Benjamin Pomfrey

Seaford Head School  [Company address]

Physics Circuit Simulator

Contents

[Analysis 3](#_Toc165662889)

[Problem Definition and Broad Aims 3](#_Toc165662890)

[Project Background 4](#_Toc165662891)

[Research 4](#_Toc165662892)

[Existing solutions 4](#_Toc165662893)

[Survey 7](#_Toc165662894)

[Data structures and algorithms 8](#_Toc165662895)

[Physics Calculations 11](#_Toc165662896)

[Python Libraries and Modules 19](#_Toc165662897)

[Objectives 21](#_Toc165662898)

[Prototyping 26](#_Toc165662899)

[User Interface Initial Ideas 26](#_Toc165662900)

[Initial Class Diagram 28](#_Toc165662901)

[Design 29](#_Toc165662902)

[Class Diagram 29](#_Toc165662903)

[Event loop flowchart 32](#_Toc165662904)

[Packages, Libraries, and Modules 33](#_Toc165662905)

[Tkinter 33](#_Toc165662906)

[Pygame 34](#_Toc165662907)

[Threading 34](#_Toc165662908)

[OS (allows use of the operating system) 34](#_Toc165662909)

[Pickle 35](#_Toc165662910)

[Math 35](#_Toc165662911)

[Files, Data Structures, and Algorithms 35](#_Toc165662912)

[Circular Buffer/Stack 35](#_Toc165662913)

[Graph 38](#_Toc165662914)

[Graph cycle detection algorithm (using DFS) 39](#_Toc165662915)

[Vectors 39](#_Toc165662916)

[Undo/Redoing moved components using files 40](#_Toc165662917)

[Screenshots 42](#_Toc165662918)

[Loading images 43](#_Toc165662919)

[Finding nearest available connectors 44](#_Toc165662920)

[Physics Calculations 46](#_Toc165662921)

[Electron movement 47](#_Toc165662922)

[Unit Graph 49](#_Toc165662923)

[User Interface 50](#_Toc165662924)

[Testing 54](#_Toc165662925)

[Test Table 54](#_Toc165662926)

[Screenshot taken during testing. 66](#_Toc165662927)

[Technical Solution 67](#_Toc165662928)

[Contents 67](#_Toc165662929)

[Code 69](#_Toc165662930)

[Evaluation 122](#_Toc165662931)

[Meeting objectives 122](#_Toc165662932)

[Objective 1: 122](#_Toc165662933)

[Objective 2: 123](#_Toc165662934)

[Objective 3: 124](#_Toc165662935)

[Objective 4: 124](#_Toc165662936)

[Objective 5: 125](#_Toc165662937)

[Objective 6: 126](#_Toc165662938)

[Successes and improvements 126](#_Toc165662939)

[Classes and object creation 126](#_Toc165662940)

[Circuit Graph, cycle detection, and electrons/vectors 126](#_Toc165662941)

[Undo-redo system 127](#_Toc165662942)

[Physics calculations 128](#_Toc165662943)

[Threading for tkinter and pygame 128](#_Toc165662944)

[Dragging and creating wires 128](#_Toc165662945)

[Screenshotting 129](#_Toc165662946)

[User Feedback 129](#_Toc165662947)

[General improvements 130](#_Toc165662948)

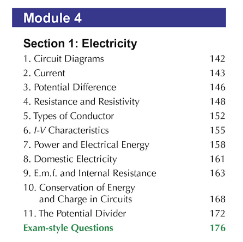
[Overall evaluation 131](#_Toc165662949)

# Analysis

## Problem Definition and Broad Aims

My Problem that I aim to solve is the lack of resources for A-Level physics students studying the current electricity module. Current electricity includes circuits and their components (filament lamps, resistors, thermistors, etc.) along with the different ways the components interact together and the different laws made by scientists like Kirchhoff or Ohm.

My program will be a circuit creator which will allow for the covering of the topics necessary.

My target audience is A-Level physics students who are studying Module 4.1-4.3 for OCR. My solution intends to act as a revision resource and aims to help them learn and improve their knowledge on current electricity. However, this can also be used by teachers as a resource to help the class when in a lesson.

Here is an example from the OCR physics A-level specification for the Module 4.1-4.3 on Current Electricity. It includes circuit diagrams and a range of components such as ammeters for current, voltmeters for potential difference and e.m.f. and all the different types of resistor like thermistors, LDRs and simple fixed resistors.

The main focus of my program will be on the circuit diagrams and how the different components work. This allows me to cover most of the specification from this section.

I aim to create a GUI which allows the user to build their own physics circuit however they want to experiment. This will allow them, as a student, to practice their skills that they feel they need to work on in an open environment. As a teacher, my program can also be used to help teach a class with the aid of a visual diagram and enrich the students’ knowledge further.

I aim to include all aspects of module 4 which require a circuit diagram, for example the ways current and potential difference act in series and parallel.

## Project Background

I have chosen to make a physics circuit simulation because I’ve always been interested in physics and enjoy learning about it, however it was always a struggle trying to get a visual representation of circuits outside a lesson that is tailored towards the A-level specifically.

For example, drawing circuit diagrams can give a good visual representation but there is lots of opportunity to make errors and potentially learn it incorrectly. Some of the existing circuit simulations can also give a good visual representation but they are not tailored to the OCR specification and may use images rather than circuit symbols.

I believe I can solve the problem of there being not enough resources to give a reliable representation of circuits. Designing and programming this simulation will not only allow the users to improve their knowledge of physics or help teach others, but it will also improve my own as I have to design and understand everything to be able to code it accurately.

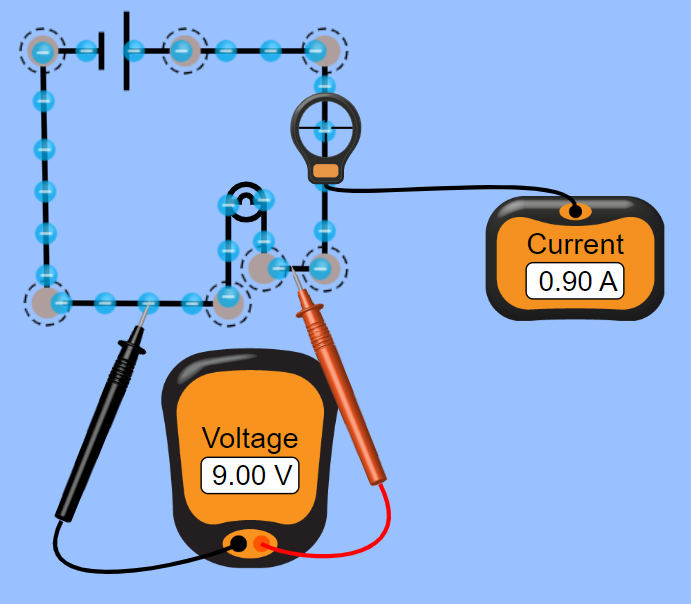
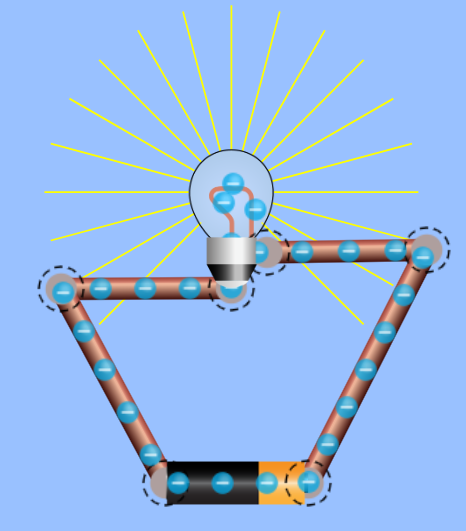
## Research

### Existing solutions

PhET Circuit Construction Kit

PhET is an interactive simulation science website covering multiple different areas of each science, including a circuit construction kit for physics. [*PhET Circuit Construction Kit*](https://phet.colorado.edu/sims/html/circuit-construction-kit-dc-virtual-lab/latest/circuit-construction-kit-dc-virtual-lab_en.html)

Phet allows you to make circuits using multiple components such as batteries, switches, and resistors. It also lets you switch between picture and diagram form for the components so you can use circuit symbols rather than pictures if needed. It also allows you to see the direction of flow of the electrons in the circuit and use voltmeters and ammeters.

Here are two circuits showing the different ways that you can represent the components and use voltmeters and ammeters.

However, it does not allow you to use more complicated components such as a diode or a variable resistor.

It also represents both the voltmeter and ammeter incorrectly as you cannot hover the ammeter over the circuit when experimenting in person, and these voltmeter and ammeters are not represented as circuit symbols, even when the symbol mode is on. In addition to this, PhET is outdated for some of the symbols like the bulb and is based in the US meaning some of the symbols are also different.

It also doesn’t have a learn or testing mode where you can test yourself on your learnt knowledge, as it is purely a free build circuit simulator. It does however have a tutorial which I will now consider making one of my own for new users.

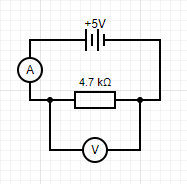
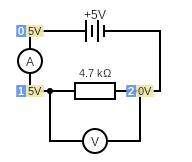
My solution will allow for accurate representation of circuit components such as an ammeter and include many more circuit components which are required to know on the A-level specification. My solution will also include a learning and testing mode which will allow people using my program to improve their circuit knowledge, and understand the current electricity modules more.

Circuit Diagram Editor

Circuit diagram editor is an online program where you can build many types of circuits and then simulate them. This ranges from the physics circuits and their components to Boolean logic circuits and much more with many complicated and intricate components. [*Circuit Diagram Editor*](https://www.circuit-diagram.org/editor/)

Although this editor can do a lot with a wide range of components, it can be quite daunting for a new user or someone trying to use it for physics when there are lots of other not relevant components which are not organised into different sections.

It is one of the smaller (less well known) existing circuit simulations available and in terms of physics, has one type of viewing format using the symbols, rather than an actual image.



Here is a simple circuit made in the edit mode and put into the simulation. It allows you to accurately represent voltmeters and ammeters using circuit symbols and has the option to use the European symbol for the resistor, despite being based in the US.

It also shows the voltage at different points in the circuit when in the simulation (as seen on the right) and so is much more visually accurate than the PhET simulation I’ve previously looked at.

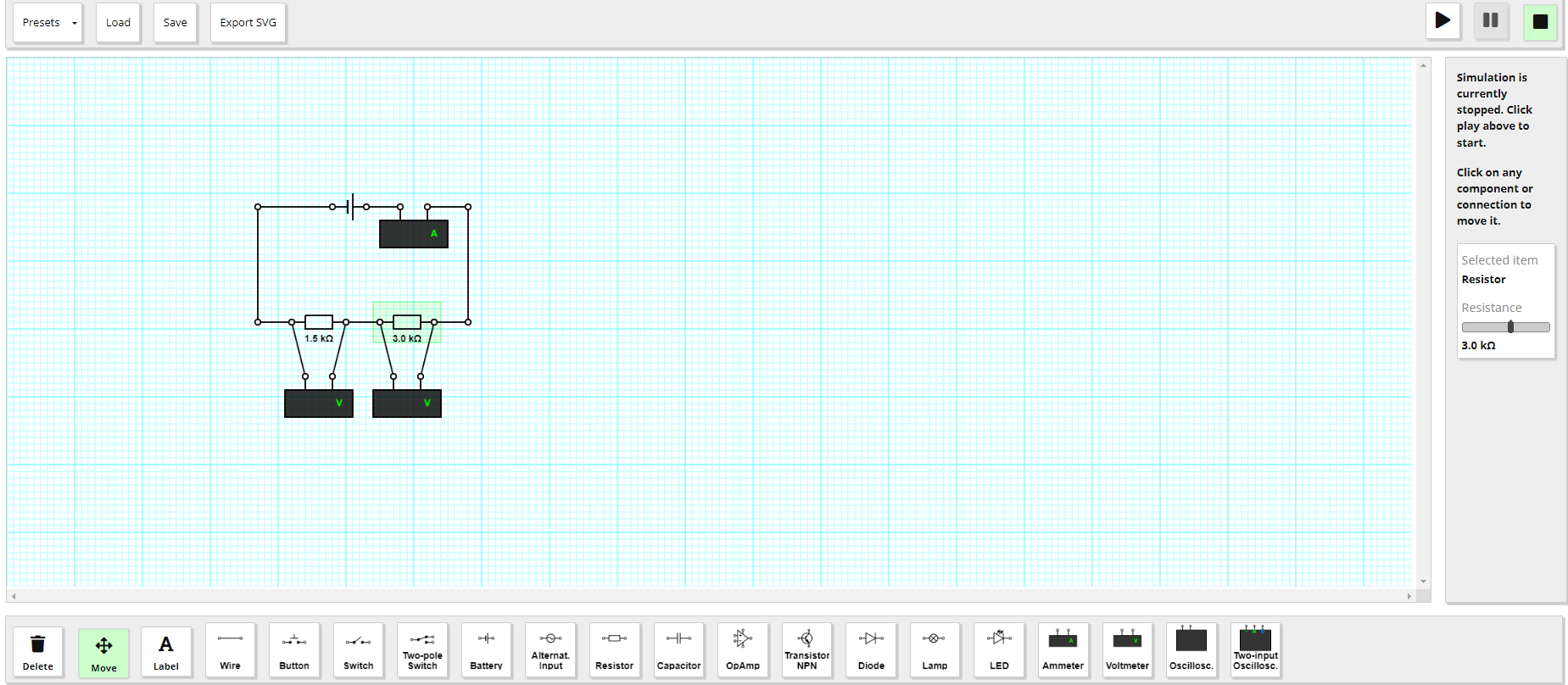
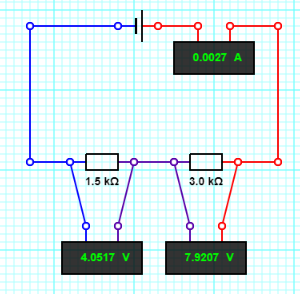
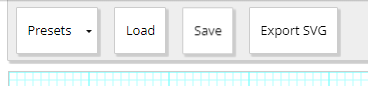
This is one of my aims for the program user interface (UI) as I think it would be beneficial top students using the program to be given examples of what it would look like in a physics exam.

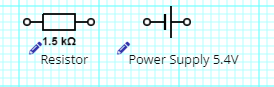
ScienceDemos Electronic Circuit Simulator

ScienceDemos is an online website that has many programs and simulations for maths, physics, and even how different colours are seen. The programs were all coded in JavaScript by Dominic Ford who works in the Astronomy Institute at Cambridge as a software developer.

[*ScienceDemos Electronic Circuit Simulator*](https://sciencedemos.org.uk/electronics.php)

The Electronic Circuit Simulator allows you to make circuits with 17 different components, allowing you to edit the values of each such as the value of the emf, or the resistance or a resistor. Once done, you can then run the circuit to show values on any components that are connected.





The visual representation is fairly accurate as it uses most of the correct symbols for all the components, however, the voltmeter and ammeter are not represented with the correct symbols and are instead just boxes which display values.

There are also pre-made templates to use by clicking the “presets” button such as a potential divider circuit to show how voltage is divided amongst resistors and even more complex presets such as a “charging/discharging capacitor” or an “OpAmp: Inverting Amp”.

