is used to connect a specific port of the component the method is called upon to the port of another component specified in the parameters. The pseudocode for this method is shown below, note that the method is called upon a component and that component uses the identifier “this”, thisPort1 refers to the port of “this” being connected and compPort2 refers to the port of “comp” being connected:

```
Function connect (Component comp , boolean thisPort1 , boolean compPort1)
    Node thisNode
    Node compNode
    End the function if comp and this are the same component
    If thisPort1 = true
        thisNode = this.port1
    Else
        thisNode = this.port2
    End else
    If compPort1 = true
        compNode = comp.port
    Else
        compNode = comp.port2
    End else
    If thisNode and compNode are both null
        Main.getNodeList.add( new Node(this , comp , thisPort1 , compPort1))
    End if
    If thisNode is null and compNode is not null
        compNode.add(this , thisPort1)
    End if
    If compNode is null and thisNode is not null
        compNode.add(comp , compPort1 )
    End if
    If thisNode and compNode are both not null
        thisNode.merge(compNode)
    End if
End function
```

By following different procedures for the different cases shown in this table, the program avoids directly connecting nodes (the MNA process cannot allow that). The table below explains these cases in terms of the nodes at the ports being connected:

Scenario	Component A already has a node at the port (node A)	Component A does not have a node at the port
Component B already has a node at the port (node B)	Merge node A with node B, adding all components from node B's "connected" list to node A's list then removing node B from the program. Set the port of component B to node A.	Add component A to node B's "connected" list and set the port variable of A to node B.
Component B does not have a node at the port	Add component B to node A's "connected" list and set the port variable of B to node A.	Create a new node and set the port of both components to that node. Then add components A and B to the "connected" list for the node.

delete - This method is used when the “Delete” button is pressed by the user as shown in 2.2.3.4. This removes the component from any nodes it is connected to by using the “sever” method of the node class. Then the component is removed from the “cellList” or “resList” depending on if it is a “Cell” or “Resistor” respectively.

getResCurrent - This method can only be called upon a resistor, as is checked by examining the “isRes" boolean of the component. When called, the current across a resistor is calculated using the formula current = potential difference / resistance. The potential difference is found by finding the difference in voltage between both ports of the resistor.

toggleSwitch - The “toggleSwitch” method calls the Resistor or Cell constructor with the parameter being the component the method was called upon. This replaces a closed switch with an open switch and vice versa. If the component is a switch and a cell, the resistor constructor is called. If the component is a switch and a resistor, the cell constructor is called.

2.3.3 - CELL AND RESISTOR CLASS

The “Cell” and “Resistor” classes are subclasses to “Component”, the use of inheritance allows the program to reuse methods from the component class, whilst separating it into two distinct groups. The cell and resistor are the fundamental categories that all components within the program fall under. If a component has any form of resistance, it is a resistor. If a component provides any power to the circuit or has no resistance, it is a cell. Since the “Component” class provides most of the methods these components need to interact with the circuit, these subclasses will contain multiple unique constructors in order to initialise variables to acceptable values and add the component to the correct ArrayList (resList or cellList).

Constructor 1 - The first constructor is used for a standard cell or resistor and has parameters “X” and “Y” which are used again as parameters for the “Component” constructor mentioned in 2.3.2. For both classes, the booleans “isSwitch” and “isMeter” are set to false to indicate that the component is neither a switch nor a meter. Additionally, the boolean “isRes” is set to false in the “Cell” class and true in the “Resistor” class. In the “Cell” class. This constructor sets the resistance and potential difference to zero and the name to “Vn” where n is the current number of cells on the board (for example V1 and V2) as this is common A-level notation. As for the “Resistor” class, the resistance is set to one, as a resistor cannot have zero resistance and the potential difference is set to zero as it does not provide any power to the circuit. The name of the resistor is set to “Rn” where n is the number of resistors on the board. Finally, the cell is added to the “cellList” or the resistor is added to the “resList”.

Constructor 2 - The second constructor is used to create an ammeter or a closed switch within the “Cell” class and a voltmeter or an open switch within the “Resistor” class. The parameters are once again “X” and “Y” and so the method uses the constructor from the “Component” class to set the coordinates of the component using these parameters. However, this constructor has a third parameter, a boolean called “isSwitch”. The “isSwitch” variable of the component is set to the value of the parameter “isSwitch”, the “isMeter” variable is set to the opposite of “isSwitch” (since the component will either be one or the other). The “isRes” boolean is set to true within the “Resistor” class and false within the “Cell” class.

Within the “Cell” class, this method sets the potential difference and resistance to zero as both an ammeter and closed switch have these properties. The name is set to “Ammeter” if “isMeter” is true, otherwise the name is set to “Switch”. Finally, the component is added to the “cellList”

On the other hand, the “Resistor” class constructor sets resistance to -1 (to indicate an infinite resistance) and the potential difference to zero, as both a voltmeter and an open switch share these values. The name is set to “Voltmeter” or “Switch” depending on the parameter “isSwitch” and the component is added to the “resList”.

Constructor 3 - The final constructor is called within the “toggleSwitch” method, being used to copy the variables of a closed or open switch to a new open or closed switch. The only parameter for this constructor is the component “comp” that is being copied. The “Component” constructor is once again used with the parameters being the x and y coordinates of “comp”. The “angle” variable of the new component is set to the value of the “angle” variable held by “comp”. If “comp” was connected to nodes on either of its ports, as indicated by variables “port1” and “port2”, the new component is connected to the same nodes at the same ports. This is done by using the “addComp” method of the “Node” class. The name of the new component is copied from the name of “comp” and the “delete” method from the “Component” class is called upon “comp” since it is no longer needed.

For a resistor, potential difference is set to zero and resistance is set to -1, whereas for a cell, potential difference and resistance are set to zero (the reasons for both are explained within the “Constructor 2” description). The boolean “isRes” is set to the opposite of the value of “isRes” stored by “comp”. By copying the variables from “comp” , this method will provide the visual effect of a switch being opened and closed without being replaced by a new component.

2.3.4 - NODE CLASS

An object of the “Node” class represents a junction where wires connected to components meet. This means that a node will have no current when connected to only one component, hence a node must require two connected components to be relevant to the flow of charge throughout the circuit.

Nodes will not be directly accessible or visible to the user as they are only needed for the modified nodal analysis algorithm and drawing of wires for the user interface. This level of abstraction through information hiding will simplify the program, as the user can focus on the connections of components as opposed to the added consideration of nodes.

Each node will have a “connected” Component list storing each component it is connected to. A collection of nodes can be treated as a graph where the “connected” list represents an adjacency list. Whenever a component's port nodes are updated, the “connected” list of that node is updated to ensure consistency across the program. In addition to this, each node will need a voltage variable that can be set during the MNA algorithm and x and y values to represent its location on the board. The “Node” class will contain the following methods:

Constructor - The constructor for the node class has four parameters: the two “Component” instances connected by the node and two booleans specifying which port of either component is being connected. The method “addComp” is called upon the node once for each component at the ports specified by the parameters.

getConSum - This method returns the total conductance of resistors surrounding the node, as is needed during the MNA process. This method will iterate through the "connected" ArrayList and sum up the conductances of each resistor connected. The pseudocode for this process is shown below:

```
Function getConSum ()
    Double conSum = 0
    For each component in connected where isRes and not isSwitch or isMeter
        conSum += 1.0 / component.resistance
    End for
    Return conSum
End function
```

getConBetween - Similar to "getConSum", this method returns the negative sum of the conductances directly between two nodes, a value of which the MNA method requires. "getConBetween" is called on a node with the parameter being the other node that the conductance between is being calculated for. The method iterates through each component in "connected" and checks if the component is a resistor and if it is also connected to the other node. If this is the case, the negative conductance is added to the value which is to be returned. The pseudocode for this algorithm is detailed below:

```
Function getConBetween (Node n)
    Double conSum = 0
    For each component in connected where isRes and not isSwitch or isMeter
        If component.port1 = n OR component.port2 = n
            conSum -= 1.0 / component.resistance
        End if
    End for
    Return conSum
End function
```

getTerminal - Called within the MNA method, “getTerminal” is called upon a node with the parameter being a “Cell” object. If the positive terminal of the cell is connected to the node, a 1 is returned. If the negative terminal of the cell is connected to the node, a -1 is returned. Finally, if the cell is not connected to the node, a 0 is returned. This value will be needed for each node with each cell. The specific terminal the node is connected to can be determined by checking the “port1” and “port2” variables of the cell, keeping in mind that port1 is the anode and port2 is the cathode.

addComp - Used in many other methods within the “Node” and “Component” classes, “addComp” is always used when adding a component to a node at a specific port. This method sets “port1” or “port2” of the component to the node. The parameters include the component being added and a boolean showing which port the component is being connected by. Lastly, the component is added to the “connected” ArrayList. To maintain consistency across the model, components will only be connected to nodes using this method; this ensures that the port variables of components and the “connected” lists of nodes match.

merge - “merge” is called upon a node (A) with the parameter being the node that is being merged (B). This method will merge together the two nodes by copying all components from the “connected” list from B to A and using “sever” to remove the components from B. This method contains the only instance of “sever” being called with the boolean parameter as true. This is to instruct the “sever” method to avoid deleting the node until all components have been removed, so that the last component can be added to the new node. Once the “connected” list of B is empty, B is removed from the nodeList. Pseudocode further explaining this method is shown below:

```
Function merge (Node n)
    If this is not n
        While n.connected is not empty
            If n.connected.get(0).port1 = n
                n.connected.get(0).port1 = this
            End if
            If n.connected.get(0).port2 = n
                n.connected.get(0).port2 = this
            End if
            n.sever(n.connected.get(0), true)
        End while
    End if
End function
```

sever - The “sever” method is similar to “addComp” except instead of adding a component to a node, the component is being removed. This is done by setting the port variable where the node is to “null” and removing the component from the “connected” ArrayList. The parameters are the component to be removed and a boolean to indicate whether “sever” is being called from the “merge” method or not.

The similarity to “addComp” comes from the fact that “sever” is always used when removing the connection between a component and a node, to ensures that both objects are updated accordingly (once again ensuring consistency and integrity throughout the program).

However, special considerations must be taken when removing a component from a node since, as mentioned previously, a node cannot exist unless it is connecting two or more components. Hence, if removing a component leads to the “connected” list having just one component and the boolean parameter is false, “sever” is called to remove the remaining component and the node is removed from the “nodeList”.