These more complicated circuits, however, are outside of the A level specification and so can be daunting to those using it when trying to learn more and practice for their exam. I intend to make my program designed specifically for the specification of OCR and so this will hopefully not be a problem.

### Survey

To gather information from potential users, I sent a survey out to some of the other physicists taking the course as well as a couple of the teachers.

1. Do you think students would benefit from having a physics circuit kit to aid them revise and why?

For this question the general response was yes.

Students said that it would be particularly useful for the “more confusing aspects of circuits, e.g. the splitting of current and voltage in parallel”, and “It would be helpful for circuit diagrams and practical questions” as it would provide a “hands on experience” without the need for physical components.

Teachers said that it would be very useful as “electricity is a conceptual part of the specification” and “an interactive visual model would greatly aid learning in a topic that is conceptually difficult”.

1. Multiple modes will be included such as free build, learn, and test. What other modes do you think would benefit the program and/or what featured would you like to see?

For this question, students suggested a flash card minigame to test the user’s knowledge of different circuit components and that it would be good to include statistics such as drift velocity or perhaps resistivity. It was also suggested that simulation controls are added to allow the user to change the simulation speed and component values.

Teachers suggested that I make levelled versions of the test feature with different difficulties e.g. easy being symbol identification and harder being working out current or voltage.

1. What topics from module 4 electricity do you believe are the hardest and would be the most important to include?

For this question, students said that it would be best to try and include as much as possible as different people may have different areas that they struggle on. They also said that Kirchhoff’s Laws were particularly hard and so would be good to include.

Teachers said similar, talking about potential difference and how it is distributed is series and parallel leading to Kirchhoff’s Laws. They also suggested that this will most likely be very difficult to put into code.

1. What design and layout for the screen do you think would be most user friendly and what should be easily accessible?

For this question, it was suggested that I make sure to include “the drag and drop components” as this is necessary for circuits to be made well and as freely as I need them to be.

1. Do you think *you* would use the circuit kit to revise or teach lesson and why would or wouldn’t you?

For this question, students said that it would be very useful for them in revising because “it would provide a visual representation and an interactive learning experience” and “definitely be good for the Kirchhoff Laws”. There was however the point made that when revising, this method uses an electronic device which would distract them on some occasions.

Teachers said that it would be useful to teach a lesson as it would help “improve understanding” and it would be very helpful using “visual models to aid teaching of circuits”.

### Data structures and algorithms

Stacks

Stacks are a last in first out data structure meaning the last item that was inputted is the first returned when an item is removed.

Using a stack may be beneficial in my program because they allow for the user to be able to undo and redo their past actions. When the user clicks an undo button, the last action can be popped off the stack and returned, which can then be pushed onto the redo stack. Stacks may also be able to be used for picking up a single component if they are overlayed/in the same position.



Graphs

Graphs are groups of nodes (points) which have various edges (lines) connecting different nodes together and are used to record different connected relationships. Each node connected to another node is called its “neighbour”.

Graph will be useful in my program because they can be used to represent the circuit that the user is building. Each component can be represented as a node and each wire can be represented as an edge. The graph used would be unidirectional and unweighted, meaning there is no specific direction of travel or length of each wire (since the effect of length on resistance is negligible).

Graph Traversal Algorithms

There are multiple graph traversal algorithms that can be used to iterate through a graph or tree, two of the algorithms being depth first search (DFS) and breadth first search (BFS).

Depth First Search

Depth First Search works by starting at a given vertex and following down the next available vertex. It traverses as far as it can down one branch before backtracking up to the next non-explored node and repeats recursively. The end of a branch is defined when the current node has no neighbours which haven’t already been visited.

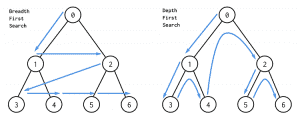
DFS uses a stack to keep track of which node the algorithm is currently on. When the program backtracks, it pops off the items in the stack to focus on the last node once a branch/section has been fully explored. Once all the nodes have been visited the traversal is complete.

Breadth First Search

Breadth first search works by starting at a given vertex (like DFS), before visiting every node that the current node is connected to (its neighbours). The algorithm then visits the first neighbour and finds all nodes connected to that, before moving onto the second neighbour and repeating. This continues on iteratively, stepping through the graph each time and going 1 node deeper, before all nodes are visited.

BFS uses a queue, rather than a stack. Each node it visits is added to the queue, which is then iterated through, de-queueing each item and visiting all the neighbours of that node (and adding them to the queue), before repeating the process.

Here is an example of a BFS (left) and DFS (right) for a tree, which is a specific type of graph:



Graph traversal algorithms will be useful in my code because they will be able to be used for detecting if the circuit the user has made is complete (if the graph contains a cycle/loop), and so can calculate other physics calculations. Graph traversals may also be used for checking if a specific component is in the graph and/or what connections it has by traversing around it and checking each node.

My program is more likely to use depth first search as this will cause the traversal algorithm to move around the circuit much quicker as it can simply follow straight around for a series circuit and down each parallel sector in advance for parallel circuits, due to it going to the furthest possible node down a branch.

Vectors

Vectors are a way of representing the transition between two positions or translation of an object and they can be represented in multiple ways.

Vectors will be useful in my program because it will allow electrons to traverse the circuit on a specific path for each wire and component. Using vectors will also be useful because it allows me to control the speed of the electrons using the magnitude and step of each position increment. Vectors may also be useful for finding the closest node on the graph to allow the user to connect and/or pick up wires for building and completing the circuit.

Queues

Queues are a first in first out data structure (FIFO), meaning the first item entered into the queue will be the first to leave, and the most recently entered item will leave last. They use pointers to track where the start of the queue is in an array. They can also be implemented as a reverse queue which means the items get removed from the rear rather than the front.

In addition to this, queues can also be implemented as a “circular” queue, which uses pointers, except when the front/rear reaches the lowest or highest index position, they wrap around to the opposite end, provided there is enough space.

Here is an example is a circular queue as a diagram:

A diagram of a circle with numbers and arrows

Description automatically generated

In my code a queue may be used for:

* Storing actions to then be put into undo and redo stacks.
* Picking up items if they are overlayed when editing the circuit – this will allow the user to pick up one item at a time rather than dragging all the components at once.

Queues are likely to be used in my code because they can provide…

### Physics Calculations

Circuit Components

My program will include all the circuit components in the OCR module 4 specification which are relevant to a circuit builder. This means that some components such as a motor will not be included as they act in the same way as a basic resistor.

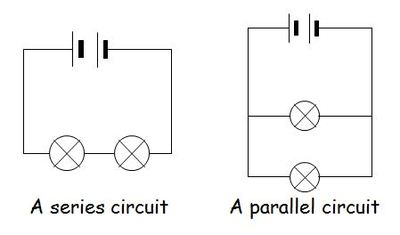
My program will include the correct symbols (in the UK) for each component:

|  |  |
| --- | --- |
| Component | Symbol |
| Cell |  |
| Resistor |  |
| Filament Lamp |  |
| Ammeter |  |
| Voltmeter |  |
| Diode (often seen without circle) |  |
| Light Dependant Resistor (LDR) |  |
| Thermistor |  |

Series and Parallel Circuits

A circuit can either be series (a single section/loop), or parallel (multiple sections/loops).

Here is an example of a series circuit (left), and parallel circuit (right).



I aim for my program to include both series and parallel circuits so that the user can build and understand more about the ways voltage, resistance, and other attributes act in both scenarios, as they often behave differently.

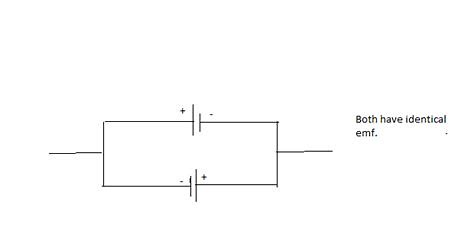
Electro Motive Force (EMF, measured in Volts/V)

My program will include Emf as an attribute for a cell/power source.

Emf is the total voltage applied to the circuit, this includes any power sources, weather it is a cell or power supply, and is defined as “the energy supplied per unit charge from the power supply”.

Emf can be calculated by summing all the Emfs in each power source to get a total voltage across the entire circuit. Emf is also directional depending which way round the power source is connected.

In the image below, if the Emf of the top cell was “10V”, and the Emf the bottom is also 10V, the total circuit Emf would be 20V as they would add together. If one cell was facing the opposite direction, the Emf would be taken as “-10V” and the total circuit Emf would be 10 + (-10) = 0V.



Including Emf in my program will be important because it is a key part of the OCR specification, and it allows me to have a total voltage value that is being inputted into the circuit to perform calculations on. It will also allow the calculations that are behind internal resistance (see later information below) to be made clearer to the user. Emf is one of the fundamental parts of a circuit as it determines how much voltage is supplied to the components, and due to this it will most likely be determined beforehand, rather than using other values to calculate it.

Current (measured in Amps/A)

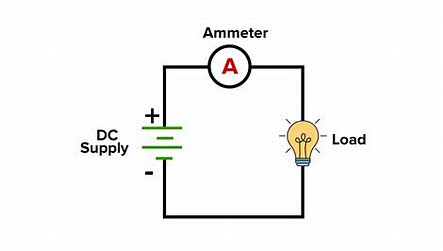
Current is defined as the rate of flow of flow of charge in a circuit (I.e. how much electrical energy is passing a particular point in a given time. This is related with the equation Q = It (Charge = Current\*Time) rearranged to give I = Q/t or “charge per second”.

To calculate current, the equation V = IR (Voltage = Current \* Resistance) can be used rearranged to I = V/R. The voltage is the Emf, and R is the total resistance of the circuit (see further down for resistance information).

Current is directly proportional to Emf, meaning that as the Emf of a circuit increases, so does the current. Furthermore, using the equation I = Anev (Current = Area of wire \* wire atomic density \* electron energy \* electron velocity), it can also be deduced that current is directly proportional to the velocity of the electrons when A, n, and e are constant. This means that a larger current causes the electron flow around the circuit to be faster.

In a series circuit, current is constant and is the same at all points, however in a parallel circuit current is split at each junction. This is due to Kirchoff’s 1st law which states that the total current before and after a junction does not change.

Current can be measured using an ammeter, which is a component that can be placed in series at a given point to display the current flowing through it. Ammeters have an extremely low resistance (see resistance section below for information about resistance) to prevent the current from being lowered too much by adding it in. Here is an example of an ammeter in series with a circuit:

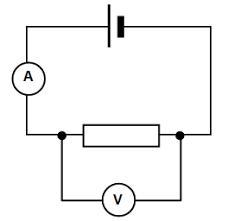


Including current in my program will be important as it is a fundamental part of physics circuits and is included in lots of calculations for both series and parallel circuits. It also gives the user a simple idea of how much energy the circuit/branch is being provided with as it is directly proportional.

Voltage/Potential Difference (measured in Volts/V)

Potential Difference is defined as the energy supplied per unit charge to the external circuit and is the voltage across a particular part of the circuit, or a specific component. It is “the difference” between the voltage before and after that circuit section.

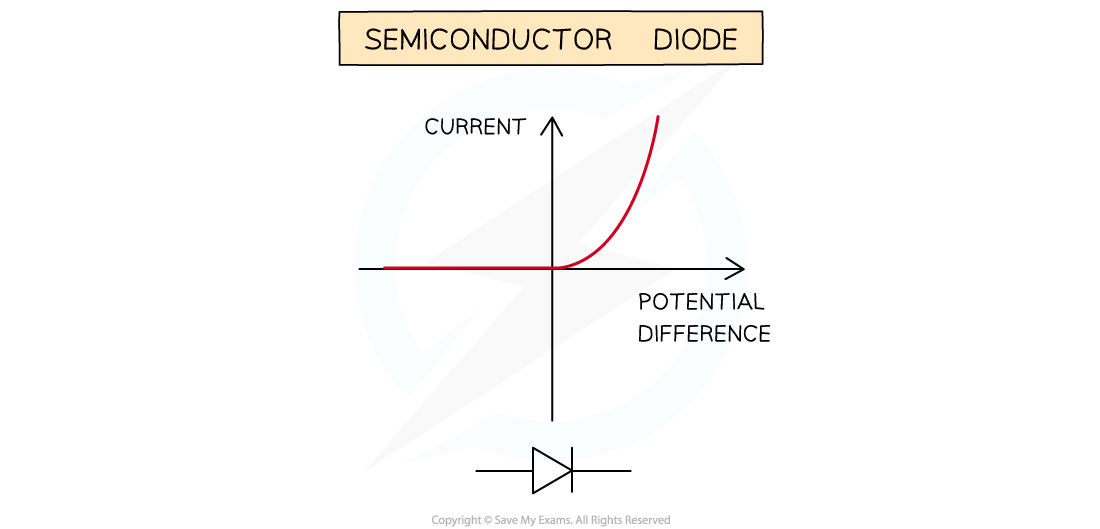
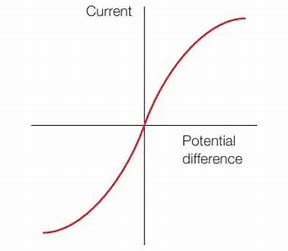
You can measure the potential difference of a component by putting a voltmeter across (in parallel with) it. This will display the voltage of that component. Here is an example of a resistor with voltmeter in parallel across it.

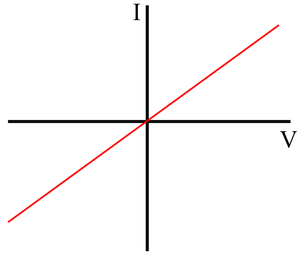


Voltage (potential difference) and current are directly proportional so that when voltage increases, current increases.

In a series circuit, voltage splits amongst each component, however in a parallel circuit, the voltage is equal in each branch (before splitting in each individual branch). This is due to Kirchoff’s 2nd law which states that in any given loop of a circuit, the total Emf must equal the sum of the potential differences across each component.

Every component has an I-V (Current-Voltage) characteristic, which can be seen by plotting a graph of current against voltage. Here are the graphs for each component in the OCR specification relevant to a circuit builder:

Fixed Resistor and LDR: Diode: Filament Lamp and Thermistor:



Including voltage/potential difference in my program is important because it, with current, is a fundamental part of the circuit which can be used to determine how different components interact under certain conditions (like in the graphs above). It also allows for calculations over individual components as the current the *component specific* potential difference can be used in equations themselves.

Resistance (measured in Ohms/Ω)

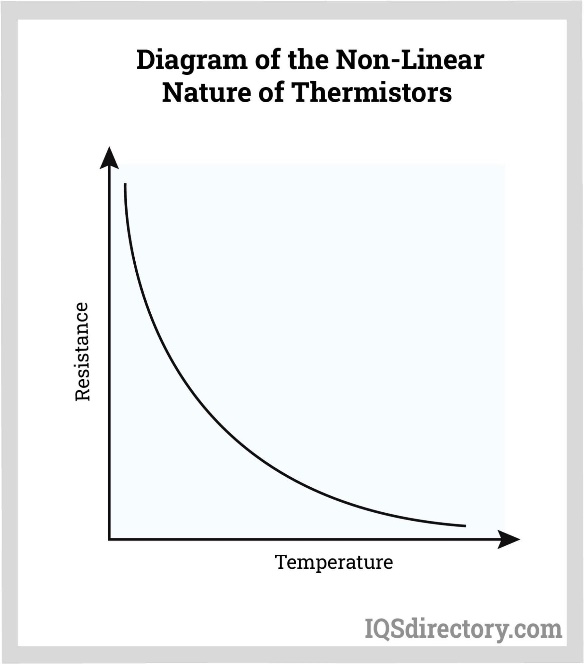
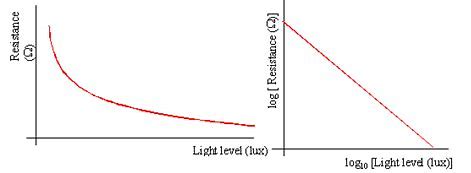
Resistance the natural occurrence for a component to “resist” or defy the movement of charge carrier (electrons) in a given circuit.

Every component of a circuit has a resistance, and this can change depending on the component.

* For a simple fixed resistor, the resistance is constant and does not change under external conditions. This can be seen on the fixed resistor’s I-V graph above with the simple linear line.
* For a diode, the resistance is extremely high in one direction (usually negative/backward) and after a given current, begins to rapidly decrease which increases current. This can be seen for the diode’s I-V graph above with the rapid increase in current. Below a certain voltage/potential difference, there is no current.
* For a filament lamp/bulb, the resistance changes depending on the voltage and current. As the voltage increases, the current increases which causes the filament in the bulb to heat up, which increases the resistance, meaning the increase in current is slowed.

For a thermistor and LDR, the resistance is inversely proportional to the temperature and light intensity respectively. Here are examples of resistance against temperature and resistance against light intensity a thermistor and LDR:

Thermistor (Resistance-Temperature): LDR (Resistance-Light Intensity):



Resistance can be calculated with the equation V = IR (Voltage = Current \* Resistance) rearranged to be R = V/I.

When voltage is split in a series circuit, it is split in an equal ratio to each component’s resistances, meaning if the Emf was 10V and the ratio of two components resistances was 2:3, one component would have a voltage of 4V, and the other would have 6V.

A diagram of a circuit

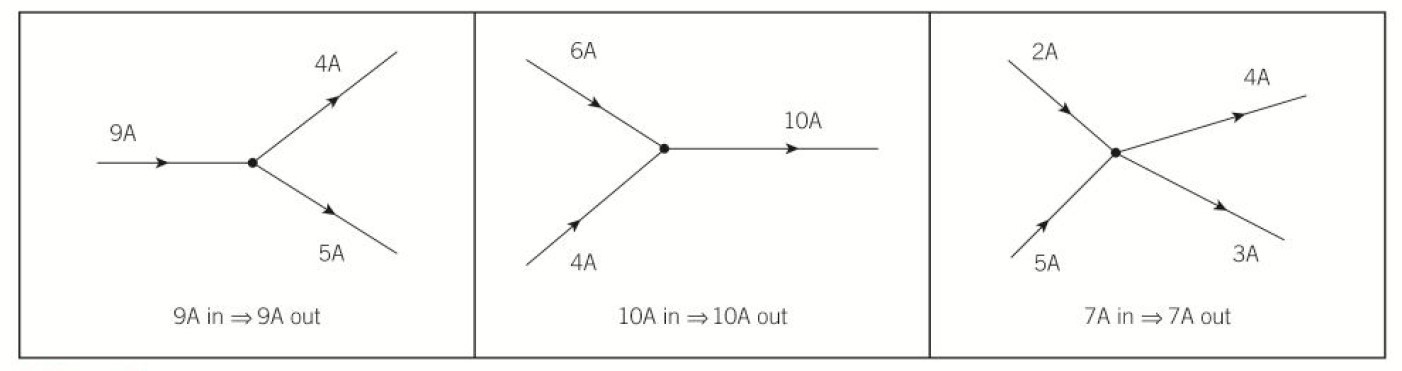
Description automatically generatedHere the ratio of resistance for R1:R2 is 1:4, meaning that 10V would be split into an equal ratio. (Kirchoff’s 2nd Law)

1+4 = 5 (total number of parts)

10/5 = 2V (voltage for 1 part)

R1 = 1part therefore 2V

R2 = 4parts therefore 2\*4 = 8V

When current is split in a parallel circuit at a junction, is it split down each wire based on the inverse ratio of the resistance of each section. If the current entering a junction was 10A and the resistance of two branches was 2:3, the first branch would have a current exiting the junction of 6A, and the second branch would be 4A. (Kirchoff’s 1st Law)

Here the current entering the junction is 9A.

The current exiting the junction in both wires is split into a ratio of 4:5. This means that the ratio for the resistance of each wire respectively is 5:4.

Calculating resistance from temperature and light intensity:

As seen by the resistance graphs of thermistors and LDRs, they both follow an exponential graph when increasing or decreasing the temperature or light intensity respectively. This means that as the change in temperature or light intensity increases, the change in resistance decreases.

To calculate the resistance the temperature or light intensity can be used in an exponential equation. Both relationships follow a base graph structure of y = e^(k\*Δ1/X) where:

* Y is the Resistance.
* e is Euler’s number.
* k is a constant based on the specific type of LDR or Thermistor.
* Δ1/X is the 1/change in temperature or light intensity.

This can be used to calculate resistance when temperature or light intensity are known, and voltage and current are not determined.

Including resistance in my program calculations is vital as it is, aside from Emf, the defining principle of all other circuit values. It allows for calculating current or a specific with the Emf and total resistance of all the components, for calculating the potential difference of each component, and calculating the current of separate branches in parallel (in addition to Kirchoff’s 1st Law). Due to resistance being a fundamental part of the physics calculations, my program will most likely use the resistances of the components as pre-determined values, as all the other values can be calculated from that.

Power (measured in Watts/W)

Power is the energy transferred in a given time from one form to another, for example electrical energy to heat. This is shown through the equation P = E/t (Power = Energy/time)

To calculate the power of a component in a circuit, you can use multiple equations:

* P = IV (Power = Current \* Voltage)
* P = I^2 \* R (Power = Current^2 \* Resistance)
* P = V^2 / R (Power = Voltage^2 / Resistance)

The more powerful a component is, the more energy will be transferred.

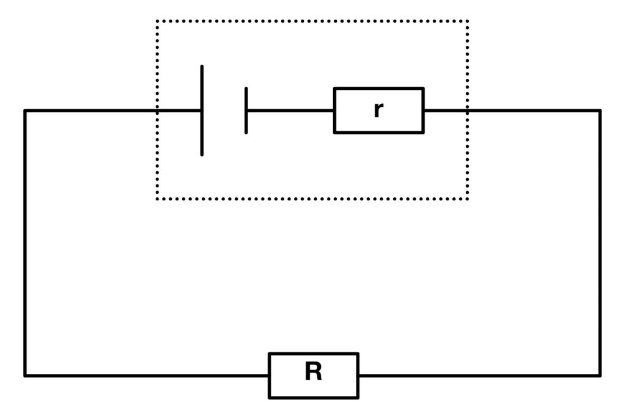
Including power in my circuit builder is important because it is a key part of the OCR specification, and it gives an easy representation of the rate of energy transfer for the circuit. Power can also be interpreted in multiple ways, for example the equation P = I^2\*R is used to calculate the power dissipated in a component (power transfer into less useful forms e.g. thermal energy).

Internal Resistance of a power source

Every power source (cell or power supply), has a resistance itself, known as the “internal resistance”, which is often (but not always) considered to be negligible.

When the power source has an internal resistance, it can be thought of as having an extra resistor in it. The extra resistor lowers the current in the circuit loop due to the total resistance being increased. This also means that in the given loop, the total emf supplied to the external circuit components is less as some of the volts are divided into the internal resistor, known as the “lost volts”.

Here is an example of a circuit with an internal resistance in a diagram:



The internal resistance, “r”, can be calculated using the formula ε = I(R + r), where:

* ε is the Emf
* R is the total resistance of the external components
* r is the internal resistance.

This can be expanded to give ε = V + Ir. (Where V is voltage across the rest of the circuit)

If the internal resistance is known, then it’s resistance must be included in the sum when calculating the total resistance of the circuit. The voltage across the internal resistor can then be found using V = IR and the “lost volts” can be subtracted from any future ratio divisions of the voltage across various components.

A power supply had an Emf of 10V and the internal resistor has a potential difference of 1V across it, 9V would be split into the rest of the components.

Including internal resistance in my program is important because it is one of the much harder concepts to grasp in the A-Level specification, and being able to use an interactive diagram to represent this internal resistance will be very helpful to improve students’ understanding as it will allow them to quickly and easily experiment themselves with how it affects the circuit. Including internal resistance also gives a better representation and idea of how physical/real circuits work, as all power sources naturally have an internal resistance which cannot be removed.

Calculations Overview

To begin the calculations in my program, I will need pre-determined values to have as a base attribute so that other properties can be calculated from them. These pre-determined values will likely be Emf and resistance, as they allow for the most calculations with the minimum known values.

For a particular loop or series circuit:

* Using the sum of the Emfs of each power source, the total Emf can be calculated.
* Using the resistances of each component, the total resistance can be calculated by summing them together, Rt = R1 + R2 … Rx
* Using the total Emf and total resistance, the current of the loop can be calculated using V = IR in the form I = V/R.
* Using the ratio of each component’s resistance, the Emf can be split into the same ratio as the resistances to calculate each component’s potential difference.
* With the potential difference or each component and the current of the section, the power of each component can be calculated using P = IV.
* If the power source has an internal resistance (which is pre-determined), it can be added to the total resistance of the circuit and subtract treat it as another resistor. This can then also be used to find the “lost volts” or potential difference across it.

For multiple sections or a parallel circuit:

* Each section can be treated as its own loop with an equal source of Emf to the “main” loop.
* The total resistance of the circuit can then be found by using the formula 1/Rt = 1/R1 + 1/R2 … 1/Rx. Where Rx is the total resistance of a particular section or loop.
* The total base/initial current of the whole circuit can be calculated using V = IR in the form I = V/R.
* Using the total current, the current of each section can be calculated using the inverse ratio of each branch’s resistances.
* When the current and resistance of each section is found, the calculations for the series circuit or particular loop can apply to find the rest of the properties for each component.

For an LDR and thermistor:

* The light intensity or temperature should be the pre-determined value and the resistance should be calculated based off the exponential equation stated previously above.

The user must be able to change the defining values of the circuit so that they can change the circuits properties. This means the user needs to be able to change both the Emf and internal resistance of the power sources, as well as the resistances of each component. For an LDR and thermistor, rather than changing the resistance, the light intensity or temperature respectively should be able to be changed, as this allows the user to see and understand how light intensity and temperature affect the resistance in an exponential way.

### Python Libraries and Modules

Pygame

Pygame allows you to:

* Create sprites, images, and shapes before blitting them to the interface.
* Use the keyboard and mouse as an input device by pressing buttons and moving the mouse pointer.
* Perform various external functions e.g. screenshotting.
* Use various events to detect changes to conditions.

Pygame may be useful in my program because it will allow me to represent the components and other images as sprites, potentially grouping them together with a sprite group. The keyboard and mouse input capabilities will also allow the user to be able to access and control the code without using shell, and events can be used for inputs to easily transfer an action done by the user to a function.

Here is an example of a window made in pygame (filled in blue with a red rectangle):

A red rectangle on a blue background

Description automatically generated

Tkinter

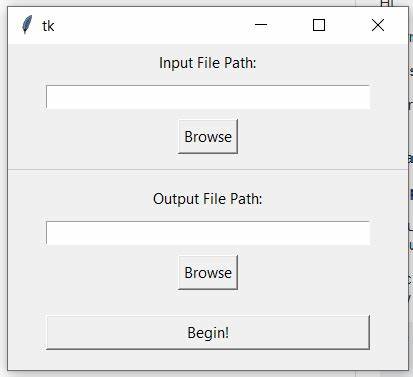
Tkinter allows you to:

* Create buttons and various input fields for example buttons, sliders, and text input boxes.
* Easily stack input fields in a grid form using the “.grid()” function.
* Use functions for each input field to easily transfer the user’s input into running a specific section of code.

Tkinter may be useful in my program because it means components can be easily created buttons and transferred input to pygame window by creating a new object with the button’s function. The “.grid()” function will allow me to easily slot the component buttons together and create multiple pages if there are too many components for the window size.

Tkinter may be useful in my program because it’s input fields and manipulation with buttons and text inputs is more advanced than pygame and other GUI libraries. This would be useful because components can be made with a button press or text input and when transferred to the pygame window to create a new component, likely as an object.

Here is an example of different Tkinter input fields (text, buttons, and text areas):



Threading

After looking into coding with multiple libraries with windows, I found that you cannot have multiple windows open at a time.

Threading works by putting each section of code into a procedure and assigning a thread to them. This will be useful in my code because it will allow multiple windows to run if needed, accepting inputs to be transferred between threads, for example in a global variable or class.

Here is an example of how threading can be used (with two functions, in this case add and multiply, being assigned a thread):

A screen shot of a computer code

Description automatically generated

Marshall and Pickle

Marshall and Pickle are both serialisation modules which can create a single string of data of characters which each represent what was saved. Using a serialisation module will be useful in my code because it can compile multiple items into a single string of data which can then be easily saved into and loaded from a file, which will be useful for saving data for classes and sprites.

Marshall and Pickle both use similar functions, however Marshall cannot save objects and restore classes (whilst pickle can), meaning it may not be suitable for my code where I may be saving objects and sprites for their positions, therefore I will most likely choose to use pickle.

Copy

Copy is a module which allows you to save the “state” of a variable into a new variable, so that if the original value used to assign changes, the new variable will not change.

Copy will be useful in my code because it will allow me to save certain attributes as other variables to use after the original ones have changed/been updated – which is often the case in the pygame event loop.

In copy you can use both a shallow copy and a deep copy, in which the shallow copy only copies base items whilst the deep copy copies all sub-items. The deep copy will be more useful in my program because I may need to copy classes and attributes within them.