In the case that the boolean parameter is true, the node is only deleted when it has no components connected to it. This ensures that, when merging two nodes, the final component is not deleted before it can be added to the other node.

Shown below is some pseudocode to further explain this process:

```
Function sever (Component comp, boolean merging)
    If comp.port1 = this
        comp.port1 = null
    End if
    If comp.port2 = this
        comp.port2 = null
    End if
    If this.connected.size = 1 AND merging == false
        this.sever(n.connected.get(0)
        nodeList.remove(this)
    Else if this.connected.size = 0 AND merging == true
        nodeList.remove(this)
    End if
End function
```

2.3.5 - BUILDER CLASS

Unlike the “Node” and “Component” classes, the “Builder” class is irrelevant to the modified nodal analysis algorithm. This class simply exists to create a window and provide a user interface for the user. By extending the “JFrame” class from the “Swing” package, I have designed a class which provides a window and modifies it to be more suitable for the program.

The constructor for the “Builder” class is the only method not inherited from “JFrame”, and carries out the following procedures:

- Sets the title of the window to “Circuit Yard”.
- Sets the size of the window to 1616 x 1039 pixels, as this size makes the inner window 1600 x 1000.
- Makes the window non-resizable by using “setResizable” with parameter “False”.
- Sets the program to terminate when the window is closed via the close button in the top right corner.
- Sets the window location to the centre of the screen using the “setLocationRelativeTo” method with parameter “null”.
- Uses “setLayout” with parameter “null” to ensure that the added “Canvas” instance is properly placed within the window.
- Adds the “Canvas” instance by using “add” with parameter “new Canvas()”.
- Make the window visible using “setVisible” with parameter “True”.

2.3.6 - EDITOR CLASS

Similar to the “Builder” class, the “Editor” class is a subclass of the “JFrame”. As opposed to the “Canvas” class, this class uses the Swing components “JLabel”, “JTextField” and “JButton” to create boxes that can be typed into and buttons that can be clicked.

The constructor for “Editor” is similar to that of the “Builder” class. For example, the window is set to non-resizable and the title is set to “Editor”. However, the editor window is not set to visible until the “Edit” button is clicked and the “setComponent" method is called.

setComponent - The “setComponent” method is called within the “mouseClicked” function when the “Edit” button is clicked, with the parameter being the current selected component. Within this method, the “Name” field is set to the name of the component. If the component is a resistor, the “Resistance” label and text field appears; whereas if the component is a cell, the “Voltage” label and text field appears. In both cases, the resistance or voltage is converted into standard form using the following formulas:

```
Mantissa = num / Math.pow(10 , Math.floor(Math.log10( num )))
Exponent = Math.floor(Math.log10(num))
```

For example, this will convert 5230 into 5.23 and 3, allowing the program to output 5.23 x 10^3. This will also work for values less than one as, for example, 0.0124 will become 1.24 x 10^{-2}.

2.3.7 - CANVAS CLASS

The “Canvas” class extends the “JPanel” class provided by Swing, and implements the interfaces “MouseMotionListener” and “MouseInputListener” to provide the pre-built methods listed in 2.2.4. The instance of this class provides a component that can be drawn on by a “Graphics2D” object, allowing for the main user interface shown in 2.2.1. The “Canvas” class has a variety of variables, including four “Color” objects from the AWT package, representing the colours shown in 2.2. This class contains a simple constructor initialising the size and background colour of the component. However, the main method is “paint”, which is called each time the “repaint” method is invoked.

paint - This method carries out all operations relating to drawing the user interface for the main screen. A long list of commands will draw the buttons, grid, circuit and such, often utilising methods to avoid redundant code. This includes “drawCell”, “drawRes” and the other methods relating to drawing components, as it is carried out multiple times throughout the paint method.

Through using the “drag” variable, changed when mouse action is detected, the paint method can draw specific shapes to match the user’s inputs, such as drawing wires between components or dragging a component on the board. When a component is being dragged, the “placeable” boolean referred to in 2.2.4.2 is updated within the “paint” method when determining if the dragged component should be red or green.

placeNode - The “placeNode” method is used once the components have been drawn, called within the “paint” method for each node in the “nodeList”. This method is recursive, with the parameters being the node being placed and the iterative variable to go through each component in the “connected” list. When the “connected” list has not been fully checked (the general case), the placeNode method is called again with the iterative variable incremented by +1 (after identifying the angle of connection between the component and the node). Once the iterative variable has surpassed the size of the “connected” list, the base case is reached. At this point, the angle of connection and position of each component is used to determine the location of the node and the wires are drawn onto the user interface.

The pseudocode for the general case of the “placeNode” method is shown below. Note that “minY”, “minX”, “avgX”, “avgY” and “vertCount” are reset to zero for each node, whilst “maxY” and “maxX” are reset to 40:

```
If current.connected.size > i
    If current is connected to the bottom of current.connected.get(i)
        If current.connected.get(i).y + 2 > minY
            minY = current.connected.get(i).y + 2
        End if
        avgX += current.connected.get(i).x
        vertCount++
        placeNode(current, i + 1)
        Draw a line from the bottom of the component to the node
    Else if current is connected to the top of current.connected.get(i)
        If current.connected.get(i).y - 2 < maxY
            maxY = current.connected.get(i).y - 2
        End if
        avgX += current.connected.get(i).x
        vertCount ++
        placeNode(current, i + 1)
        Draw a line from the top of the component to the node
    Else if current is connected to the right of current.connected.get(i)
        If current.connected.get(i).x + 2 > minX)
            minX = current.connected.get(i).x + 2
        End if
        avgY += current.connected.get(i).y
        placeNode(current, i + 1)
        Draw a line from the right side of the component to the node
    Else
        If current.connected.get(i).x - 2  < maxX
            maxX = current.connected.get(i).x - 2
        End if
        avgY += current.connected.get(i).y
        placeNode(current, i + 1)
        Draw a line from the left side of the component to the node
    End if
End if
```

The “vertCount" variable is used to show the number of vertical components connected, and can be used to find the number of horizontal components as well (total number of components - vertCount). This is used when there are only horizontal or only vertical components and the average Y or X value must be calculated to correctly place the node.

As for the base case of the “placeNode” method, the following pseudocode demonstrates the process of calculating the ideal node location:

```
Else
    If minX = 0 OR maxX = 40
        If maxX is not 40
            current.x = maxX
        Else if minX is not 0
            current.x = minX
        Else
            current.x = avgX / vertCount
        End if
    Else
        current.x = minX + maxX / 2
    End if
    if minY = 0 OR maxY = 40
        If maxY is not 40
            current.y = maxY
        Else if minY is not 0
            current.y = minY
        Else
            current.y = avgY/(current.connected.size() - vertCount);
        End if
    Else
        current.y = (minY + maxY) / 2
    End if
End else
```

After the base case has calculated the ideal node location, using the variables modified during previous “placeNode” calls, the method ends and all previous calls of the method finish by drawing the wire from the node to the component. The use of a recursive algorithm ensures time efficiency as the “connected” list only has to be iterated through once as opposed to iterating once to update the variables and calculate the node location then iterating once more to draw the wires. This recursivity also shortens the amount of code by a great deal, ensuring ease of understanding and reduced overall program size.

2.4 - MODIFIED NODAL ANALYSIS ALGORITHM

The aim of the MNA algorithm is to find the current and voltage through each component. This is done via:

- Grounding nodes in each circuit to provide reference points of zero voltage.
- Populating a matrix to form an abstraction of a system of linear equations.
- Handling cases of identical rows and columns within the matrix.
- Solving the system of linear equations.
- Assigning solutions to nodes and cells.
- Using said solutions to calculate voltages and currents through each resistor.

Each of these processes will be carried out in sequence in a single method called MNA within the main class.

2.4.1 - GROUNDING NODES VIA DEPTH FIRST SEARCH

After developing a prototype MNA algorithm which could simulate a simple circuit, I attempted to simulate two unconnected circuits simultaneously. This led to an error as one circuit was ungrounded since only one node was grounded at the beginning of the algorithm.

To amend this, I developed a recursive depth-first search algorithm (DFS) to be carried out on the entire circuit. The process is to call the DFS on the first node in the list of unvisited nodes and, for every node visited, remove that node from the unvisited list. Once the DFS is finished, an entire disconnected subgraph has been identified and the first node visited is grounded. This process repeats until the visited list is empty, since all subgraphs will have one ground node. Below is the pseudocode for this process:

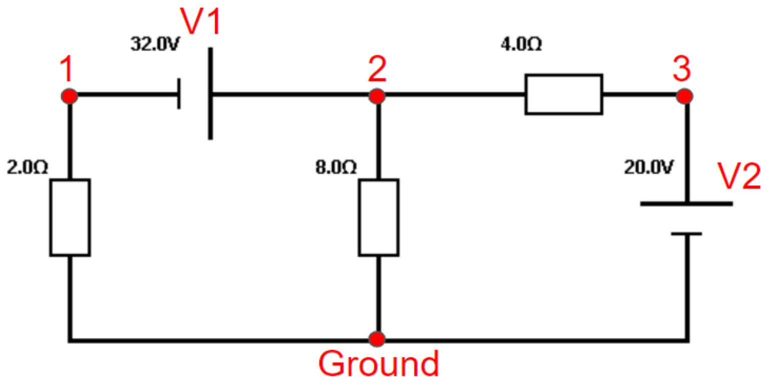
```
Function depthFirstSearch (Node node)
        remove node from unvisited
        For each Component in node.connected
                If unvisited contains Component.port1
                        depthFirstSearch(Component.port1)
                End if
                Else if unvisited contains Component.port2
                        depthFirstSearch(Component.port2)
                End else
        End for
End function
Function MNA
        unvisited = copy of nodeList
        ungrounded = copy of nodeList
        While unvisited is not empty
                remove unvisited[1] from ungrounded
                depthFirstSearch
        End while
        (Remaining MNA function)
End function
```

2.4.2 - POPULATING THE MATRIX

Using the guide made by Swarthmore College (shown in Analysis 1.3.1), I've developed an algorithm to populate a matrix that represents the simultaneous equations within the circuits. This is done in 3 parts since the final matrix (the 'A' matrix) is composed of the G, B, C, D and Z matrices.