## Objectives

1. The program should include most theories and aspects of A-level Physics OCR Module 4.1-4.3.
   1. The program should include all module 4 circuit components relevant to a circuit builder.
      1. Each component should be visually accurate and represented using the correct circuit symbols that are used in the OCR course.
      2. Components should be able to be connected in series and parallel.
   2. The program should include all circuit properties which are not calculated using a time period (which is not possible to show on a static diagram).
      1. The program should include Emf and internal resistance.
      2. The program should include Current.
      3. The program should include Voltage/Potential difference.
      4. The program should include Resistance.
      5. The program should include Power.
      6. The program should include light intensity and temperature to calculate the resistance of and LDR or Thermistor respectively.
   3. The program should follow the correct mathematical calculations that apply to real circuits.
      1. The program should follow common concepts and theories in physics circuits such as Kirchhoff’s and Ohm’s Laws.
      2. The program should follow all equations and relationships needed for calculations between different circuit properties.
      3. The physics calculations should be implemented as a procedure so that it can easily be called at any point in the code when it is needed.
      4. The physics calculations procedure should be called when the circuit is completed.
      5. The program should calculate each circuit property with a minimum number of pre-determined properties – Emf and internal resistance, resistance, and light intensity or temperature for LDRs and thermistors.
      6. The program should allow the user to change the pre-determined properties so that they can change the rest of the circuit values as a result.
      7. When the user changes a property/value, the physics calculations procedure should be called again to update the other properties of the circuit as a whole and for each component.
      8. If the circuit is broken, the program should reset all the properties of the circuit and components.
2. The program should allow the user to make circuits freely.
   1. The program should be able to detect when circuits are complete.
      1. The program should run an algorithm to traverse through the circuit and detect if there are any loops.
      2. The program should be able to find which components are in each section, or loop if the circuit is parallel.
   2. The program should allow the user to create their own components.
      1. The program should have drag and drop capability to move the components around.
      2. Components should not use a grid snap when placing down for smooth movement.
      3. Components should be able to be placed anywhere in the circuit building section (pygame window).
      4. Components should change image when being hovered over to notify the user the mouse will pick up that component when clicked.
      5. Components should have a unique ID to identify them separately from other components for data processing.
      6. The user should be able to see what properties a circuit component has for example voltage and power etc should be able to be displayed.
      7. Components should be represented in python as a pygame sprite.
      8. Components should be implemented as a class.
      9. Components should be able to be deleted by the user.
      10. The filament lamp component should turn on and change image when the circuit is complete.
      11. The ammeter should be able to be connected in series.
      12. The voltmeter should be able to be connected in parallel across another component and not in series.
   3. The user should be able to create wires and connect the components together to complete the circuit.
      1. Users should be able to drag wires from either side of a component.
      2. The program should use a shortest distance subroutine/function to determine the closest connector to the mouse, if it is within a certain range.
      3. The closest connector found to drag from should indicate that it is selected the closest for the user to drag from it.
      4. While dragging, the wire should appear between the start component and the mouse pointer to indicate that the user is still making a wire.
      5. While dragging the program should find the closest available connector within a range.
      6. The closest connector shown when dragging should allow the wire to be connected to it by the user.
      7. Once a connection is made on a side of a component, the user should not be able to drag or connect any more wires to that specific side.
      8. Users should still be able to drag and drop components when wires are connected, and the wires should update to the new ends of the component accordingly.
   4. The user should be able to capture an image of their circuit.
      1. The image should be saved into a screenshot folder.
      2. There should be a simple button to screenshot the circuit.
      3. The screenshot should only include the circuit build area and not the components box or any another buttons.
      4. The user should be notified that a screenshot has been successful or not and that it has been saved to the desktop or specific folder.
3. The program should use a graph to store the circuit.
   1. Each component should represent a node.
   2. Each wire should represent an edge.
   3. The graph should be implemented with an adjacency list as there are more likely to be more nodes than edges.
      1. Each component and wire should be stored in the graph using their IDs for each node and edge.
   4. The graph should be unidirectional with no specific direction as this will be decided by the Emf calculated when the circuit is complete.
   5. The graph should be unweighted as the effect of the length of wire on resistance is considered to be negligible.
   6. The graph should be implemented with a class so that it’s attributes and functions can be easily accessed and called by the rest of the program.
      1. The program should be able to add and remove both nodes and edges, which should be a callable function for whenever a change to the circuit occurs.
   7. The graph should have a cycle detection algorithm using a depth first search (DFS) to detect if the circuit is cyclic and therefore complete.
      1. When complete, the circuit should change a condition or variable to allow the other conditions to run.
4. The program should also have a simple, easy to use user interface (UI) to allow for newer users or those who have less technological experience.
   1. The UI should have clear, toggleable labels that show the user what to do.
   2. The program should be entirely accessed through the pygame window and not use the shell to run and process any information and inputs.
   3. The UI should include a reset button to clear the circuit.
      1. When the circuit is reset all components and wires should be deleted, and all other variables or properties should be reset or updated when the user begins to make a new circuit.
   4. The program should have an undo and redo button to undo and redo the user’s last actions respectively.
      1. The undo and redo button should use stacks to keep track of the users last actions.
      2. When the user performs an action, the last action should be added to the undo stack.
      3. When the user clicks the undo button, the last action should be popped off the undo stack and added to the redo stack.
      4. When the user clicks the redo button, the last undone item should be popped off the redo stack and added into the undo stack again.
      5. The undo and redo functions should be implemented as a class so that they can be easily referenced for adding and removing items.
      6. The undo button should be able to store up to 5 of the user’s last actions.
      7. When a new action is performed, the item at the bottom of the undo stack should be popped off and the new action should be added to the top.
      8. The redo stack should have a maximum size to allow for only 5 actions to be stored and undone.
      9. The user should be able to undo and redo creating a component.
      10. The user should be able to undo and redo moving a component.
      11. The user should be able to undo and redo deleting a component.
      12. The user should be able to undo and redo creating a wire.
      13. When the undo button is pressed, if the last action involves a component the last attributes of a component should be saved to a file, using a file serialisation module (pickle).
      14. When the redo button is pressed, if the last action involves a component the previously undone attributes should be loaded from the file and re-applied to the correct component.
      15. A pointer should be used to keep track of which file the program is currently on and which one to call if the user performs an undo or redo.
   5. The UI should include a section for selecting a unit graph to show.
      1. The graph will plot the relationship for different units selected and how they plot for each type of component.
      2. The user should be able to choose what units to show on the graph.
      3. The graph should include both linear relationships and curved relationship lines.
         1. The graph should include the I-V characteristic graphs for each component.
         2. The graph should include the relationship between resistance and light intensity for an LDR.
         3. The graph should include the relationship between resistance and temperature for a thermistor.
      4. The graph should not show a relationship for something both values on each axis are independent of each other, for example the light intensity on a fixed resistor.
   6. The UI should have a box on the side of the screen (using the tkinter module).
      1. The box should include all components as buttons to click on. This should create the components in the pygame window to be used.
      2. The box should include the circuit reset button.
      3. The box should include the unit graph type selection as buttons.
      4. The box should have a search bar or scroll bar if it is not large enough to store all the components needed.
5. The program should be able to show the electrons flowing around the circuit.
   1. The user should be able to toggle the electrons being visible.
   2. The electrons should move the correct way around the circuit (negative to positive), which depends on whether the calculated total Emf is positive or negative.
   3. The speed of the flowing electrons should depend on the current of the circuit.
      1. The electrons should have a maximum speed to prevent them from being too fast to interpret. This means that beyond a certain current value the electrons should not speed up.
   4. The electrons should move along the wires and through each component.
      1. The program should create vectors along each wire and through each components in a loop.
      2. The vectors should be implemented as a class so that they can include their own attributes and variables for being accessed.
      3. The electrons should follow the path of each vector, increasing/decreasing their x and y positions in proportion to the gradient of the vector.
      4. The electrons should traverse through the components along each vector in the correct order around the graph/circuit.
   5. The electrons should be created as sprites and implemented as a class to allow their positions and other attributes to be changed easily.
   6. The program should determine the number of electrons in the circuit based on its total size/total vector magnitude.
   7. The electrons should be spaced out equally through the circuit as close to 50px apart as possible.
   8. If the user moves a component, the vectors should be updated to follow the new path of the wire.
      1. While a component is being moved electrons should not be visible as they would not follow the correct path of the wire.
6. In addition to components and electrons, the program should represent any other non-static elements being shown on the screen as a class.
   1. The program should use sprite groups to link different objects (sprites) together.
      1. There should be a sprite group to access all sprites that can be shown on the screen.
      2. Components and electrons should both have their own respective sprite groups to allow quick access when moving/updating components or moving electrons along the circuit’s vectors.
      3. Each type of component should have the same base attributes and variable names so that they can be easily referenced. Component type specific attributes should be created based on certain conditions.

## Prototyping

### User Interface Initial Ideas

**Here is my first prototype of the UI:**

There are two separate windows, one for components (tkinter) and the other for building the circuit (pygame). Initially, I put the tkinter window to the right of the pygame window.

There are also two places that the attributes and unit graph can go – either in the tkinter window underneath the components or in the pygame window to the side of the build area.

A screenshot of a computer

Description automatically generated

**Here is my second prototype of the UI:**

The tkinter and pygame window have been switched around so the tkinter window is now to the left of the pygame window. This made more logical sense as although the user will be interacting with the pygame window more, they will be creating the components on the left and then moving to the right to drag them around.

The attributes and graph display section is now in pygame rather than the tkinter as this will be allow there to be more room for the components and various input fields in the tkinter window (e.g. reset button and graph selection).

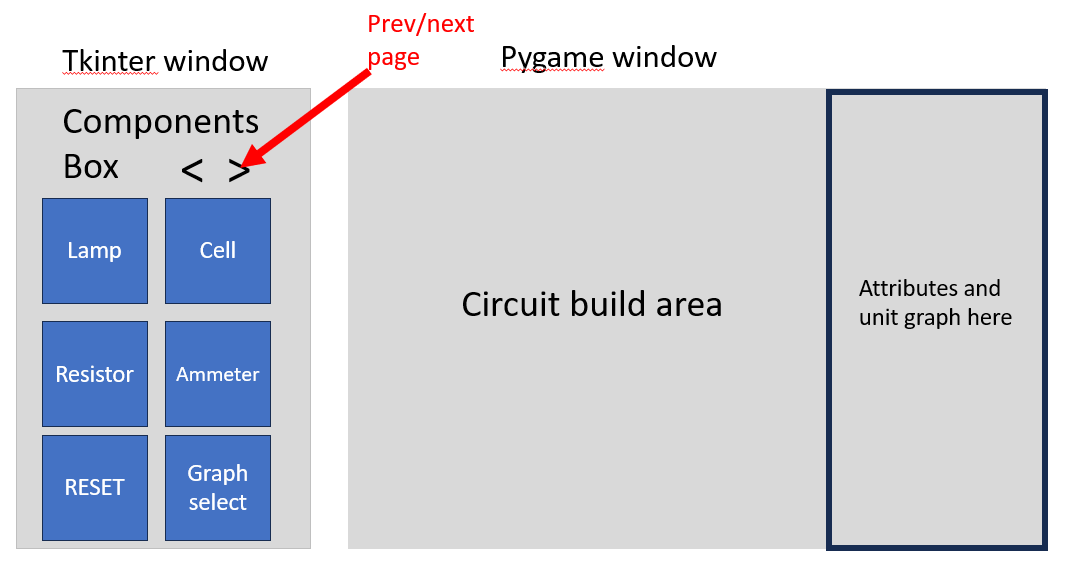
A screenshot of a computer

Description automatically generated

**Here is my final prototype of the UI:**

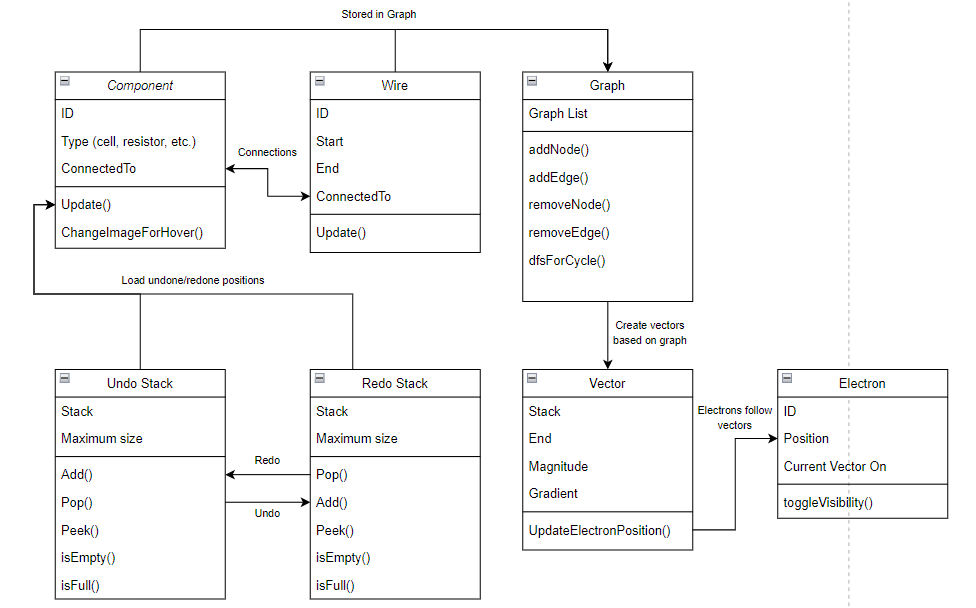
The attributes and unit graph have been moved over to the opposite side of the pygame window, as this means the distance between the tkinter window (component creation) and the circuit build area is slightly smaller. The box has also been extended further to take the whole height of the screen, as this allows for showing many attributes and leaves more room for the graph.

I have also made sure to include the ability to change pages in the components box as this may be needed if there are too many components and/or input fields that need to be in the box.



### Initial Class Diagram

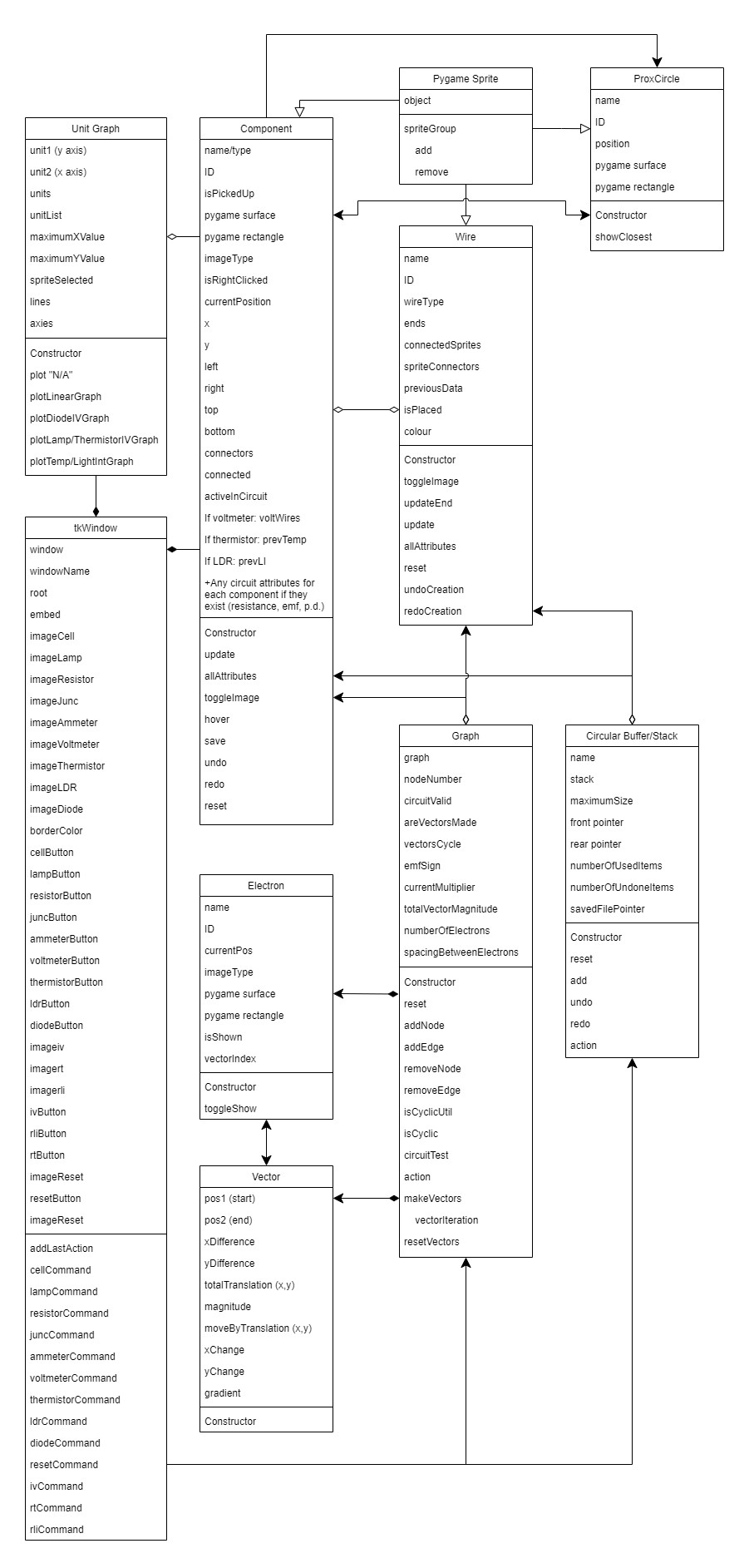
**Here is my prototype for the class structure of my program using a class diagram:**

****

# Design

## Class Diagram

Here is a class diagram showing all attributes and methods for each class and how they are related:



The Component, Wire, and ProxCircle classes all inherit from the Pygame Sprite class and are organised into sprite groups. Sprite groups are also organised into a list “allGroups” which can be iterated through.

Sprite groups created are:

* allSprites – for every sprite in the program.
* componentSprites – for every component in the program.
* IDSprites – for every recurring sprite in the program (can have multiple at once).
* IDTracker – for every sprite with an ID (used to assign the next ID whenever a new sprite is created.
* electronSprites – for every electron in the program.

The Component and Wire classes are connected through their “connections” in the circuit, for example if “component1” was connected to “wire5”. This link is important as it means both the components and wires can be updated using their connections to other components.

The Circular Buffer/Stack class (cStack in the actual program) is connected to both the Component and Wire classes as when the user undoes or redoes actions, the components and wires are updated to be their previous or next positions. This link is important as it means component and wires can be easily updated when the user wants to backtrack or retrack their past actions.

The Graph class is connected to both the Component and Wire classes as when a wire/connection is made between two components, a graph data structure can be made storing the components as nodes and adding an edge between them for each wire. This link is important as it allows nodes and edges in the graph to be easily added.

The Circular Buffer and Graph classes are connected as when actions are undone/redone, if the circuit changes when wires are removed, the edge stored in the graph between the two nodes (components) can be removed easily before testing for a complete circuit again. This link is important as it allows nodes and edges in the graph to be easily removed or re-added.

The Graph and Vector classes are connected because when the circuit is deemed complete with the cyclic test, vectors are created in the makeVectors method. The vectors are then stored in the graph “vectorsCycle” attribute. The Graph class is also connected to the Electron class as when the vectors for the circuit are made, the electrons are created and assigned start positions. These links are important because it allows the Graph class to directly communicate with the Electron and Vector classes so that both electrons and vectors can be made in a single method (makeVectors).

The Electron and Vector classes are connected as when electrons are moving, they store a vector index to point to a particular vector and use the moveBy translation in the form (x,y) to update their position. This link is important because the electrons can directly access the vector’s moveBy attribute for their position to be updated.

The ProxCircle class uses the Component class to update its position to the closest component’s connector (either left or right). This link is important because it allows the user to know where to drag and connect wires to and from.

The UnitGraph class uses the Component class the determine which graph to plot for the selected component, which is stored as the “sprite” attribute (shown as spriteSelected in the diagram).

## Event loop flowchart

Some processes mentioned in the flowchart diagram and description are mentioned in more detail below.

Here is an abstracted flowchart showing the main functionalities of the Pygame event loop (more on pygame below):

A diagram of a flowchart

Description automatically generated

Here is a verbal explanation breakdown of the flowchart:

1. Event loop cycle starts.
2. The screen is filled with colour hex code ‘#999999’
3. The existing pygame events are iterated through to find if an event if being pressed, which if so the appropriate processes and functions are performed.
4. If left click is pressed, the program checks if the mouse is touching a component which if so it picks up the component.
5. If a component has not just been picked up, if another component is being moved already it is checked to be a voltmeter in which if so, two wires are created when it is in range of another valid component.
6. If right click is pressed, the program uses first checks if a wire is currently being dragged and, if not, if the mouse pointer is touching the proxCircle, a wire is created and begins being dragged by the mouse. If the mouse is not touching the proxCircle, a variable to check if right click was being pressed in the previous iteration of the event loop. If it was and the mouse is touching a component, any red components are set to be black and the component touched is set to red (selected).
7. If none of the mouse buttons are pressed, variables used when dragging a wire and picking up a component are reset, and any voltmeter wires that have been made when dragging are created and placed as objects.
8. The graph and stats box lines are then drawn.
9. If the user has pressed 1 to show the labels, the labels are drawn/shown.
10. If the labels are not shown, if the user have pressed 3 to show the controls, the controls are drawn/shown.
11. If controls are not shown, the unit graph axes are drawn, and if a graph and a component are selected, the axes units and relationship line are drawn respectively.
12. If the circuit is complete and vectors have not been created, the vectors for each wire and component are created along with the electrons which are assigned initial positions.
13. The program finds the nearest connector to the mouse, which is “None” if not within 75px.
14. If the mouse is touching a component, change the component image to show it is being hovered over.
15. Show components, then the proxCircle if applicable, and then any wires between a component and either another component or the mouse pointer.
16. If the circuit is complete, update each electron position to by the amount specified by its current vector and other factors using a calculation.
17. If the electron has passed the end of its current vector, set its position to the end of the vector and increase/decrease its vector index to be following the next vector in the circuit.
18. If a screenshot has been taken, save the screenshot and show the user a message to notify that a screenshot has been taken for the next 400 event loops (approx. 1-1.5s)
19. End of the event loop so return to the start of the flowchart.

## Packages, Libraries, and Modules

### Tkinter

Tkinter is used for my “components box”, which includes the ability to create components, select the relationship graph and reset the circuit.

I used Tkinter to create buttons with an image of set dimensions (80x80px) and used the in-built grid geometry manager to arrange each button into rows and columns. I added padding on each side to help distinguish between two buttons and on the top and bottom in order to separate the component, graph, and reset buttons.

When buttons are pressed the program accepts the input as a function for each button, and (for a component button) a new Component object is created. This component is set to be a sprite and gets added to multiple sprite groups which can then be used and accessed through Pygame. For a graph button, the specific function for that graph is called and a new UnitGraph object is created, which can be initialised and used in future processes. For the reset button, the reset command is called, and all components, sprites, and objects are deleted and the pointers for the undo-redo stack are reset.

### Pygame

Pygame is used for my main circuit build area, as well as displaying component attributes and the unit relationship graph on the right.

I used Pygame to create surfaces and rectangles for each image, text, or shape being shown, and used the in-built sprite object management to organise the components into sprite groups and allow the rest of the program to perform actions on them, whilst pre-defining any constant position texts or shapes.

I also took advantage of Pygame’s ability to take inputs from the user using the keyboard directly, which was useful in allowing the program to be controlled entirely using the GUI interface (without having to look and input into the shell). This also allowed me to make easy and quick controls, since the user can simply click a keyboard button rather than navigating to press buttons using the mouse. It also allowed me to accept inputs using the mouse and get the position of the mouse when needed.

### Threading

Threading is used for allowing my program to run and accept inputs from both Tkinter and Pygame windows at the same time.

I used Threading to create two threads, one for Tkinter and one for Pygame, and assigned each thread to a subprogram containing all the code for each one. These threads are called upon the start.

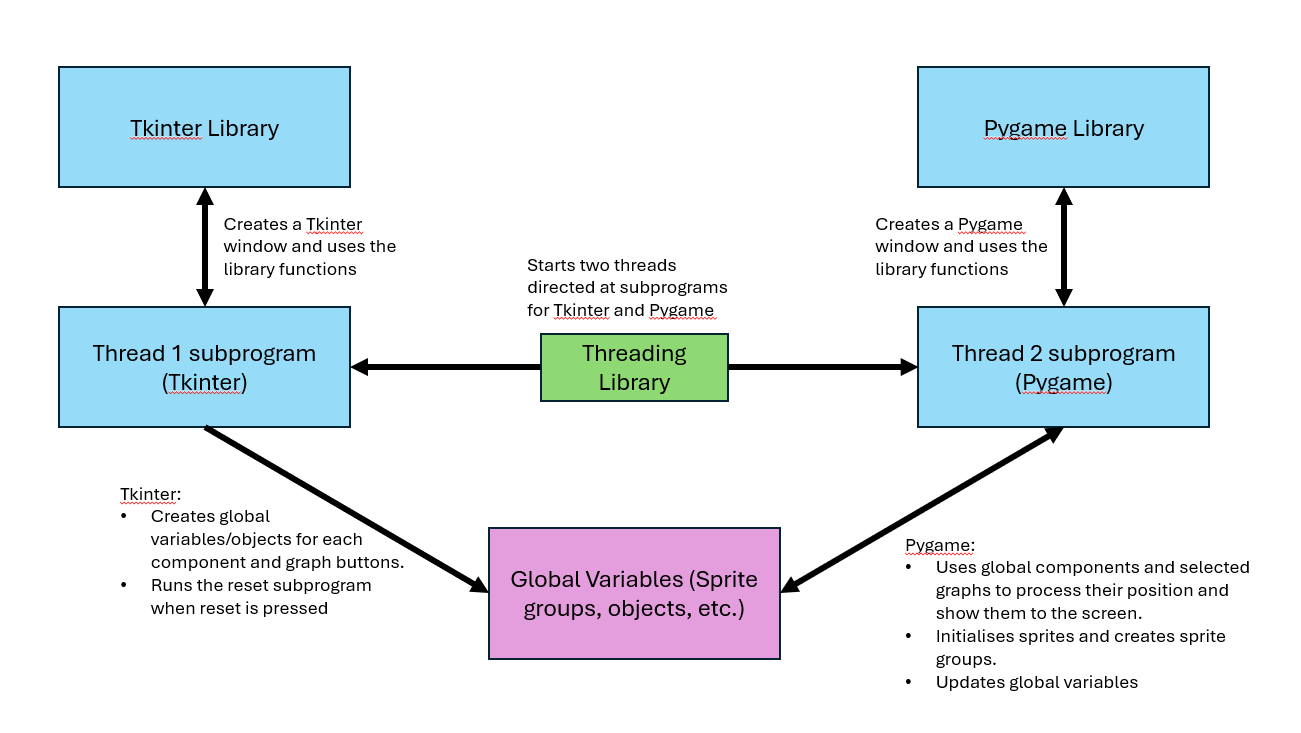
This library was very useful in my code because it allowed me to use the strengths of both Tkinter (for simple buttons) and Pygame (for images and circuit building). This was necessary for my code to run, function, and flow well.

### OS (allows use of the operating system)

After realising the need to make a non-machine specific program, OS is used for allowing my program to access various components from the operating system of the computer.

This library was useful in my code because it allowed me to position the Tkinter and Pygame windows at set points on the screen as, depending on the screen’s size, the windows may not normally fit in certain other positions that they may appear in.

Here is a diagram of how Threading allows Tkinter and Pygame to run together:



### Pickle

Pickle is used serialising objects to be saved and recalled from files.

This was useful in my program because it allowed me to easily create a single string for component objects and save the serialised string into a specific files with each for each action, which are loaded when the undo and redo buttons are pressed with the pointer indicating to each specific file.

### Math

Math is used for the physics calculations between resistance and temperature, and resistance and light intensity.

This library was very useful in my code because it allowed me to use “math.exp(1)” to use Euler’s number in the physics equation. This ultimately allowed me to represent the calculations and relationship between these circuit values in a correct and accurate way.

## Files, Data Structures, and Algorithms

### Circular Buffer/Stack

I used a Circular Buffer (implemented as a stack for FILO) to track each action for the programs undo-redo system. It is represented similarly to a circular queue, in that when there is space, the stack wraps around the front from the end, allowing a quick insertion/removal of the data into the stack.

It is initiated with a constant maximum size (10), which I decided to change from what I initially said (5), as the user can easily perform 5 actions quickly, meaning they may not be able to undo as much as they need to. The new maximum size of 10 allows the stack to store more items so the user can decide to undo to a further point in time.

Actions that can be undone and redone are:

* Creating a component
* Moving a component
* Deleting a component
* Creating a wire

It is designed such that it has a front and rear pointer to allow the program to keep track of where to insert and “remove” from, whilst tracking the number of items in the stack array which is currently being used. A file pointer is also created which can be universally accessed by each component as they saved and load previous data (more on saving to files for undo-redo below).

Wires can be undone and redone in addition to the components. Undoing a wire will remove the sprite from any sprite groups iterate through the components which are connected to it and set them to no longer store a connection. The particular edge is removed from the graph, and the wire’s connections are reset after saving them to a variable using the copy module. Redoing a wire creates the edge on the graph again and re-instates the connections using the previously saved data, which is iterated through to re-instate the connections in each relevant component.

Items are added when the user performs actions (which include creating a component, moving a component, deleting a component, and creating a wire). When added, they are given as an array containing the type of action e.g. “movedComponent”, and the sprite object the action was performed on. When the item is being accessed, the type of action is identified, and the relevant method is called for the particular sprite involved.

Actions are undone when the user clicks the backspace key, which lowers the number of items being currently used and moves the rear pointer backward by one place. This means that the item in the previous position is not actually removed (meaning it can be redone later), but new items can replace it if more actions are done.

Actions are redone when the user clicks the tab key, which increases the number of items being currently being used, moving the rear pointer forward.

Here is a diagram explaining the format of how actions are stored in the stack and what happens when items are added (note that all diagrams of the stack are represented with a maximum of 8 actions to easily represent them in a circle, however my program actually allows 10 actions to be stored at once):

A blue circle with black text

Description automatically generated

Here is a diagram explaining how the pointers and actions are taken/updated when the undo and redo buttons are pressed:

A blue circle with text and a blue circle with text

Description automatically generated

Here is a diagram explaining how the pointers and actions are updated when a new action is performed while the stack is full:

A blue circle with arrows pointing to the front

Description automatically generated

### Graph

I used a graph to represent the circuit as, using a traversal algorithm, a cycle can be detected therefore showing that it is complete allowing the program to start calculating the physics of the circuit.

The graph is implemented using an adjacency list, rather than an adjacency matrix as there are likely to be more nodes than edges at any given time, meaning using an adjacency list is much more memory efficient. It also allows for the graph traversal algorithm to easily see all the nodes which any given vertex is connected to, since the graph is undirected and unweighted.

The graph represents the components and wires as each node and edge respectively, storing their object IDs to uniquely identify them and make the graph algorithms and/or data processing easy to work with.

When creating or removing nodes or edges, the circuit runs the circuit test algorithm, as the conditions may have changed. When testing, the algorithm first checks that at least one of the components in the circuit is a cell (as there must be a power source). When a cell is found, the function then tests if there is a cycle afterwards as it requires more processing. After testing for the circle, if the circuit is valid (with at least 1 cell and a cycle), “circuitValid” is set to true which can then be used to check in the event loop and run new parts of code (e.g. physics calculations).

When the graph is complete, a function is recursively called which makes and adds vectors into an array which follow the path of the wires in the correct order around the loop. (more on vectors below).

Here is an example of a circuit made in my program, showing how each component is stored as a graph:

A diagram of a diagram of a diagram

Description automatically generated with medium confidence

### Graph cycle detection algorithm (using DFS)

During the circuitTest function, the “isCyclic” function is called which returns True or False depending on if a cycle has been detected (making the graph cyclic).