Firstly, the G matrix is populated, with dimensions n x n (where n is the number of nodes). The elements across the diagonal contain the sum of the conductances surrounding each node, other elements contain the negative sum of conductance directly across two nodes. Shown below is an example of a circuit, its corresponding G matrix, and pseudocode representing this process:

```
2Darray double A[ungrounded.size+cellList.size][ungrounded.size+ cellList.size + 1]
For i from 0 to ungrounded.size
        Node node= ungrounded[ i ]
        A[ i ][ i ] = getConductanceSumAround(node)
        For j from i + 1 to ungrounded.size
                A[ i ][ j ] and A[ j ][ i ] = getConductaneBetween(node and ungrounded[ j ])
        End for
End for
```



$$\begin{pmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2}+\frac{1}{8} & \frac{-1}{4} \\ 0 & \frac{-1}{4} & \frac{1}{4} \end{pmatrix}$$

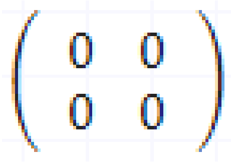
Since the matrix is symmetric diagonally, only n(n+1)/2 unique elements are calculated in my algorithm instead of n^2 to optimise the process.

Secondly, the B and C matrices are populated, the B matrix is an n x c matrix (where c is the number of cells). Each row corresponds to a node and columns refer to cells. A ‘1’ is placed in an element if the node is connected to the positive terminal of the cell, a ‘-1’ if it is connected to the negative terminal of the cell or a ‘0’ if the node and cell are not in contact. The C matrix is and c x n matrix following the same rules. Once again, to optimise the algorithm, only the B matrix is calculated and the C matrix is simply the transpose of the B matrix. The B and C matrices as well as the code is shown below for the same example:

$$\begin{pmatrix} -1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} -1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
For each node in ungrounded
        For j from ungrounded.size to ungrounded.size + cellList.size
                A[ i ][ j ] and A[ j ][ i ] = node.getTerminal(cellList[j - ungrounded.size])
        End for
End for
```

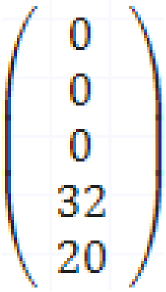
The D matrix is a c x c matrix with no non-zero elements as shown:



$$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$$

This can be ignored for the purpose of my algorithm as the 2D matrix is initialised with all zero elements.

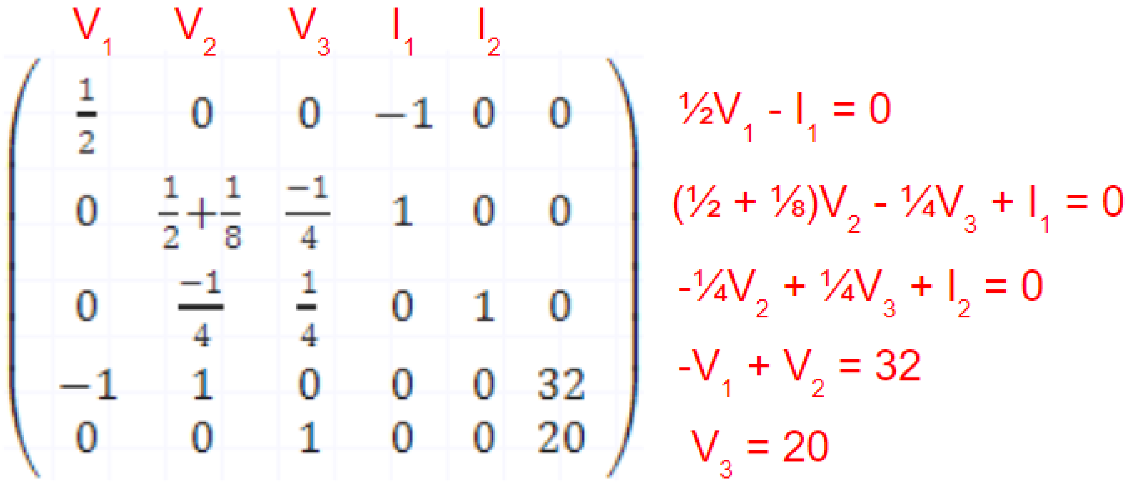
Finally, the Z matrix is an (n + c) x 1 matrix representing the constants in the system of linear equations. The first n elements are 0 and the lower c elements are the voltages of each cell. Shown below the Z matrix for the previous example and an algorithm in pseudocode which can populate it.



$$\begin{pmatrix} 0 \\ 0 \\ 0 \\ 32 \\ 20 \end{pmatrix}$$

```
For i from ungrounded.size to ungrounded.size to cellList.size
        A[ i ][ungrounded.size + cellList.size] = cellList[ i - ungrounded.size].pd
End for
```

Each of these matrices combine to form the A matrix, which represents a system of simultaneous equations like so:



$$\begin{array}{cccccc} V_1 & V_2 & V_3 & I_1 & I_2 & \\ \left(\begin{array}{c} \frac{1}{2} \\ 0 \\ 0 \\ -1 \\ 0 \end{array}\right. & \begin{array}{c} 0 \\ \frac{1}{2}+\frac{1}{8} \\ \frac{-1}{4} \\ 1 \\ 0 \end{array} & \begin{array}{c} 0 \\ \frac{-1}{4} \\ \frac{1}{4} \\ 0 \\ 1 \end{array} & \begin{array}{c} -1 \\ 1 \\ 0 \\ 0 \\ 0 \end{array} & \begin{array}{c} 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{array} & \left.\begin{array}{c} 0 \\ 0 \\ 0 \\ 32 \\ 20 \end{array}\right) \end{array}$$

$\frac{1}{2}V_1 - I_1 = 0$

$(\frac{1}{2} + \frac{1}{8})V_2 - \frac{1}{4}V_3 + I_1 = 0$

$-\frac{1}{4}V_2 + \frac{1}{4}V_3 + I_2 = 0$

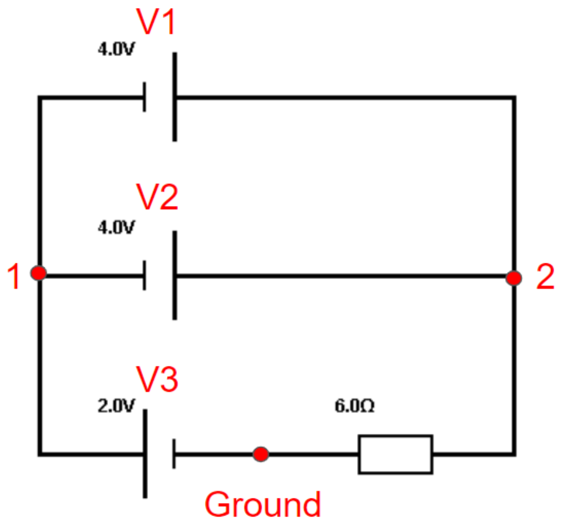
$-V_1 + V_2 = 32$

$V_3 = 20$

Such that V_n is the voltage at the nth node and I_n is the current across the nth cell. The next step is to solve this set of equations.

2.4.3 - HANDLING OF DUPLICATED ROWS

To solve any system of equations, the number of equations must be equal to or greater than the number of unknown variables. Within the MNA algorithm, if there are identical rows, there are not enough unique equations to calculate each unknown; however, I have developed a way to remedy this. Consider the following circuit and corresponding A matrix:



$$\begin{array}{ccccccc} V_1 & V_2 & I_1 & I_2 & I_3 & & \\ 0 & 0 & -1 & -1 & 1 & 0 & 0 \\ 0 & \frac{1}{6} & 1 & 1 & 0 & 0 & 1 \\ -1 & 1 & 0 & 0 & 0 & 4 & 2 \\ -1 & 1 & 0 & 0 & 0 & 4 & 3 \\ 1 & 0 & 0 & 0 & 0 & 2 & 4 \\ 0 & 1 & 2 & 3 & 4 & 5 & \end{array}$$

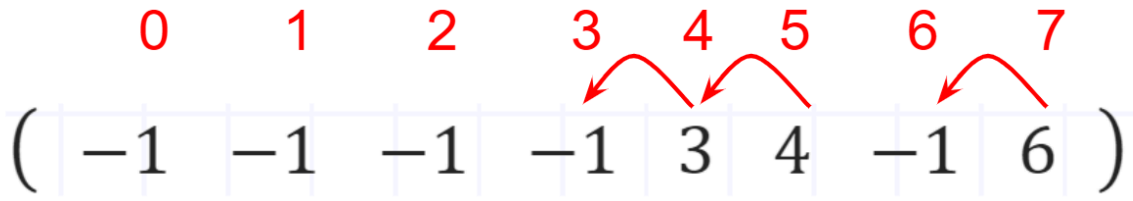
The highlighted rows 2 and 3 are identical which would typically render this system of equations unsolvable. However, by examining the circuit, it can be seen that the currents through V1 and V2 are equal (I_1 and I_2). Hence the number of unique equations is still equal to the number of unknowns. The duplicated rows and columns can be removed and added to the original column to create a smaller system of equations with an equal number of unknowns and equations like such:

$$\begin{array}{ccccc} V_1 & V_2 & I_2 + I_1 & I_3 & \\ 0 & 0 & -2 & 1 & 0 \\ 0 & \frac{1}{6} & 2 & 0 & 0 \\ -1 & 1 & 0 & 0 & 4 \\ 1 & 0 & 0 & 0 & 2 \end{array}$$

It is always the case that when row M and N are equal, the unknown variables of columns M and N are also equal. For example, the matrix above has equal rows 2 and 3, hence the variables corresponding to columns 2 and 3 are equal. To implement the elimination of rows and columns, I have developed the following pseudocode:

```
1Darray int copy[ungrounded.size + cellList.size]
fill copy with -1
int tempCopy
For i from ungrounded.size to ungrounded.size + cellList.size - 2
        For j from i + 1 to ungrounded.size + cellList.size - 1
                tempCopy = copy[ j ]
                copy[ j ] = i
                If A[ j ] != A[ i ]
                        Copy[ j ] = tempCopy
                Else
                        Add all elements from column j to column i
                End if
        End for
End for
```

This algorithm works by creating a one-dimensional array with a size equal to the number of rows in the A matrix. A for-loop iterates through each row with an index greater than the number of ungrounded nodes - 1, since these are the only rows which can be duplicates. Within this, a nested for-loop iterates through each row with a higher index. The “Copy” array assumes row j is a copy of row i by replacing Copy[j] with a value of i then checks if it is a copy. If row j is not a copy of row i then Copy[j] is restored to its original value using the tempCopy variable. The result of this process is an array where each index represents a row and the value within shows if the row is a copy and what row it is a copy of, with non-copies holding a -1 value as a null pointer. Shown below is an example of a “copy” array:



This states that row 7 is a copy of row 6 and row 5 is a copy of row 4 which is a copy of row 3. When carrying out further matrix operations, rows and columns can be ignored if the element in the copy array is not -1, creating a smaller and solvable matrix.

2.4.4 - SOLVING SYSTEMS OF LINEAR EQUATIONS

2.4.4.1 - GAUSSIAN ELIMINATION

After the matrix has been populated, it must be converted to upper triangular form using row operations. For a system of linear equations, rows can be:

- Multiplied by scalar constants.
- Swapped with other rows.
- Added to another row.

I've developed a Gaussian elimination algorithm to convert the matrix into upper triangular form, the pseudocode for which is shown below:

```
For i from 0 to ungrounded.size + cellList.size - 1 where copy[ i ] is not -1
    If A[ i ][ i ] is 0
        For j from i + 1 to ungrounded.size + cellList.size -1
            If A[ j ][ i ] is not 0
                Swap rows j and i
            End if
        End for
    End if
    For j from i + 1 to ungrounded.size + cellList.size - 1
        If A[ j ][ i ] is not 0
            Double factor = - A[ i ][ i ] / A[ j ][ i ]
            For each column in row j
                A[ j ] = A[ j ] * factor
                A[ j ] = A[ j ] + A[ i ]
            End for
        End if
    End for
End for
```

This gaussian elimination algorithm works by examining each element in the lower triangle of the A matrix for every column, and multiplying the entire row by a constant. This constant is calculated such that when the row above is added to it, the element in the lower triangle becomes zero, unless the element is already zero. This is repeated across all columns and lower triangle rows until the matrix is in upper triangular form (where all lower triangular elements are zero). However, if the diagonal element of a row is zero, that row is swapped with a row below that has a non zero element in the same column. An example of part of this process for a 3 x 3 matrix is shown below:

$$\begin{tikzcd}
\begin{array}{cc} & \begin{matrix} 0 & 1 & 2 \end{matrix} \\ \begin{matrix} 0 \\ 1 \\ 2 \end{matrix} & \begin{pmatrix} 0 & 2 & 4 \\ 1 & 6 & 5 \\ 8 & 2 & 1 \end{pmatrix} \end{array} \arrow[r, "\text{Swap row 1 and row 0}"] & \begin{pmatrix} 1 & 6 & 5 \\ 0 & 2 & 4 \\ 8 & 2 & 1 \end{pmatrix} \arrow[d, "\text{Multiply row 2 by -1/8}"] \\
\begin{pmatrix} 1 & 6 & 5 \\ 0 & 2 & 4 \\ 0 & 23/4 & 39/8 \end{pmatrix} & \begin{pmatrix} 1 & 6 & 5 \\ 0 & 2 & 4 \\ -1 & -1/4 & -1/8 \end{pmatrix} \arrow[l, "\text{Add row 0 to row 2}"']
\end{tikzcd}$$

2.4.4.2 - BACK-SUBSTITUTION

After the matrix has been converted to upper triangular form, the equations can be solved using back-substitution from the bottom row to the top row. An example of this process is shown below:

$$\begin{array}{cccc} a & b & c & \\ \end{array}$$

$$\begin{pmatrix} 1 & 6 & 5 & 20 \\ 0 & 2 & 4 & 10 \\ 0 & 0 & 2 & 4 \end{pmatrix} \begin{array}{l} a + 6b + 5c = 20 \\ 2b + 4c = 10 \\ 2c = 4 \end{array}$$

Solutions

a	b	c
20	10	4
20	10	2
20	1	2
4	1	2

$$2c = 4 \rightarrow c = 2$$

$$2b + 4c = 2b + 8 = 10 \rightarrow 2b = 2 \rightarrow b = 1$$

$$a + 6b + 5c = 20 = a + 16 \rightarrow a = 4$$

I have developed the following algorithm to carry out this process efficiently:

```
1Darray double solutions[ ungrounded.size + cellList.size ]
For i from ungrounded.size + cellList.size - 1 to 0
        If copy[ i ] is -1
                solutions[ i ] = A[ i ][ ungrounded.size + cellList.size ]
                For j from ungrounded.size + cellList.size to i + 1
                        subtract A[ i ][ j ] * solutions[ j ] from solutions [ i ]
                End for
                divide solutions[ i ] by A[ i ][ i ]
        End if
End for
```

Once this process is complete, the solutions matrix is fully populated with elements 0 to “ungrounded.size” each storing the voltage of a node and any subsequent elements storing the current through a given cell. This is excluding any elements belonging to copied variables.

2.4.5 - COPYING SOLUTIONS FOR DUPLICATE VARIABLES

As mentioned in the end of 2.4.4.2, the solutions array is missing any copied variables. Using the copy array, I have designed a module which will iterate through each element and, where the element is a copy, set the element's value to that of the element it is a copy of. The pseudocode for this process is shown below:

```
For every index in the copy array
        If copy[index] is not -1
                solutions[index] = solutions[copy[index]]
        End if
End for
```

This process will fully populate the solutions array and the elements can be transferred to the variables held within each component and node object.

2.4.6 - ASSIGNING SOLUTIONS TO COMPONENTS AND NODES

Now that the solutions for all cells and nodes have been calculated and stored, the final step is to update the values within the components themselves. This uses the nodeList, cellList and resList ArrayLists held in the main class. Firstly, part of the “solutions” array is iterated through and the elements are assigned to the “current” and “voltage” of the cells and nodes represented by the correct index. Secondly, the potential difference across each resistor must be calculated.

Once the voltage at each node has been assigned, the potential difference for a component can be easily calculated by finding the difference in voltage for each node connected to the component. The current across the component can be easily calculated using the equation V=IR (potential difference = current x resistance). Dividing the potential difference by the resistance will give the current through the component. The pseudocode for calculating current and potential difference across a component is shown below:

```
For each resistor in the resList ArrayList
    resistor.pd = resistor.port1.voltage - resistor.port2.voltage
    resistor.current = resistor.getResCurrent()
End for
```

After this, the current and potential difference across each component has been calculated, meaning the circuit is fully simulated and ready to be displayed to the user.

3 - Testing

3.1 - TESTING PLAN

Now the software has been developed to the degree specified within the Design section, I plan to carry out necessary testing to detect issues and update the software to rectify the problems. To accomplish this, I have constructed a table with the following columns:

- **Test Number and Timestamp -** An index numbering each test carried out, as well as a timestamp of when the test is carried out within a testing video.
- **Description -** A detailed explanation of the test.
- **Input Data -** The data or input provided to the program.
- **Input Type -** Whether the input data is normal, erroneous or boundary data (denoted by “N”, “E” and “B” respectively). Normal data lies within the accepted and expected range of values, whilst erroneous data does not. However, boundary data lies at either end of the acceptable range for an input.
- **Expected Output -** The expected output based on the original design.
- **Tested Output -** The output received during the test.
- **Passed or Failed -** A tick indicates that the feature being tested is fully functional, whereas a cross indicates a newly discovered failure that must be amended.
- **Changes Needed -** A brief description of a change that can be made to amend the fault, if necessary.

The test will cover all features of the software, including invalid, erroneous inputs as well as some general usage similar to that of a typical end-user. This will include the simulation of a variety of circuit types (many of which were specifically accounted for in the Design section) using all of the components available.

3.2 - TEST TABLE AND VIDEO

To provide a visual demonstration of testing as well as the software itself, I have created and uploaded a video where I conduct each test in the test table. Within this video, I will provide input data for each test and record: the output, whether the test was passed or failed, and (if the test has failed) any changes required for the current version of the software to pass said test. The QR code and hyperlink for the Youtube video are shown below:

https://www.youtube.com/watch?v=VV6T4z352hU

Time stamps have been included in the video, separating each unit of tests for the viewer's convenience. The test table is shown on the following pages, having already been populated whilst recording the video.

The testing units are as follows:

1) Manipulating the circuit layout by creating, moving and connecting components.
2) Utilising buttons on the menu whilst selecting components.
3) Updating component properties via the editor window.
4) Simulating a variety of circuits, covering most if not all of the circuit types encountered during A-level physics.
5) Simulating circuits to aid with A-level physics exam questions. These will be displayed outside of the table to make it easier to read the questions.

Test No. and Time	Description	Input Data	Input Type	Expected Output	Tested Output	Pass or Fail	Changes Required
1.1 **(01:11)**	Dragging a new component from the menu and releasing outside of the board	Dragging and dropping	E	No component is placed and the dragged component is removed	Identical to the expected output.	✓	None needed
1.2 **(01:31)**	Dragging one of each component from the menu to the board such that they do not overlap	Dragging and dropping	N	A new component is created in each space where the mouse is released	Identical to the expected output.	✓	None needed
1.3 **(01:59)**	Dragging a new component from the menu and attempting to place it over an existing component	Dragging and dropping	E	No component is placed and the dragged component is removed	Identical to the expected output.	✓	None needed
1.4 **(02:26)**	Right clicking on a component to rotate it	Right clicking over a component four times	N	Component will be rotated by 90° four times, ending as its original position	Identical to the expected output.	✓	None needed
1.5 **(03:10)**	Drawing a wire from a component to itself	Dragging and dropping	E	No connection is made	Identical to the expected output.	✓	None needed

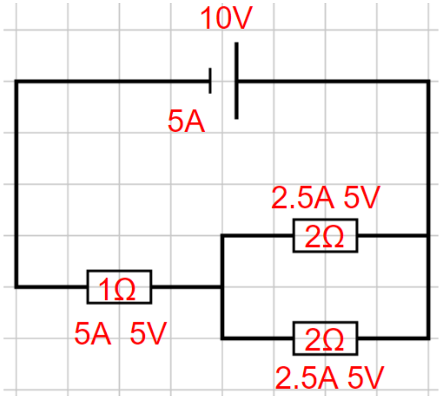
1.6 **(03:43)**	Drawing a wire from one component to another	Dragging and dropping	N	A new connection is made between the component ports connected	Identical to the expected output.	✓	None needed
1.7 **(04:07)**	Connect both ports of one component to one port of another component	Dragging and dropping	E	Connection will not be allowed as that would create a short circuit	Two connections are made	❌	Iterate through the connected list to prevent this from happening
1.8 **(05:17)**	Test “merge” function by connecting two ports that already have connections	Dragging and dropping	N	The nodes merge and the ports are connected	Identical to the expected output.	✓	None needed
1.9 **(05:59)**	Attempting to connect ports that are already connected to each other.	Dragging and dropping	E	No changes are made the the circuit	Identical to the expected output.	✓	None needed
1.10 **(06:24)**	With wire cutter activated, click a port connected to at least two ports	Left click on the port	N	The other ports remain connected, however the port pressed is severed	Identical to the expected output.	✓	None needed
1.11 **(06:55)**	With wire cutter activated, click a port connected to one other port	Left click on the port	N	The other port is severed and the node between them is deleted	Identical to the expected output.	✓	None needed

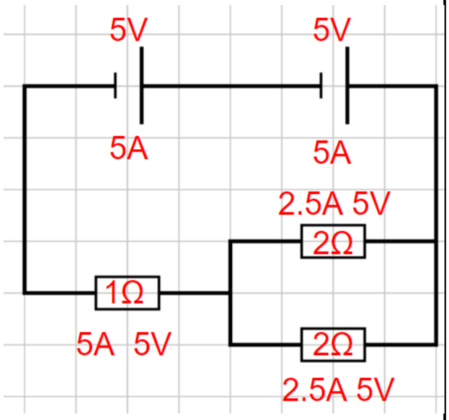
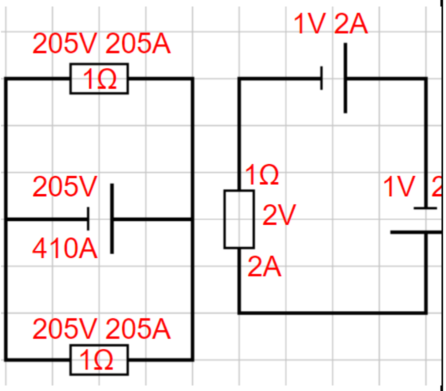
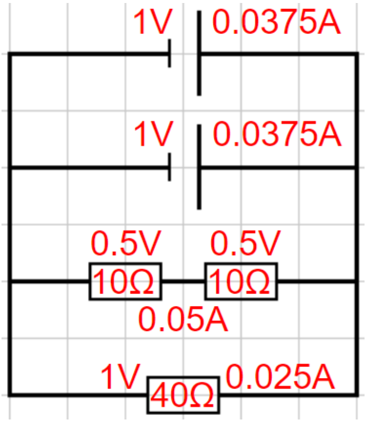
1.12 **(07:44)**	With wire cutter activated, click a port with no connection	Left click on the port	E	Nothing happens	Identical to the expected output.	✓	None needed
1.13 **(08:10)**	Drag a component to an empty location via left click and dragging then dropping	Dragging and dropping	N	The component is moved to the location of the mouse release	Identical to the expected output.	✓	None needed
1.14 **(08:37)**	Drag a component to a location that would overlap itself	Dragging and dropping	N	The component is moved to the location of mouse release	Identical to the expected output.	✓	None needed
2.1 **(09:37)**	Selecting one of each component type	Left clicking the centre of each component	N	The “Select a component” box is replaced with a screen showing info on the component. Each variable should be displayed in standard form correctly	Identical to the expected output.	✓	None needed
2.2 **(10:16)**	With a switch selected, open and close the switch repeatedly	Left clicking the Toggle open/closed button	N	The switch appears to open and close	Identical to the expected output.	✓	None needed

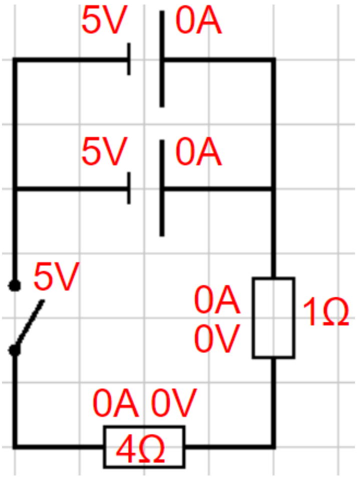
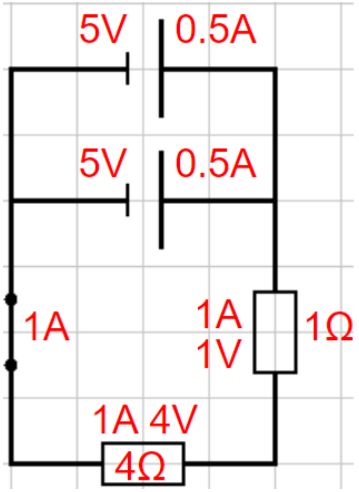
2.3 **(10:37)**	Opening and closing a switch connected to other components.	Left clicking the Toggle open/closed button	N	The switch appears to open and close, maintaining connections with other components	Identical to the expected output.	✓	None needed
2.4 **(10:59)**	Delete two of each component, one with no connections and one with connections	Left clicking to highlight a component and left clicking “delete”	N	Each component is removed from the board, and any connections made are severed	Identical to the expected output.	✓	None needed
2.5 **(12:30)**	Pressing “Simulate” with no components on the board	Left clicking simulate button after deleting all components	E	There are no visible changes	Identical to the expected output.	✓	None needed
2.6 **(12:49)**	Pressing “Simulate” with many disconnected components on the board	Left clicking simulate button after placing one of each component	E	An error window is shown stating that the circuit is invalid	Identical to the expected output.	✓	None needed
3.1 **(13:44)**	With a component selected, open the editor window	Left clicking the “Edit” button	N	Editor window opens, displaying information for the selected component	Identical to the expected output.	✓	None needed
3.2 **(14:18)**	Update properties then discard any changes using the “cancel” button.	Typing into each box then left clicking “cancel”	N	The editor window closes and the component remains unchanged	Identical to the expected output.	✓	None needed

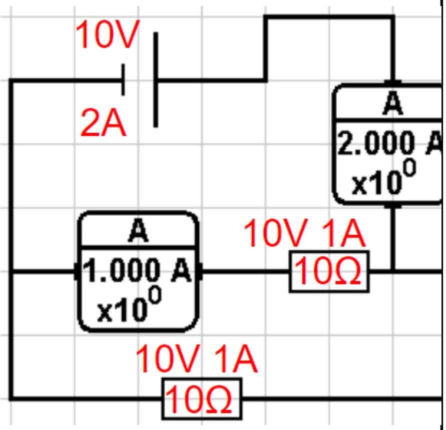
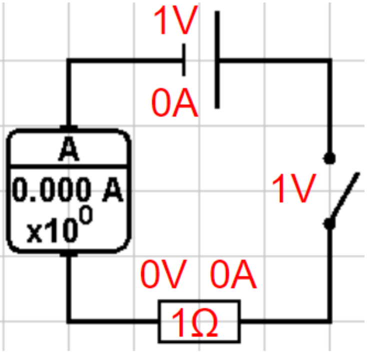
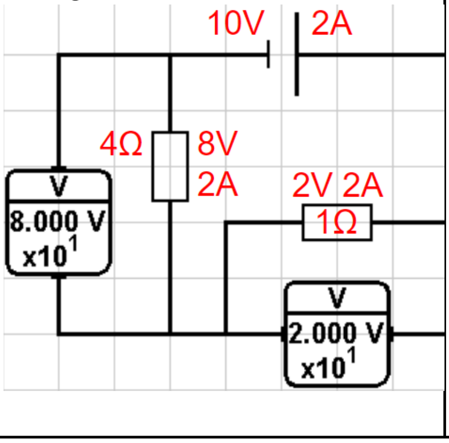