The algorithm follows a DFS (depth first search), creating a “visited” dictionary with a key for each node, storing True or False for if it has been visited yet. The graph is iterated through for each node, which calls a recursive subprogram checking through each other node that it is connected to, to determine if one of the other nodes is already visited, provided it is not the parent node (the node used in the previous function call). The function is recursively called until either all the nodes in the visited dictionary have been called, or one of the nodes is found to be visited and is not the parent node.

If a node is already visited and is not the parent node, then the graph contains a cycle “isCyclic” returns True.

### Vectors

I used vectors to allow the electrons flowing around the circuit to easily follow a particular path along each wire.

When the graph is complete, vectors are created using a recursive function, and put into an array alternating between a vector along a wire and a vector from one side of a component to the other.

Each vector is initialised with a start and end position, a magnitude, gradient, and relative “translation vector” in the form (x, y) for the electron’s position to be updated by that amount each time when moving.

A diagram of a circuit

Description automatically generated

### Undo/Redoing moved components using files

When a component is moved, it’s data and positions get stored as soon as it gets picked up so that when the user clicks the undo button, the previous data for the component can replace the current data.

Each components saved data is stored in a particular file, which is determined by a pointer stored in the circular buffer/stack (separate to the rear pointer as not all actions involve moving a component).

Here is an image below showing the 10 files used to store component data, the number at the end indicating the file pointer number when the data was saved. (There are 10 files for 10 movements, as although not all files will always be filled with relevant data, the user may perform 10 movements in a row which needs a capacity of 10 files).

A screenshot of a computer

Description automatically generated

When a movement of a component is done, the “save()” method is called which takes all the attributes of the component object (as a dictionary) and saves them each as separate values in a list. The file with the particular pointer number is opened (overriding the old data in the file) and serialising using the Pickle library, allowing a single string to be written to the file. Any attributes using of containing a Pygame surface are not saved as they cannot be serialised. The file pointer is then incremented.

When a movement is undone or redone, the file pointer is incremented or decremented respectively. The corresponding file is then loaded in read mode, and the string is un-serialized. The list is then iterated through, setting each existing component attribute to the loaded corresponding value. Surfaces (which were not saved) are re-assigned if necessary.

Here is a diagram showing how saving and loading from files is processed for moving components:

A screenshot of a computer

Description automatically generated

Here is an abstracted diagram showing the processes of saving and loading to or from and file after moving a component:

A diagram of a software flow

Description automatically generated

### Screenshots

Screenshots can be taken of the Pygame window which are saved to a screenshots folder. The folder also contains a text file which holds the number of screenshots that have been taken. This is used to give the screenshot a unique name, e.g. screenshot1, screenshot2, … screenshotX.

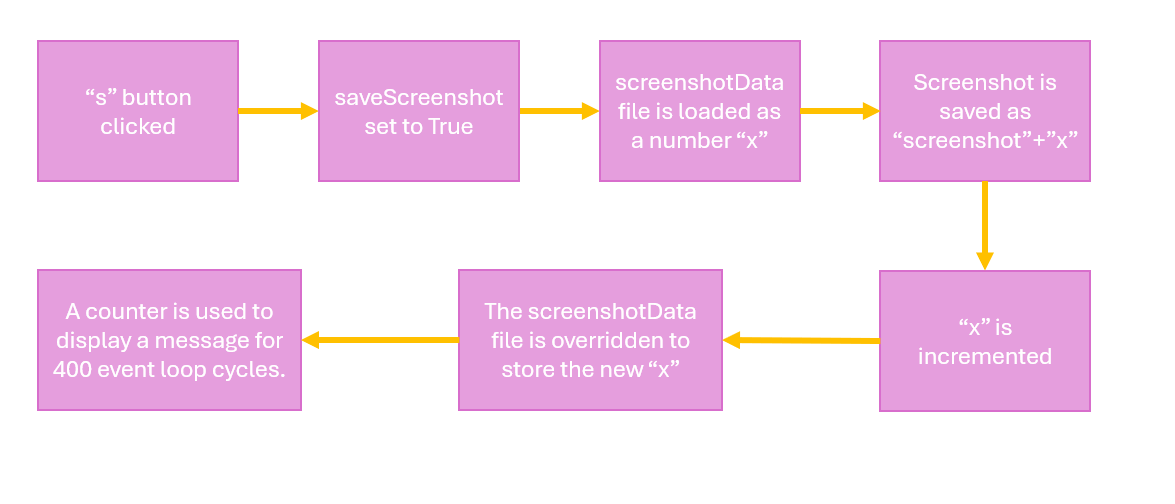
When the “s” key is pressed, a variable is set to true which is tested in a condition below in the event loop. This saves a screenshot to the folder in the form “screenshot+number”, and a message is displayed for 400 cycles of the event loop (~1.3s) before disappearing.

Here is an example of the screenshots folder with 4 saved screenshots:

A screenshot of a computer

Description automatically generated

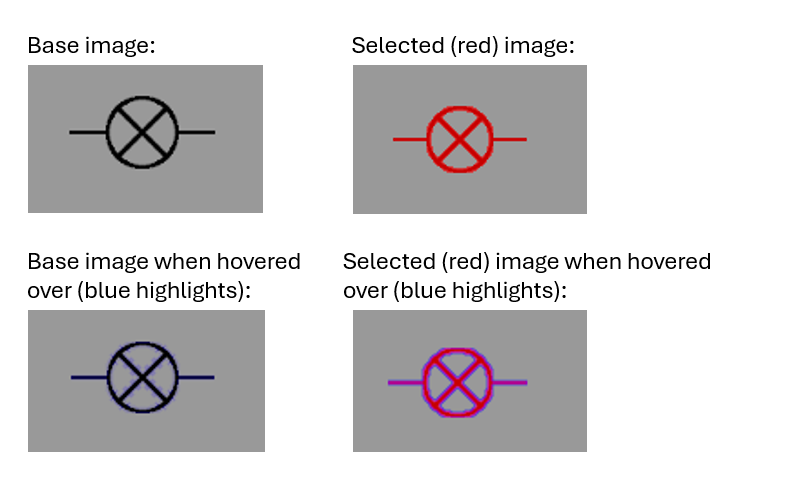
Here is an abstracted diagram for the process involved in saving screenshots:



### Loading images

Any images used in the program are loaded from the graphics folder. Each component (aside from the cell) has a preset width of 80px, and each Tkinter button image has a set size of 80px\*80px.

All components have 4 image variants, except for the lamp and cell which have extra images for being both turned on and with a resistance respectively:



### Finding nearest available connectors

When connecting wires between the components, the program runs an algorithm to find the nearest available (non-connected) component connector, e.g. left of right. When found, if the mouse is within 75px of the closest point, a red circle “proxCircle” is shown so that the user can either drag a wire from it or let go of a dragged wire for it to snap on.

Here is an example of the red indicator circle that shows:

A diagram of a wire

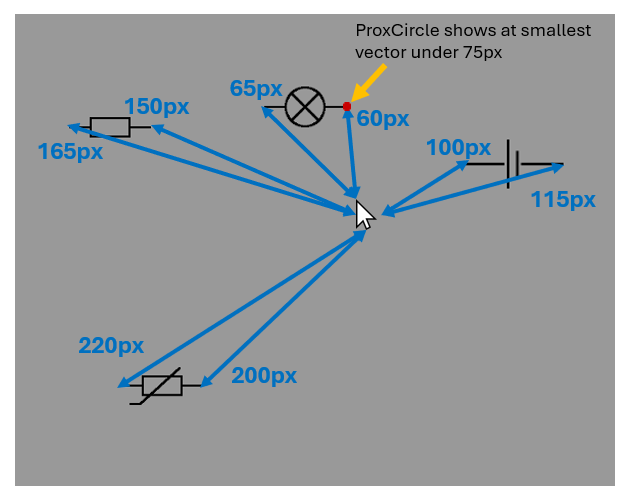
Description automatically generated

To calculate the nearest point, the “findNearestConnector” function is called. The algorithm iterates through each component, checking that the component is not a voltmeter (which uses a different connecting system) and that is it not where the wire is being dragged from already (as you can’t connect a wire to the same component it started from).

The distance between each of the sprite’s connectors is found by taking each connector’s coordinates and finding the magnitude of the vector between it and the mouse using Pythagoras’ a2 + b2 = c2 in an embedded “getMagnitude” function. If the magnitude is greater than 75(px), the function returns None.

Each magnitude in range is stored in a 2D list with each element in the form [spriteObject, connectorType, magnitude]. The list is then iterated through and the list element containing the lowest magnitude is returned.

Here is a diagram showing how each distance for each connector is taken from the mouse (hyperbolic):



Here is an abstracted diagram of the findNearestConnector function any other processes involved:

A diagram of a diagram

Description automatically generated

### Physics Calculations

When complete, the circuit runs the “physicsCalc” algorithm to calculate all the circuit attributes, e.g. the current and the voltage across each component. To calculate all the values, some must be pre-determined in order for the correct equations to be used.

Pre-determined values are:

* The emf of each cell
* The internal resistance of each cell
* The temperature of each thermistor
* The light intensity of each LDR
* The resistance of any other components

The subprogram/algorithm runs by:

1. Checking if the circuit is active (i.e. the circuitTest from the graph returned True).
2. The current and emf are initially set to 0 and each component is iterated through using the “activeSprites” which contains all spriteObjects which are in the graph.
3. If the component is a cell, the emf (positive or negative) is added to the current emf to give a total at the end.
4. If the component is a thermistor or LDR, the of either is calculated using R = R0\*ek\*Δ(1/x), where R0 is the initial resistance, k is a constant co-efficient to scale the exponential increase/decrease (3000 for a thermistor and 5000 for an LDR), and Δ1/x is the change in 1/temperature or 1/light intensity, e.g. “(1/x) – (1/x0)”.
5. The emf sign is determined (positive, negative, or 0) and stored in the graph, before the emf value is taken to be the absolute value for further calculations. The emf sign is used to determine the direction of electron flow later on.
6. ActiveSprites is iterated through to calculate the total resistance of the circuit, which is then used to calculate the current using emf/totalResistance.
7. Each component’s current is set to the circuit’s current.
8. If any cells have an internal resistance, the “lost volts” is calculated by considering the internal resistor as another resistor, and each other component is assigned a potential difference (voltage) value depending on their resistances (for an equal ratio split).
9. Power is calculated if applicable using the current multiplied by the potential difference/emf.
10. If the component is a diode, its voltage is checked against its minimum voltage, and if it is not above the minimum value, the current in the circuit is set to 0 and other attributes are not calculated.
11. A current multiplier is set for setting the speed of the moving electrons (more about electron movement below).
12. If the component is a voltmeter, it is checked if it is connected to any other components in which its potential difference is set to be equal.

Here is an abstracted diagram of the run-through of the procedure:

A diagram of a diagram

Description automatically generated

When a component is selected, the user can change certain values for certain components, which are outlined using a red box, as described in the UI descriptions above. The values can be incremented or decremented using the up and down arrow keys. When pressed, the custom value for that selected component will increase/decrease by the “increment step” amount, in which the physicsCalc subprogram will be run again to re-calculate all the values.

To change the increment step, the left and right arrow keys can be pressed, which changes the step by a factor of 10, between 1, 0.1, and 0.01 for the majority of components and 10, 1, and 0.1 for light intensity (which has naturally larger numerical values).

To change the value being incremented, if applicable, the “c” key can be pressed which will switch the components current customisable value, moving the red box to the correct position.

### Electron movement

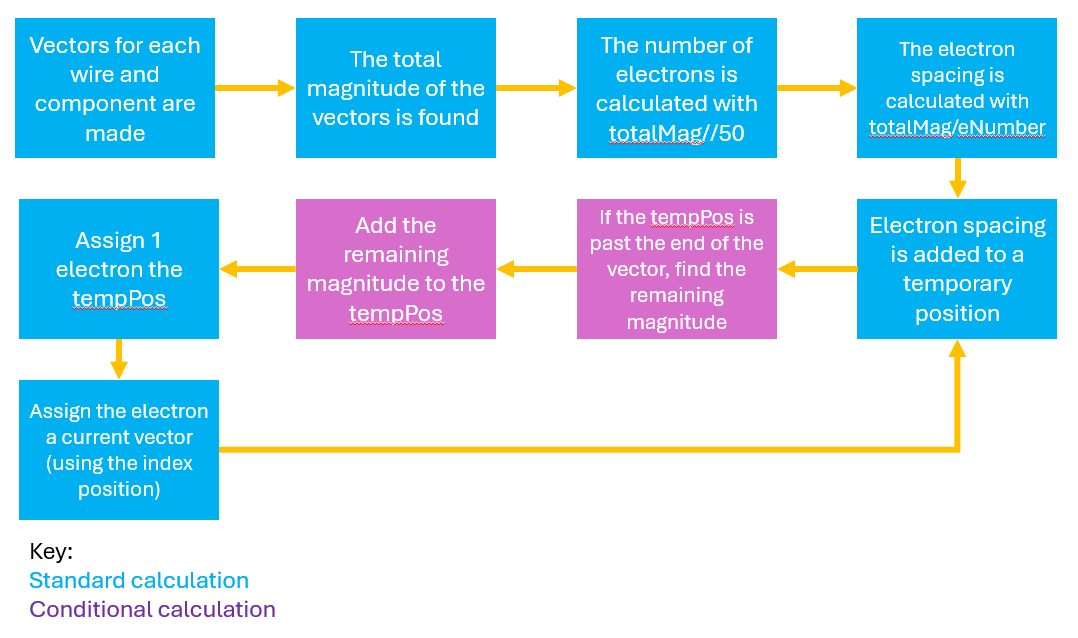
After the circuit is found to be complete and the vectors for the circuit are made, electrons are created as sprite objects.

Each electron is spaced approximately 50px apart, and so the number of electrons for each circuit must be calculated:

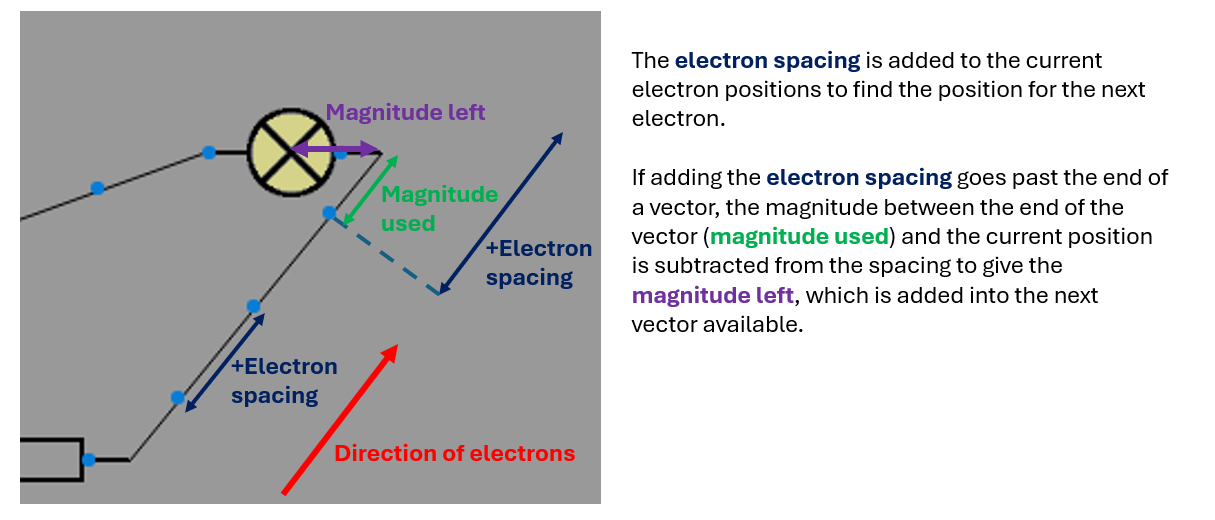
1. The list of vectors is iterated through and the total magnitude of the circuit is found.
2. The integer number of electrons (electronNumber) is found through floor division using the totalMagnitude//50.
3. The actual spacing (electronSpacing) between each electron is then found through regular division using the totalMagnitude/electronNumber.

Each electron is then spaced out along each wire and component. An initial position of the start of a vector is found and set to an electron. The next position to place an electron is found by adding the electronSpacing. If the next position is past the end of the vector then the distance between the current position and the end of the vector is found, the remaining distance is then added into the next vector, where the electron is placed.

Here is an abtracted diagram showing the processes involved in creating and placing the electrons in their initial positions:



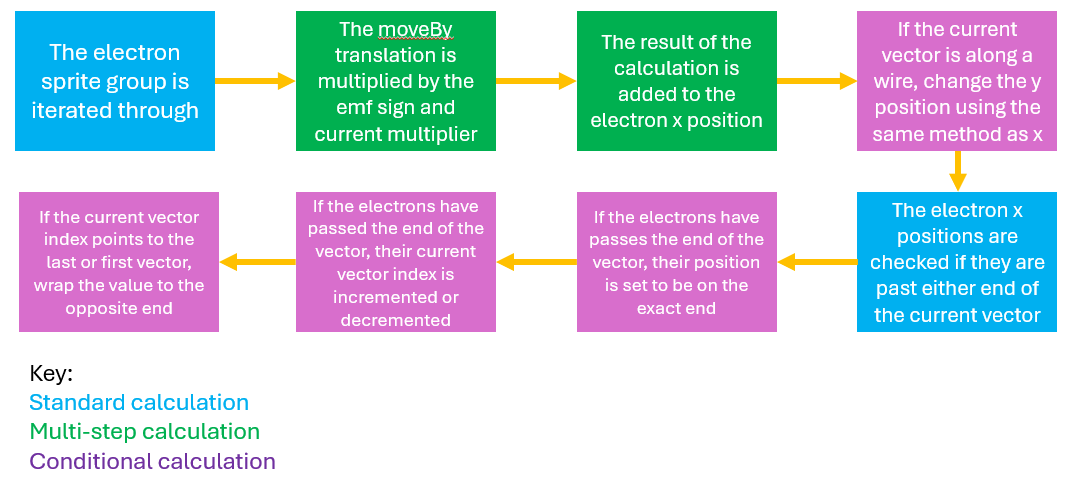
Here is a diagram showing how the initial position of each electron is calculated:



To move electrons:

1. Each electron is assigned a current vector index which is used to access the part of the circuit is on from the 2D vectorsCycle list stored in the graph.
2. The “moveBy” translation for the current vector of each electron is taken and multiplied by both the emfSign (direction of flow). The emf sign is 1, -1, or 0 which causes the circuit to add, subtract, or do nothing to the x and y values respectively.
3. The result of step 2 is then multiplied by the current multiplier which scales the translation to be larger or smaller to move the electrons at respective speeds.
4. The x positions, and y positions if on a non-component vector (Δy != 0), are then added to using the result from step 3.
5. The electron x positions are then checked for if they are greater then or less then (past the end of) the end of the current vector, in which case the electron currentVector index position is incremented or decremented to point at then next vector.

Here is an abstracted diagram of the processes involved in moving the electrons:



### Unit Graph

The user can use the graph buttons in the Tkinter window to plot the relationship between two specific units in the unit graph section of the Pygame window (bottom right), shown and explained in the UI descriptions above in the document.

There are 3 difference graph types to select:

* Current (y) against voltage (x) to show the I-V characteristic for the selected component.
* Resistance (y) against temperature (x) to show the relationship between resistance and temperature for a thermistor.
* Resistance (y) against light intensity (x) to show the relationship between resistance and light intensity for an LDR.

When the user clicks one of the buttons to select the graph, the x and y axis changes to show the value it is plotting and the unit the value uses, e.g. resistance, R, measured in ohms, Ω. The title of the graph is also changed to be “y” against “x” where y and x are the values on each axis respectively.

If no component is selected, the graph does not show any lines, however if a component is selected it will display the relationship between the two axis values for that specific component. If the values do not relate or have any characteristics for a particular component (e.g. a diode’s resistance does not change with light intensity), the graph will display “N/A” to show that relationship is not applicable for the selected component.

Here are all the graph relationship lines which are drawn with arcs and basic lines using Pygame:

A graph of a line

Description automatically generated with medium confidence

## User Interface

User interface image 1:

A screenshot of a computer

Description automatically generated

**4**

**5**

**6**

**9**

**8**

**7**

**10**

**3**

**2**

**1**

Image 1 references:

I chose to put the tkinter window with buttons to the left of the pygame window, as we naturally start on the left and move to the right (in reading), so the user can more easily follow the program as they first create components on the left and go to the right to build the circuit and move them around.

I also chose to put the main building area in-between the component buttons and the statistics section (on the right as ‘7’ and ‘9’) as it meant the user does not have to move the mouse as far to get the where the components will be made.

1. All components which can be selected and used in the circuit (From top to bottom: Cell, Junction, Resistor, Filament Lamp, Ammeter, Voltmeter, Thermistor, Light Dependant Resistor (LDR), Diode). I chose to put these at the top of the tkinter window as they are the first and most important part that the user will use. In addition to this, the components used more often in the A-level are closer to the top (e.g. cell, junction, resistor). They have images showing each component accurately.
2. Graph selection button to select the graph for (9). This has been placed under the components as it is the next most important button included. They have images showing the x and y axis, as well as clearer words in the centre to allow the user to see which graph they want to pick.
3. The reset button for the circuit. This is placed at the bottom of the tkinter window as it is the least likely (and likely the last) button to be used by the user.
4. An example circuit diagram with 4 components and 4 junctions. The components in the circuit as represented accurately with the correct symbols and implementation (e.g. the voltmeter can only be placed in parallel across a component).
5. When the circuit is complete the lamp image changes to be lit up, indicating the circuit is running. I decided to make this, although only for the lamp, because it can help the user to recognise that they have completed a circuit and show them when the circuit is broken.
6. The select component (right clicked) changes the image colour to red to indicate it is selected. The statistics box (7), and the graph (9) are updated to show the statistics and/or graph if applicable.
7. The statistics box shows the selected component’s (6) circuit attributes/values (e.g. voltage, current, and resistance). I chose to put this on the right of the screen along with the graph (9) as it means they are away from the circuit and tkinter window, since the user will not be interacting with them using the mouse pointer.
8. The current custom value for the selected component. This is shown clearly using a red box placed around the value to be changed, to show the user which value they will be changing pressing ‘c’ will change the current custom value if applicable (i.e. for a cell or diode). The increment step displays the amount the custom value will be changed by when the up and down arrow keys are pressed. This is displayed at the bottom of the statistics box so that it is always does not interfere with the attributes above it being displayed.
9. The graph shown the relationship between the x and y axis values selected (if applicable). This is shown in the UI as it can help the user to understand how certain attributes of a circuit interact with each other.
10. The basic controls for toggling both the labels, controls, and electrons.

User interface image 2:

A screenshot of a computer

Description automatically generated

Image 2 references:

Image 2 shows the design/appearance of the togglable labels, shown when 1 is pressed. I decided to use arrows to easily communicate the function of each part of the user interface, which is important for new users to understand the program. The labels are toggleable so that if the user fully understands how to use the circuit builder, they do not have to have the labels visible.

Looking at the simple circuit that is made in the building area, the electrons can be seen which show that the circuit is running and which direction the flow of electrons is. This is included in the UI as it can help the user understand when their circuit is complete and how the emf affects the speed and direction of the electrons.

User interface image 3 (zoomed in on pygame window):

A screenshot of a computer

Description automatically generated

Image 3 references:

Image 3 shows the controls (on the right) for the program. They are togglable to allow the user to view them when needed and are displayed in a different colour to stand out more in the interface.