3.3 **(14:53)**	Open the editor window for one of each component type	Left clicking the “Edit” button after left clicking each component	N	Each component type only displays changeable properties for the right type	Identical to the expected output.	✔	None needed
3.4 **(15:32)**	Attempt to change the name of a component to one with over 15 characters	Left clicking “Edit” button and Typing a string into the “Name” box, then clicking “save”	E	A text box will appear stating that the name cannot be longer than 15 characters.	Identical to the expected output.	✔	None needed
3.5 **(16:15)**	Attempt to change the name of a component to an empty string	Typing an empty string into the name box and clicking save	B	A text box will appear stating that the name cannot be empty	The name was saved to an empty string.	❌	Update the if statement to exclude strings with a length of 0
3.6 **(17:24)**	Attempt to change the resistance / voltage of a resistor / cell to a negative value	Opening the editor for a resistor / cell, typing a negative string into the “resistance” or “voltage” box and clicking “save”	E	A text box will appear stating that the resistance/voltage must be greater than zero	Identical to the expected output.	✔	None needed
3.7 **(18:12)**	Attempt to change the resistance / voltage of a resistor / cell to zero	Opening the editor for a resistor / cell, typing “0” into the “resistance” or “voltage” box and clicking “save”	B	A text box will appear stating that the resistance / voltage must be greater than zero .	Identical to the expected output.	✔	None needed

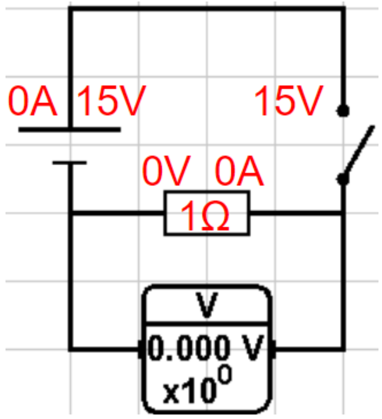
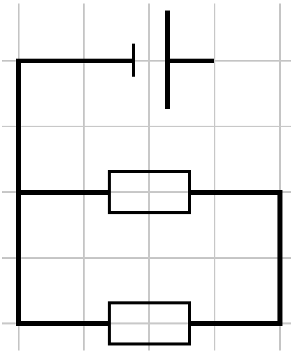
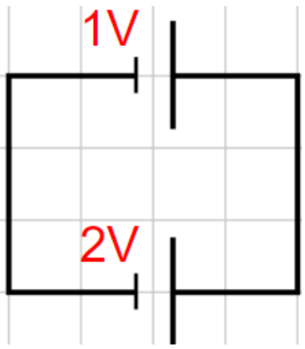
3.8 **(18:36)**	Attempting to save a string as a resistance / voltage	Entering a string that does not represent a double (such as “abc20”) into the resistance or voltage box and clicking save	E	A text box will appear stating that variables excluding name must be a number	Identical to the expected output.	✓	None needed
3.9 **(19:20)**	Attempting to change the resistance / voltage to a number with more than 6 characters	Typing a value such as “2.5101284” into the resistance / voltage box and clicking save	E	A text box will appear stating that the resistance / voltage cannot be saved to that degree of accuracy	Identical to the expected output.	✓	None needed
3.10 **(20:10)**	Attempting to save an exponent outside the range of -15 to 15	Typing an integer such as -20 into the exponent box and clicking save	E	A text box will appear stating that the exponent must be between 15 and -15	Identical to the expected output.	✓	None needed
3.11 **(20:50)**	Attempting to save an exponent between -15 and 15, however not an integer	Typing a value such as 2.7 into the exponent box and clicking save	E	A text box will appear stating that the exponent must be an integer	Identical to the expected output.	✓	None needed
3.12 **(21:47)**	Attempting to save a string as the exponent	Typing a string such as “a2b” into the exponent box and clicking save	E	A text box will appear stating that all variables besides name must be a number	Identical to the expected output.	✓	None needed

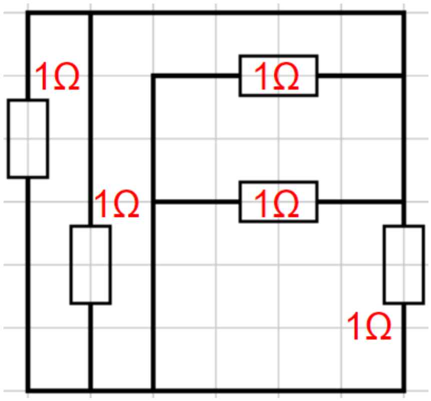
3.13 **(22:18)**	Updating the name of a component to an acceptable value	Typing a string less than 15 characters long into the name box and clicking save	N	The window will close and the name of the component will be updated to match the inputted value	Identical to the expected output.	✓	None needed
3.14 **(23:25)**	Updating the resistance / voltage of a component to a suitable value	Typing a positive, non zero resistance / voltage as a double of suitable length with an exponent between -15 and 15	N	The window will close and the variables of the selected component is updated to match the inputted values	Identical to the expected output.	✓	None needed
3.15 **(25:20)**	Open the editor window for a component with a new resistance / voltage	Clicking "edit" for a component with an updated resistance / voltage	N	The resistance / voltage will be converted to standard form for the user's convenience	Variables less than 1 are not properly rounded	✗	Round values when opening the editor window
4.1 **(28:20)**	Simulate the following simple circuit with resistors in series and parallel: 		N	The circuit is simulated successfully with the voltage and current across each component matching the diagram	Identical to the expected output	✓	None needed

4.2 **(29:14)**	Simulate the following circuit with cells in series:  5V 5V 5A 5A 2.5A 5V 2Ω 1Ω 5A 5V 2Ω 2.5A 5V	N	The circuit is simulated successfully with the voltage and current across each component matching the diagram	Identical to the expected output	✓	None needed
4.3 **(30:09)**	Simulate the following circuits simultaneously:  205V 205A 1Ω 205V 410A 205V 205A 1Ω 1V 2A 1Ω 2V 2A 1V 2	N	The circuits are simulated successfully as if they were simulated separately, with the voltage and current across each component matching the diagram	Identical to the expected output	✓	None needed
4.4 **(31:15)**	Simulate the following circuit with cells in parallel:  1V 0.0375A 1V 0.0375A 0.5V 0.5V 10Ω 10Ω 0.05A 1V 40Ω 0.025A	N	The circuit is simulated successfully with the voltage and current across each component matching the diagram	Identical to the expected output	✓	None needed

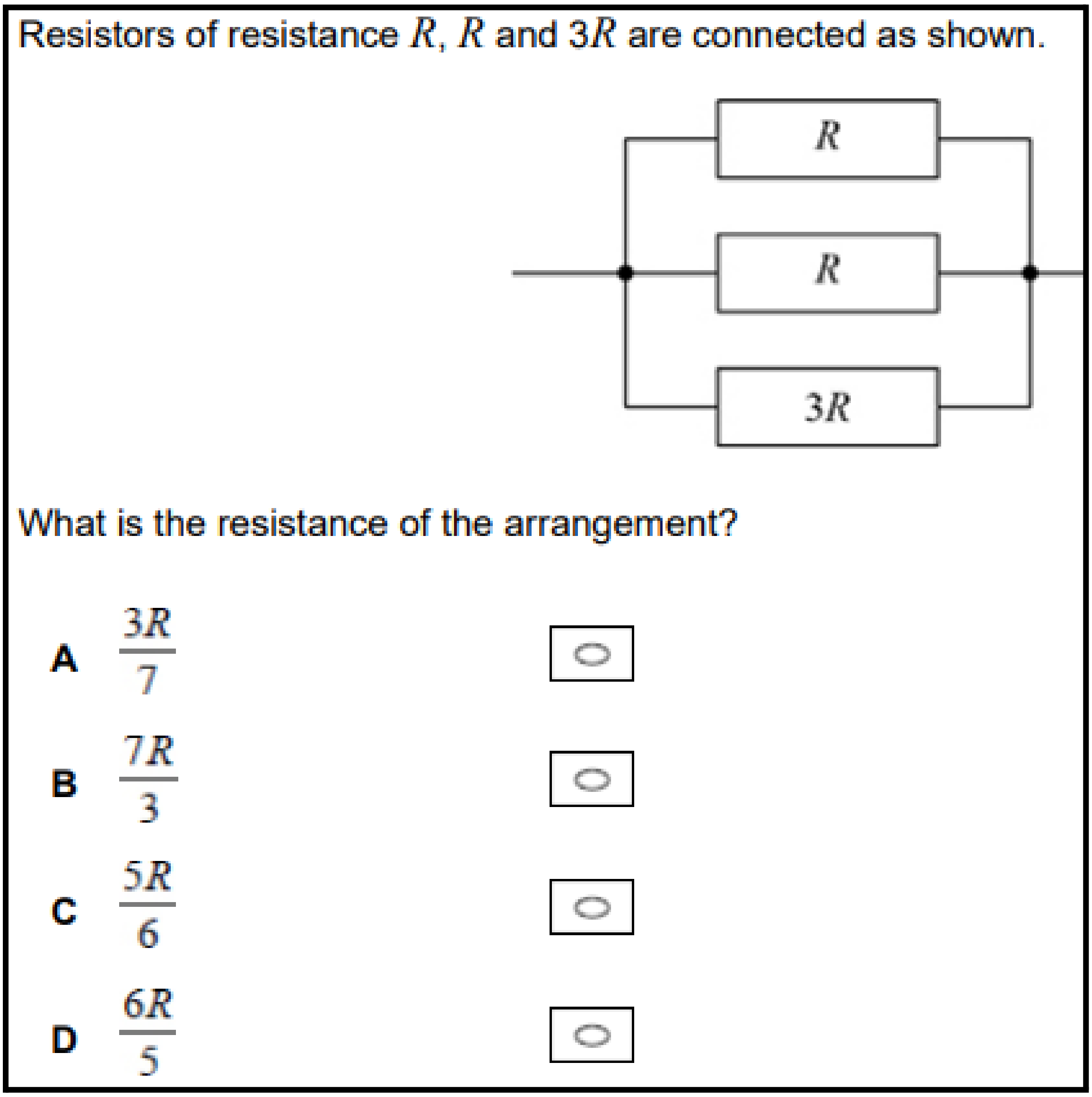
4.5 **(32:25)**	Simulate the following circuit with an open switch preventing current flow. Repeat after closing the switch:   5V 0.5A 5V 0.5A 1A 1A 1V 1Ω 1A 4V 4Ω	B	The circuits are simulated successfully with the voltage and current across each component matching the diagram	Identical to the expected output	✓	None needed

4.6 **(33:46)**	Simulate the following circuit with ammeters to display the current passing through a wire:  10V 2A A 2.000 A x10^0 A 1.000 A x10^0 10V 1A 10Ω 10V 1A 10Ω	N	The circuit is simulated successfully with the voltage and current across each component matching the diagram	Identical to the expected output	✓	None needed
4.7 **(35:05)**	Simulate the following circuit, with ammeters carrying zero amps of current: 	B	The circuit is simulated successfully with the voltage and current across each component matching the diagram	Identical to the expected output	✓	None needed
4.8 **(35:39)**	Simulate the following circuit with voltmeters to display the voltage across two nodes:  10V 2A 4Ω 8V 2A 2V 2A 1Ω V 8.000 V x10^1 V 2.000 V x10^1	N	The circuit is simulated successfully with the voltage and current across each component matching the diagram	Invalid circuit message appeared and one voltmeter did not simulate correctly	✗	Check how voltmeters can affect a circuit

4.9 **(39:14)**	Simulate the following circuit with a voltmeter reading zero volts: 	B	The circuit is simulated successfully with the voltage and current across each component matching the diagram	Identical to the expected output	✓	None needed
4.10 **(39:55)**	Simulate the following incomplete circuit: 	E	An error window is displayed stating that the circuit is invalid	Identical to the expected output	✓	None needed
4.11 **(40:21)**	Simulate the following circuit which is invalid due to having voltage without resistance: 	E	An error window is displayed stating that the circuit is invalid	Has not returned any error and set all currents to zero	✗	Analyse partly equal rows and check for rows with only a differing final element

4.12 **(41:52)**	Simulate the following circuit, containing no voltage sources:  	B	The circuit is simulated without an error message each resistor has no current or voltage across it	Identical to the expected output	✓	None needed

5.1) (43:53) Use the simulator to aid with answering the following multiple-choice exam question and/or validate any answers:



Resistors of resistance R, R and $3R$ are connected as shown.

What is the resistance of the arrangement?

A $\frac{3R}{7}$ ○

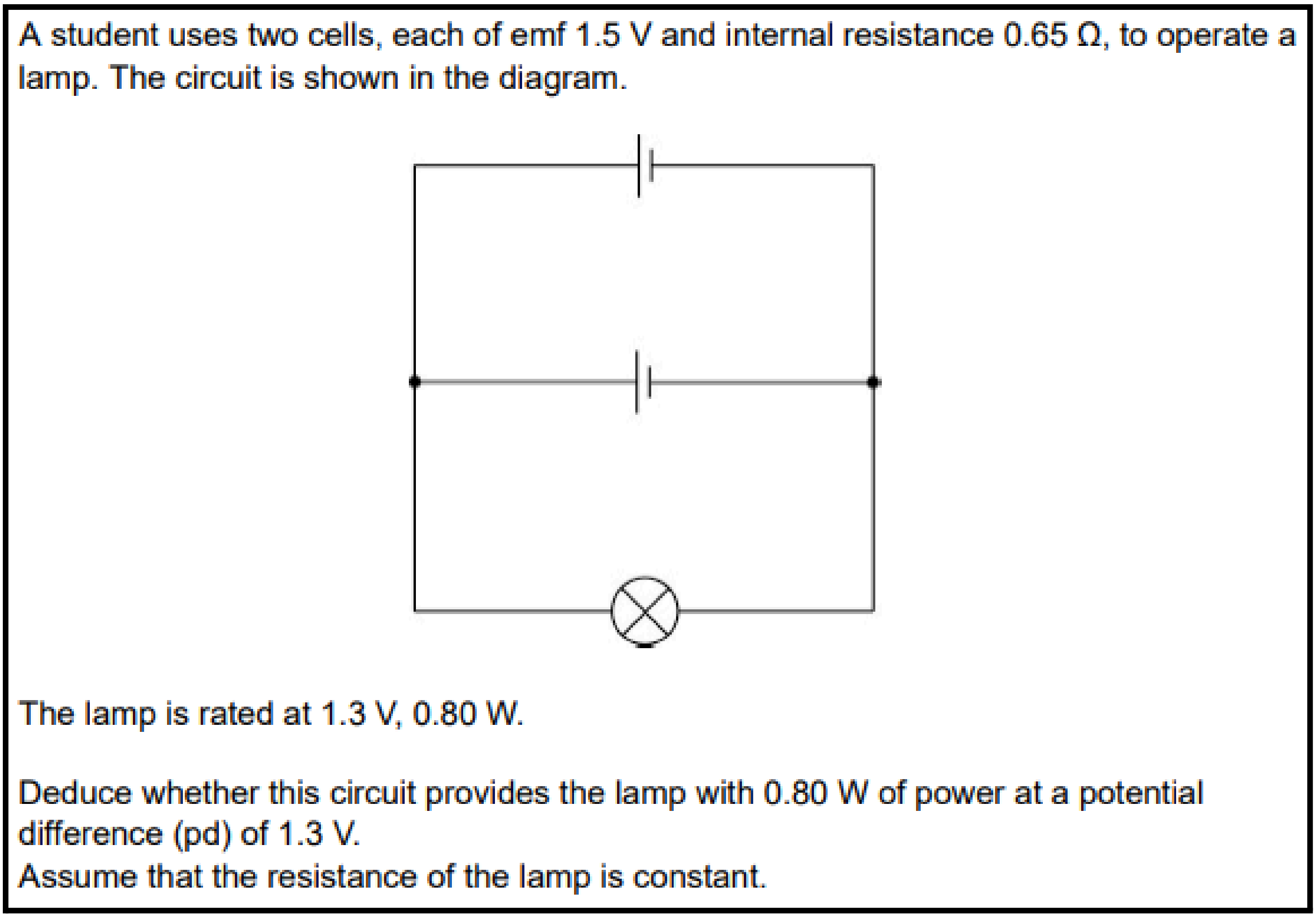
B $\frac{7R}{3}$ ○

C $\frac{5R}{6}$ ○

D $\frac{6R}{5}$ ○

Outcome - The simulator provided an accurate calculation of the current through a cell, leading to the correct answer.

5.2) (46:34) Use the simulator to aid with answering the following exam question and/or validate any answers:



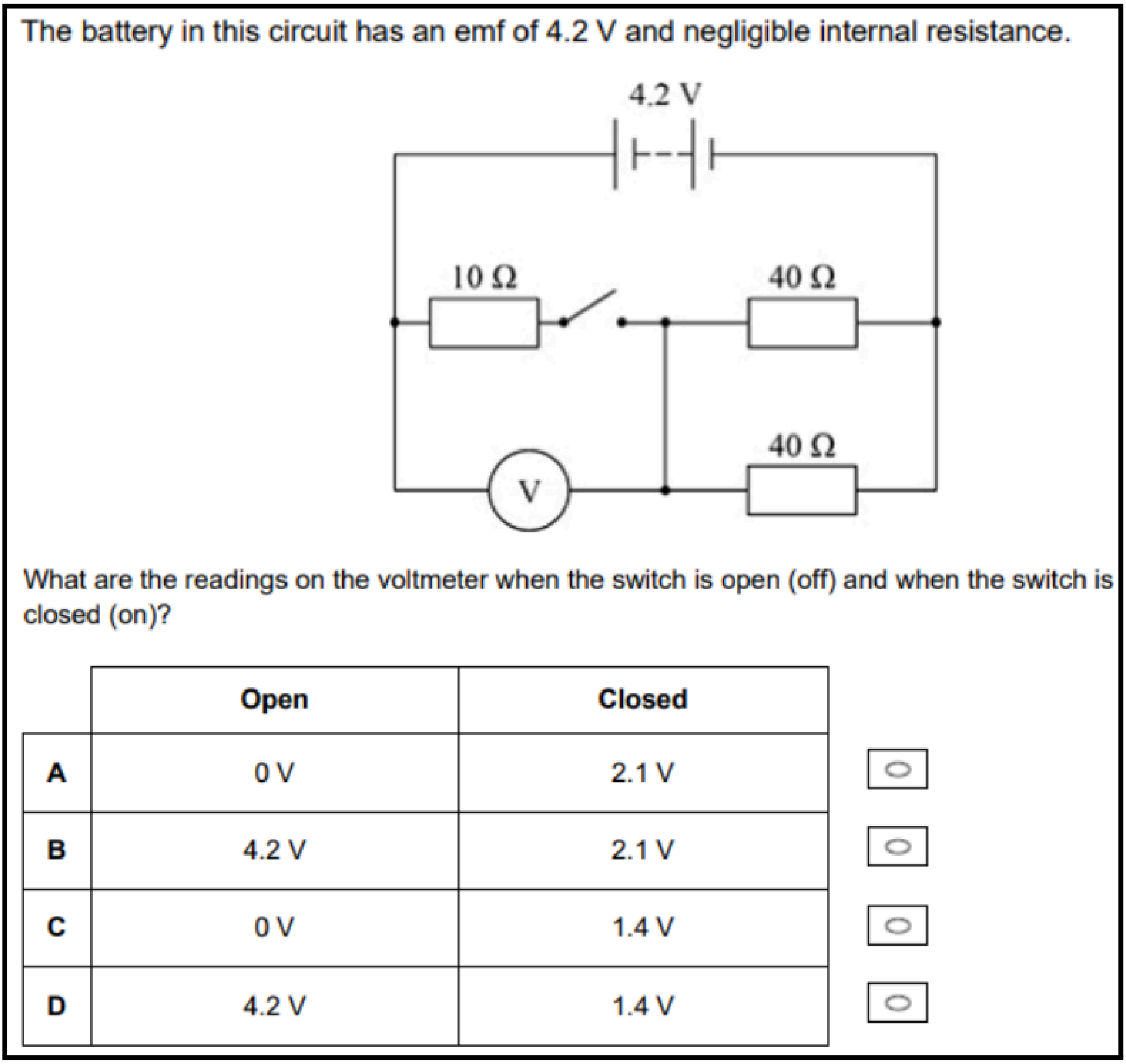
A student uses two cells, each of emf 1.5 V and internal resistance 0.65 Ω, to operate a lamp. The circuit is shown in the diagram.

The lamp is rated at 1.3 V, 0.80 W.

Deduce whether this circuit provides the lamp with 0.80 W of power at a potential difference (pd) of 1.3 V.
Assume that the resistance of the lamp is constant.

Outcome - The simulator provided an accurate simulation of the problem alongside human input in providing the resistance of each component. The simulator led to the correct answer

5.3) (50:24) Use the simulator to aid with answering the following multiple-choice exam question and/or validate any answers:



The battery in this circuit has an emf of 4.2 V and negligible internal resistance.

4.2 V
10 Ω
40 Ω
40 Ω
V

What are the readings on the voltmeter when the switch is open (off) and when the switch is closed (on)?

	Open	Closed	
A	0 V	2.1 V	○
B	4.2 V	2.1 V	○
C	0 V	1.4 V	○
D	4.2 V	1.4 V	○

Outcome - Simulator provided another accurate simulation, resulting in the correct answer with minimal human input.

4 - Evaluation

4.1 - COMPLETION OF OBJECTIVES

This section contains the objectives for the system (as previously introduced in Analysis 1.6) as well as a summary explaining the extent to which each objective was met by the final system. The summary will be based on the testing shown in Testing 3.2.

4.1.1 - USER INTERFACE

1. **User Interface** - The visible menu that the user will have access to. This will be used to view and modify circuits. The user interface must:

    1.1. Be well designed to provide a lot of room for the user to place components on their circuit as well as room for a toolbar to add components and such.

    1.2. Have a user-friendly colour scheme that does not prevent visually impaired users from accessing the software. To do so the colour scheme must adhere to the Web Content Accessibility Guidelines (WCAG) by having a contrast ratio of 4.5:1.

    1.3. Contain a list of components that can be added to the current circuit. This list must only contain what is necessary for A-level physics for the purpose of simplicity. Adding components will utilise dynamic generation of objects from a user-defined use of an OOP model, part of group A for technical skills.

    1.4. Be visually similar to standard A-level exam papers by using the same symbols for components and units to provide familiarity with the software and prevent confusion.

    1.5. Clearly present the results of the simulation for each component. This must include the current and voltage (as well as the resistance for resistors).

1.1 has been met as the board itself contains a 30 x 25 grid where each component requires a 2x2 square. This allows for the simultaneous representation of multiple complex circuits. The menu contains enough space for each component option as well as the information for the selected component, therefore the objective has been fully met in all regards

1.2 has been met and surpassed as the text is black on a white background, exceeding the WCAG guidelines as this combination has the highest possible contrast ratio of 21:1. The only exception to this is the error messages within the “editor” window. However the colour combination has a contrast ratio of 13.61:1, once again exceeding WCAG guidelines and ensuring the interface is readable for a wider range of users.

1.3 has been met as the user-interface contains a directory of each component, and the wide selection allows a user to simulate most A-level physics problems to some degree. The program also utilises OOP and user-defined methods to generate these objects.

1.4 has been met since the symbols for resistors, cells and switches are identical to that used by A-level exam boards. The voltmeters and ammeters are slightly unique to show the reading on the component itself, however the purpose of the component can be easily deduced by a user.

1.5 has been met as each variable for each component can be easily shown in standard form by selecting the component. This was demonstrated in testing unit 2.1

4.1.2 - ALGORITHM

2. **Algorithm** - The algorithm used to simulate the circuit. It operates mainly via Modified Nodal Analysis. The algorithm should:

    2.1. Be designed to handle erroneous data such as components with zero or infinite resistance and nodes that only have one connection forming a loose circuit.

    2.2. Utilise a depth-first-search for the purpose of grounding nodes in each circuit on the board. This will include Graph-traversal and recursive algorithms, both of which are part of group A for technical skills.

    2.3. Be optimised for performance to ensure that results can be calculated quickly and accurately. However since operations are not repeatedly carried out in real-time, a computer with relatively low processing power could run the program.

    2.4. Use advanced matrix operations in order to solve the system of equations, falling under group A for technical skills. This should use Gauss-Jordan elimination, graph theory and laws of electricity such as Kirchoff's laws and Ohm's law. This will allow the creation of a system of equations representing any circuit and the solving of said system.

2.1 has been partly met as voltmeters and open switches represent infinite resistance whilst ammeters and closed switches represent zero resistance. Circuits with loose connections can be detected and reported by the program and ammeters, voltmeters and switches can be simulated. However, the program is unable to handle infinite current due to no resistance in a circuit, as demonstrated in testing unit 4.11.

2.2 has been met as the depth-first-search subroutine within the "Main" class is fully operational and has been tested by simulating multiple circuits at a time in testing unit 4.3. The routine effectively and efficiently grounds one node from each subgraph.

2.3 has been met as the entire modified nodal analysis routine has no noticeable delay or time to perform on an average computer. However, there may be room for improvement through further optimisation. In doing so, the program could be altered to consistently carry out the simulation over time, therefore being able to represent time-dependent effects such as gradual heating of components. The paint method and methods for mouse inputs are carried out quickly, allowing for the window to be repainted promptly after each input, leading to no noticeable lag or performance issues.

2.4 has been met as the program can effectively and efficiently translate a matrix into row echelon form through gaussian elimination and solve it further to determine the value of each variable in the system of equations that it represents. Furthermore, the algorithm can handle identical rows and variables when they occur, leading to an accurate representation of cells in parallel as shown in testing unit 4.4. The matrix is created using the electrical laws proposed by Kirchoff and Ohm and is accurately solved to provide the current and voltage across each component.

4.1.3 - OBJECT ORIENTED PROGRAMMING MODEL

3. **Object Oriented Programming Model** - The model used to provide data structures and methods to carry out the modified nodal analysis algorithm as well as provide classes relating to the user interface. The OOP Model should:
    3.1. Represent the circuit as a graph with edges being components and vertices being nodes, using Java and its object oriented programming paradigm. Inheritance and polymorphism will be used to create an efficient model of the problem. Graphs and complex OOP models fall under group A for technical skills.
    3.2. Provide classes and methods allowing creation of a user-interface through the Java Swing and AWT packages.

3.1 has been met as each node uses a list of components where each component contains at least two nodes. These objects and their lists can be represented as a graph with adjacency lists stored about each node as mentioned in Design 2.3.4. The program effectively uses inheritance and polymorphism for both the components and the GUI.

3.2 has been met as the “Canvas”, “Builder” and “Editor” classes successfully create a user interface through inheritance of classes from the Swing and AWT packages.

4.1.4 - USER INTERACTIVITY

4. **User Interactivity** - The features that are available to the user to customise their circuit and more. These features will include:

   4.1. The ability to modify the properties of any component to copy questions a user has seen or to investigate how changing properties will affect the circuit. This should be convenient and easily accessible for the user, since most components will have to be modified to suit their needs.

   4.2. Having a convenient way to create circuits by selecting components from the list and dragging them onto the circuit, then being able to connect components with wires. The process should be simple and easy.

4.1 has been met as the component selection feature allows the user to quickly update the variables of a component, with convenient features such as a standard form input option; this was demonstrated throughout testing units 2 and 3. The simple dragging and dropping as well as the wire cutter tool allows for fast modification of a circuit’s layout, as shown within testing unit 1.

4.2 has been met by constantly showing buttons on the menu that can be dragged from and released onto the circuit, in order to quickly create a new component. The wire drawing tool is simple and ports near the mouse are highlighted to indicate a suitable place to release the mouse.

4.2 - USER FEEDBACK

In order to gain feedback on the final product, I provided my current physics teacher Mr Davison with a copy of the program. Similar to Mr Allan, he is also looking for interactive ways to demonstrate electricity and was eager to test my program. After a demo where Mr Davison used the simulator for a variety of tasks, he wrote the following review in the form of an email:

This program meets the needs of the student and teacher within a number of parts of the A level Physics specification, including resistors in series and parallel, potential dividers with basic resistors as well as EMF’s and internal resistances. To extend upon this Thomas could develop the program to include other components such as bulbs, diodes, LDR’s, capacitors and thermistors.

It is extremely easy to set up circuits using the components available. In fact, by dragging the wires the program completes the circuit for the user. This element goes beyond other programs available online which will increase speed / efficiency for the user. It was easy to insert ammeters and voltmeters within the circuit which would allow the display of currents and potential differences. It was also very useful to be able to click on any component to see the current and p.d across it. This allows the user for quick checks at any point / position within the circuit.

The program was challenging to use on my laptop and the graphics were different which did not meet the initial needs of the program. However, any challenges were easy to overcome with verbal instructions. Written instructions inbuilt within the program would be beneficial for initial use.

Another issue was that cells in parallel were only working if they contained two identical cells. This does not take into account having two branches of different cells but an equal sum of EMF’s. This could be fixed as an improvement.

Thomas may wish to incorporate other elements of the Physics specification within the program, such as the ability to collect data and plot graphs linked to components. i.e. to explore I – V characteristics.

This is an impressive program for a student to complete within an A level computing project and I will certainly use going forward within the A level Physics course.

Mr Davison

Any improvements that this feedback suggests will be explored in section 4.3.2.

4.3 - POTENTIAL IMPROVEMENTS

In accordance with the testing results and the review from Mr Davison, I have devised a series of improvements and additions that could be implemented for an updated version of the final product. These will either: fix vital issues with the program, add new and relevant features or improve current features.

4.3.1 - TESTING IMPROVEMENTS

This section will show improvements based on issues brought up within the testing video and table of section 3.2. These improvements will ensure that each test failed would be passed upon its implementation.

Firstly, the program allows a user to connect two ports of one component to the single port of another component. This was shown in test 1.7 and is easily fixable by adding a conditional to the "connect" method that ensures that when connecting two components, the other port of the first component cannot be connected to the port of the second component.

Secondly, in test 3.5 it was noticed that the name of a component could be set to an empty string, which should not be allowed. This can be simply amended by updating the conditional that checks if the string is less than 15 characters to exclude strings of length zero.

Additionally, in test 3.15, the program had not correctly rounded a variable to four significant figures. However, I have been unable to replicate this issue through simulation or using the editor window. Nethertheless, this could be fixed by investigating how the code that handles rounding deals with variables less than one where the exponent in standard form is negative.

Furthermore, in test 4.8, the program was unable to represent a circuit with voltmeters on the first attempt. After repeating the test meticulously, I've noticed that the error only occurs when the components are connected in a specific order, although the reasoning behind this is unknown. To amend this, I would look conduct further troubleshooting with this example, to determine if the issue was caused by the grounding of a specific node or an issue with the infinite resistance of a voltmeter.

Finally, in test 4.11, I had discovered that a circuit with infinite current due to a short-circuit did not return an error when expected. After reviewing the console and the matrix constructed from the circuit, it was apparent that the issue lied with rows having equal coefficients of variables yet different constants. This is impossible to solve as (for example) if 2x + y equals 5, 2x + y cannot equal 3 as well. The problem can be solved however, by adding a section to the MNA method that scans each row to ensure that no rows share the same variables yet a differing constant.

4.3.2 - FEEDBACK IMPROVEMENTS

In light of the feedback from Mr Davison in section 5.2, I've listed some useful and beneficial improvements that were suggested, as well as any amendments to errors found by Mr Davison.

Firstly, it was suggested that capacitors be implemented, in order to cover all of A-level electricity. A brief introduction for doing so is found in the guide by Swarthmore College, shown in section 1.3.1, hence it is possible with the MNA method I have used. This would also require a dimension of time within the program as a capacitor must be charged and discharged over a period. A time feature would also allow the addition of components heating up over time and gaining resistance, allowing for a more realistic simulation of electricity.

Building on the addition of new components, it was also suggested that non-ohmic conductors (resistors with a non-constant resistance) be added. This could be easily done by changing how voltage is used to calculate the current though these resistors, using an equation that increases resistance for a greater voltage. As well as this, diodes have been suggested, being a component with infinite resistance when a voltage is applied in one direction, and little resistance when a voltage is applied in the opposite direction. This could be achieved in the same way, by using an equation to calculate current from voltage that matches how a diode behaves.

Furthermore, it was suggested that a graphing feature could be added, where a user could view a graph demonstrating how current and voltage varies for a resistor. This would be particularly useful for the non-ohmic conductors and diodes mentioned previously, as the graph would be non-linear demonstrating how their resistance can change. This would be relatively simple to implement as a graph could easily be drawn using the component's relationship between current and voltage and the Java AWT package.

Additionally, an issue was raised where cells in parallel were improperly simulated in some situations. This was only when the parallel branches had equal voltages, yet a different arrangement of cells. For example, having a 2 volt cell in one branch and two 1 volt cells in the other should work, however the program failed to recognise this. Fixing this issue would require a complex new section to the MNA algorithm that can merge cells in series and parallel to form a single cell, which would have to be done through recursion as there could be a series connection within a parallel connection and so on. As a bonus, this would lead to a smaller matrix size as individual nodes and cells within a set of cells could be ignored, improving performance by decreasing calculation times.

In terms of the user interface, it was noted that the graphics were improperly displayed on Mr Davison's laptop, perhaps because the resolution was different to the 1080p I had used across all devices throughout the implementation. This is particularly problematic as the interface is pixel-precise when it comes to the circuit diagram and any text. This could be solved by investigating how the Java Swing and AWT features can account for differing resolutions and ensuring that all resolutions can neatly present the GUI to the user. Aswell as this, it was suggested that a help menu be added to display instructions and guidance to an unfamiliar user.

Besides these suggestions and noticed faults, Mr Davison is pleased with the final product and finds it useful, and simple when it comes to teaching and learning A-level electrical theory.