# Testing

## Test Table

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Test Number | Description | Data Type | Expected Outcome | Outcome | Evidence (test video timestamp) |
| 1 | Press 1, then press 1 again. | Normal | When 1 is pressed, labels and arrows should appear that show the user how to work the program. When 1 is pressed again the labels should disappear. | Pass | 0:16-1:00 |
| 2 | Press 3, then press 3 again. | Normal | When 3 is pressed, the controls of the circuit should show and replace the graph and statistics area on the right of the screen. When 3 is pressed again the controls should disappear and the section with the graph should show again. | Pass | 1:00-1:25 |
| 3 | Press the cell button in the Tkinter window. | Normal | A cell component should be created and appear in the pygame window. | Pass | 1:25-1:35 |
| 4 | Hover the mouse over the cell and then away from the cell twice. | Normal | The cell component’s icon/image should change to have a slightly different appearance when the mouse pointer is hovering over it and change back to the original when the mouse is no longer hovering over it. | Pass | 1:35-1:50 |
| 5 | Hover the mouse over the cell and hold left click. Move the mouse around the screen to another point and release left click. | Normal | When left click is pressed and held while the mouse pointer is hovering over the cell, the cell should snap to the mouse pointer’s position while left click is still held and follow it, until left click is released when the cell will be left and updated to be in the location of the mouse pointer at that time (drag and drop). | Pass | 1:50-2:10 |
| 6 | Use the mouse to pick up the cell again, and while moving the cell, move the mouse (and therefore cell) around the entire pygame window. | Extreme | The cell should be picked up when the cell is clicked on and left click is held. The cell should follow the mouse across the entire pygame window and be able to move anywhere inside it. | Pass | 2:10-2:25 |
| 7 | Create and move every component in the Tkinter window using the 9 component buttons to various positions in around the pygame window. | Extreme | All components should be created when their respective buttons are pressed. All components should be able to be picked up and moved using drag and drop. | Pass | 2:25-2:53 |
| 8 | Right click on the cell component, then move the mouse pointer over and away from the cell. | Normal | When right clicked, the cell should change its icon/image to be red, and when the mouse is hovered on and off the cell the icon should change to be slightly blue and then plain red respectively. When right clicked the cell’s circuit attributes should show on the right side of the screen in the selected component box. This includes current, emf, (internal) resistance, power, and “lost volts”. | Pass | 2:54-3:32 |
| 9 | Right click all other components. | Extreme | When right clicked, each component’s icon/image should behave respectively to the cell did in the previous test. The selected component box on the right side of the screen should show all the correct attributes for that component. The junction should show current; the (fixed) resistor, (filament) lamp, thermistor, LDR, and diode should show current, voltage, resistance and power; the ammeter and voltmeter should show current and voltage respectively; the thermistor should additionally show temperature; the LDR should additionally show light intensity; and the diode should additionally show the minimum voltage value. | Pass | 3:32-4:13 |
| 10 | Right click one component, then move another component. | Erroneous | The component should turn red and become selected, showing its statistics. When the other component is moved and put down, the selected component should no longer be selected and return to black. | Pass | 4:14-4:30 |
| 11 | Select the I-V characteristic graph using the current against voltage graph button in the Tkinter window. | Normal | When pressed, the graph in the bottom right of the pygame window should change to have a title of “current against voltage”, and units should appear on each axis (I (current) measured in A (amps), and V (voltage/emf/potential difference) measured in V (volts)). | Pass | 4:30-4:47 |
| 12 | With the I-V graph selected, select (right click) the cell, and then select the resistor, lamp, thermistor, LDR, and finally diode. | Normal | When selected with a component, the graph should show the correct relationship between current and voltage for that component. | Pass | 4:47-5:35 |
| 13 | With the I-V graph selected, select the ammeter and then the voltmeter. | Erroneous | When selected with both the ammeter and voltmeter, the graph should not plot a line and instead show N/A, as the ammeter and voltmeter do not have an I-V relationship. | Pass | 5:35-5:50 |
| 14 | Select the resistance against light intensity graph in the Tkinter window and select the LDR. | Normal | The graph should change to show resistance against light intensity with the correct units, and when the LDR is selected the resistance against light intensity relationship for it should show. | Pass | 5:50-6:10 |
| 15 | Select all other components while the resistance against light intensity graph is selected. | Erroneous | The graph should display N/A for all components selected in this test (all but the LDR), as there is no relationship between resistance and light intensity for any of these components. | Pass | 6:10-6:39 |
| 16 | Select the resistance against temperature graph in the Tkinter window and select the thermistor. | Normal | The graph should change to show resistance against temperature with the correct units, and when the thermistor is selected the resistance against temperature relationship for it should show. | Pass | 6:39-6:53 |
| 17 | Select all other components while the resistance against temperature graph is selected. | Erroneous | The graph should display N/A for all components selected in this test (all but the thermistor), as there is no relationship between resistance and temperature for any of these components. | Pass | 6:53-7:12 |
| 18 | Selected the cell, then press the delete key. | Normal | The cell should be deleted and the attributes being shown for the cell in the top right should disappear. | Pass | 7:12-7:30 |
| 19 | Select any component, then press the reset button in the Tkinter window. | Normal | The pygame window should reset, all components created should be cleared and any statistics shown for selected components should also clear. | Pass | 7:30-7:55 |
| 20 | Create a cell, lamp, ammeter and junction and move them to four different places. Next hover the mouse near and far from each side of each component. Then select the junction and press delete. | Normal | When the mouse gets close to a component, a red circle indicator should appear at the edge (left or right) of the component, and then disappear when the mouse pointer gets further away. This should occur for both the left and right of each component. For the junction, the red circle indicator should be much smaller than for the other components (this is due to the junction being a much smaller component and so it has a smaller red circle indicator to allow the user to still select it. | Pass | 7:55-8:44 |
| 21 | Hover the mouse pointer over any red circle indicator and hold right click. Then move the mouse around. | Normal | When the mouse is hovering over the red indicator circle, a wire should be created and drawn between the correct edge of the correct component. When the mouse is moved around. The end of the wire not connected to the component should follow the pointer and its position should be updated when the mouse moves (dragging a wire). | Pass | 8:44-9:00 |
| 22 | While dragging a wire, move the mouse pointer far away from any other components. Then release right click. | Erroneous | When a wire is dragged far away from any components and right click is released, the wire should be removed from the screen and deleted as it was not connected to an components. | Pass | 9:00-9:12 |
| 23 | Re-drag a wire from the side of any component and move the mouse pointer close to and far from another component. | Normal | The red indicator circle should re-appear when the mouse pointer is dragging a wire close to another component. | Pass | 9:12-9:27 |
| 24 | Release right click when dragging and close to another component (red indicator circle is shown). | Normal | The wire’s end position on the mouse pointer should be updated to snap onto the edge of the component which shows the red indicator circle – the wire should now be connected between two components. | Pass | 9:27-9:43 |
| 25 | Hover the mouse pointer near and far from one of the sides of a component that has already been connected to a wire. | Erroneous | The red indicator circle should not show as two wires cannot be connected to the same point on a component. | Pass | 9:43-9:59 |
| 26 | Drag a wire from the remaining non-connected component and move the mouse pointer to the other side of the same component. | Erroneous | The red indicator circle should not show as a wire cannot be connected from one side of a component to the other side of the same component. | Pass | 9:59-10:20 |
| 27 | Connect a wire from the side of the remaining non-connected component and connect it to another component. Drag and drop the component with two wires connected, and then one of the components with 1 wire connected. | Normal | The wire should connect to the component correctly and when both components are moved, the ends of each connected wire should follow the correct side of the component being moved and the end position(s) of each wire should be updated. | Pass | 10:20-10:42 |
| 28 | Press the “s” key. | Normal | A screenshot should be taken and saved to the screenshots folder under the name screenshot(number) where number is the “number of screenshots before + 1”. A message pop up should show in the pygame window to notify the user that a screenshot has been taken. This should show for a short amount of time and then disappear. | Pass | 10:42-11:04 |
| 29 | Complete the circuit by connecting a wire between the two final non-connected sides of the components. | Normal | The circuit should be completed when the final wire snaps on the there is a loop found between the three components. The lamp icon/image should change to be lit up – an added indicator for the lamp to show that the circuit is complete. | Pass | 11:04-11:27 |
| 30 | Select each component, then select the ammeter, then press delete. Then select the cell and lamp again. | Normal | The circuit attributes should show on the right side of the screen, and they should be updated to have calculated values using physics. When the ammeter is connected it is in series (like other components) and shows only current in the selected component box. When deleted, the ammeter should disappear along with any wires connected between it and other components. The circuit should be reset, and the lamp’s icon should revert to not be filled in (turned off). When the lamp and cell are now selected, their non fixed attributes should be reset to 0 as the circuit has been broken. | Pass | 11:27-12:22 |
| 31 | Create a resistor and re-complete the circuit by dragging wires from the resistor to other components. Create a voltmeter and move it near and far from a component, before releasing left click when near a component. | Normal | The circuit should connect correctly with the resistor and re-complete with the lamp turned on. The voltmeter should be created and when moved near components, two red wires should appear to indicate that they’ll be created when the voltmeter is released. When the voltmeter gets far away from the component, the wires should disappear. When left click is released while dragging the voltmeter and near a component, the two red wires should be created and placed as black wires. | Pass | 12:22-13:03 |
| 32 | Select the voltmeter when it is connected to a component, then select the component. Move the voltmeter to connect to another component and select it again, then select the new component it is connected to. Connect the voltmeter to the cell and select it, then select the cell. | Normal | The voltmeter should turn red and only the voltage of the component it is connected to should show in the top right. When the voltmeter is moved, the wires connecting it to the component should disappear when it is moved away and re-appear when it is close to the next component. The voltmeter will turn red when selected again and the voltage value will be updated to be the same as the component it is connected to. When the voltmeter is connected to the cell and selected, its voltage value should be updated to be the same as the emf of the cell. | Pass | 13:03-13:45 |
| 33 | Press the left arrow key three times, then press the right arrow key three times. | Normal | The increment step at the bottom of the selected component box should change from 1, to 0.1, to 0.01, and back to 1 when pressing the left arrow key. The increment step should then change back to 0.01, then 0.1, then 1 again when the right arrow key is being pressed. This will control how much the custom circuit attribute for the selected component will increase and decrease by when the up and down arrow keys are pressed. | Pass | 13:52-14:20 |
| 34 | Select the lamp, then select the cell. Press the up arrow key 5 times, then press the down arrow key 5 times. Next press the up arrow key twice, change the increment step to 0.1 and press up 7 times, then change the increment step to 0.01 and press the up arrow key 5 times. Select the lamp again. Then select the resistor | Normal | The selected lamp should show a calculated circuit voltage value in the selected component box. When the up and down arrow keys are pressed 5 times respectively, the emf of the cell should increase by 5V and then decrease by 5V back to the original value. When the up arrow is pressed in the rest of this test, the emf should increase by the increment step value at the time to give a value 2.75 more than the original value. When the lamp is selected again, the voltage value should have increased and been updated to follow the increase in emf of the cell. Throughout the test the current in the components should increase and decrease respectively to emf. When selecting the resistor, it should be seen that the emf (7.75) is split between the lamp and resistor in an equal ratio to their respective resistances. | Pass | 14:20-15:46 |
| 35 | Select the lamp, then press the up arrow key 5 times, then the down arrow key 5 times. Then repeat this for the resistor. | Normal | The selected lamp should increase and decrease in resistance by “5 \* the increment step”. This should repeat for the resistor. When the resistance for both components is increased, the current in the circuit should decrease and the voltage for the component should increase. The power of each component will increase or decrease in proportion to the voltage across each component, however lowering overall with a higher total circuit resistance. When resistance is decreased the opposite of the previous explained sentence will happen to each attribute. | Pass | 15:52-16:17 |
| 36 | Select and delete the resistor, then create a new cell and connect two wires between the two open components and the cell. Select each cell, then select the lamp. | Normal | The total voltage and current on the lamp should be much higher than in previous tests will just 1 cell. This is due to the cells emf summing to give a total emf for the whole circuit. | Pass | 16:17-17:01 |
| 37 | Select each cell and use the arrow keys to set the emf of one cell to 5V, and then the other cell to -5V. Then select each other component. | Normal | The current in the circuit should be set to 0A as the total emf is (5)+(-5)=0V. The voltage and power in the lamp and resistor should both be 0 as the circuit is now not flowing. |  | 17:01-17:25 |
| 38 | Select and delete one cell and re-create and connect a resistor in its place. Select the cell (if not already, set the emf to 5V), press the “c” key twice. Select the resistor and press “c”, repeat for the lamp. | Normal | When the “c” key is pressed while the cell is selected, the red box outline indicating which attribute will change when arrow keys are pressed will move to be on resistance – allowing the user to change resistance. When the “c” key is pressed again, the box will go back to the emf. When “c” is pressed with the resistor or lamp selected, nothing will happen as only the resistance of the cell can be changed. | Pass | 17:25-18:20 |
| 39 | Select the cell and press “c” to make the resistance the current value being changed. Increase the resistance to 3 ohms, then decrease it again. | Normal | The internal resistance for the cell should increase and the icon/image of the cell should change to include a built-in resistor. This resistance should cause the “lost volts” value to be calculated. The resistance should decrease again and, when 0, the cell icon should revert back having no resistor. | Pass | 18:20-19:07 |
| 40 | Delete the lamp, create a thermistor, and connect it to the circuit. Select the thermistor and increase and decrease the temperature. | Normal | The lamp should be deleted, and any existing wires should be removed, the thermistor should be created and when connected, it’s temperature-resistance relationship should be calculated. When the temperature is changed, the resistance should increase/decrease exponentially, meaning each increase past it’s default value of 21 degrees should increase the resistance by more. | Pass | 19:07-19:47 |
| 41 | As in the previous test, replace the thermistor with an LDR. Select the LDR and increase and decrease light intensity. | Normal | When the LDR is connected, it’s light intensity-resistance relationship should be calculated. When the light intensity is changed, the resistance should increase/decrease exponentially, meaning each increase past it’s default value of 500 lux should increase the resistance by more. | Pass | 19:47-20:45 |
| 42 | Replace the LDR with a diode. Select the diode and increase and decrease the resistance. Press “c” and increase the minimum voltage using decimals so it falls just above the diode’s voltage, then decrease it again. Select the cell and decrease the emf so that the current turns to 0, select the diode. | Normal | The diode should be connected, and its resistance should allow it to act the same as a normal resistor certain values when resistance is increased and decreased. When “c” is pressed the customisable value should change to be minimum voltage. Increasing the minimum voltage should cause the current to fall to 0 when it increases above the diode’s voltage. Decreasing the emf should decrease the voltage value in the diode and cause it to fall below the minimum voltage, setting the current to 0. | Pass | 20:45-22:33 |
| 43 | Replace the diode with a lamp. Select the lamp and decrease its resistance to 0 using the down arrow key. Attempt decrease the resistance to a negative value by pressing the down arrow key again. | Erroneous | The lamp should be selected. When the down arrow key is pressed the resistance will decrease by the increment step each press until 0. When the down arrow key is pressed and the resistance is 0, nothing should happen as the resistance cannot be negative (it should stay at 0). | Pass | 22:33-23:07 |
| 44 | Select the resistor and decrease its resistance to 0. Then increase the resistance of both the lamp and resistor again by selecting them. | Extreme | The resistor should be selected, and its resistance should decrease to 0. The total resistance in the circuit should now be 0, and the current should be set to infinity. When the resistance of both components is increased, the current should no longer be 0 and be calculated once again. | Pass | 23:07-23:31 |
| 45 | Press the “2” key three times. | Normal | The circuit should reconnect with a lamp. The electrons should toggle on and off when “2” is pressed, and after three presses should show on (when initially off). The electrons should show following the path of the circuit at a given speed. They should be spaced roughly equal distances apart (~50px). | Pass | 23:31-23:52 |
| 46 | Select the cell and increase the emf. | Normal | Increasing the emf should increase the current in the circuit. The electrons should speed up and move faster around to circuit. | Pass | 23:52-24:08 |
| 47 | Decrease the emf to 0V. | Normal | Setting the emf to 0 should cause the current to be 0 and any electrons should become stationary (no current/flow of charge). | Pass | 24:08-24:20 |
| 48 | Decrease the emf to be negative, then increase it again to 5V. | Normal | Decreasing to emf to be negative should reverse the direction of the electrons and they should move in the opposite way to before. Increasing the emf to 5V again should cause the electrons to slow down until they’re stationary at 0V and then increase in speed in the original direction again to 5V. | Pass | 24:20-24:44 |
| 49 | Move the lamp further away from the rest of the components. Press “2”. | Normal | The electrons should disappear when the lamp is picked up and moved so that they do not follow an incorrect path. When “2” is pressed the electrons should reappear and they should be following the new path of the wires connected to the lamp. There should be more electrons in the circuit as it is larger, and they should still be spaced roughly equally apart. | Pass | 24:44-25:20 |
| 50 | Click the reset button, create a resistor. Then press backspace, then press tab. | Normal | The circuit board should be cleared and resistor should be created. Pressing backspace acts as an “undo” and so the creation of the resistor should be undone, and it should be deleted. Pressing tab acts as a “redo” button and so the creation of the resistor should be redone, and it should reappear and be re-created. | Pass | 25:32-25:58 |
| 51 | Move the resistor to two locations, press backspace 3 times, then press tab 3 times. | Normal | After the resistor is moved twice, the backspace button should undo the last actions, moving the resistor back through each previous position, before deleting it (the 3rd previous action). The tab button should redo these actions re-creating the resistor and moving it in the correct order to its last final position. | Pass | 25:58-26:20 |
| 52 | Move the resistor 8 more times around to various places on the screen. Press backspace 11 times, then press tab 11 times. | Extreme | The resistor should be moved 8 times, and when backspace is pressed, the resistor should backtrack through its previous positions in the correct order. After 10 presses, the 11th press should do nothing as the maximum number of stored actions is 10. Pressing redo 10 times should re-perform each movement in the correct order and the resistor should reach its final position. The 11th press should do nothing as only 10 actions can be redone as only 10 were undone. | Fail – This test almost passed, however the last action in the buffer stored the most recent position that the resistor moved to, rather than the least recent location. | 26:20-27:35 |
| 53 | Press backspace 5 times, move the resistor once, and press tab. | Normal | The resistor should backtrack through 5 previous positions in the correct order. After the resistor is moved, pressing tab should not redo any actions as a new action has been performed. | Pass | 27:35-28:05 |
| 54 | Select (right click) the resistor and press delete. Press backspace three times, then press tab three times. | Normal | The resistor should be deleted. Pressing backspace 1st will allow the user to undo the action of deleting the resistor and further presses will backtrack through previous movements. Pressing tab will allow them to re-track through the movements before deleting the resistor again. | Pass | 28:05-28:39 |
| 55 | Create a lamp and resistor and move them once to two positions. Connect a wire between the resistor and the lamp. Press backspace, then press tab. | Normal | After the resistor is connected to the lamp with a wire, pressing backspace should undo the creation of the wire (deleting it). Pressing tab should redo the creation of the wire (re-making and connecting it). | Pass | 28:39-29:12 |
| 56 | Press backspace three times, then press tab three times. | Normal | Pressing backspace three times should undo the last three actions in the correct order: the wire should be deleted, the lamp should be moved to its original position, the lamp should then be deleted. Pressing tab three times should redo these actions in the correct (opposite) order: the lamp will be created, then it will be moved, and the wire will be created again. | Pass | 29:12-29:39 |
| 57 | Select the lamp and press delete. Then move the resistor to another location. Press backspace twice, then press tab twice. | Normal | The lamp should be deleted and removed from the screen and the resistor should be moved. Pressing backspace will first move the resistor back to its previous position, and then secondly undo the deletion of the lamp re-instating the wire connected between them. Pressing tab will then redo through these actions in the correct order. | Pass | 29:39-30:17 |
| 58 | Reset the circuit and create and fully connect a circuit with a cell, lamp, and resistor (moving each component only once). Press backspace 10 times, then press tab 10 times. | Extreme | After the circuit is created and connected, pressing backspace 9 times should undo the entire circuit, backtracking through each action until the screen is clear. Pressing undo a 10th time should do nothing as only 9 actions were performed. Pressing tab 9 times should re-create the entire circuit and re-make all of the components and wires into their correct positions. Pressing tab a 10th time should do nothing as only 9 actions had been undone. | Pass | 30:17-31:23 |
| 59 | Move the resistor to a different location. Press undo, then press tab. | Normal | The wires should be updated to follow the resistor. When backspace is pressed they should be updated to be at the previous position, before tab is pressed where they should be updated to be at the new/final position again. | Pass | 31:23-31:57 |
| 60 | Following the previous test, press backspace 4 times, then select a component and delete it. Then press backspace three times and then tab three times. Re-create and connect the circuit so that it contains a cell, lamp, and resistor | Extreme | The past 4 actions should be backtracked through and undone. The component selected should be deleted and any wires (if present) connected to the component should also be removed. When backspace is pressed, the component should be “un-deleted” and the wires that were connecting the circuit (if they were there) should be reinstated, and the next two presses of backspace should backtrack through the next previous actions. Pressing tab should redo these two actions before re-deleting the component. | Pass | 31:57-33:00 |
| 61 | Create a voltmeter and connect it across a component. Select the component and delete it. Press backspace. | Normal | The voltmeter should be created and connected across a given component. When the component is selected and deleted, the wires connecting it to other circuit components should be removed and the wires connecting the component to the voltmeter should be removed. When backspace is pressed the component should be un-deleted, the voltmeter can now be re-connected again. | Pass | 33:00-33:33 |
| 62 | Connect the voltmeter to a component. Select the voltmeter, then press delete. Press backspace. | Normal | The voltmeter should be connected across a component. When selected and deleted, it should be removed and the wires connecting it to the component should also be removed. When backspace is pressed the voltmeter should be undeleted and it can now be re-moved to be connected again. | Pass | 33:33-33:56 |
| 64 | Connect the voltmeter to a component in the circuit a. Press “2”. | Erroneous | The voltmeter should be connected to a component in the circuit. When “2” is pressed the electrons flowing through the circuit should show. The electrons should not be flowing through the voltmeter as voltmeters are considered to have an infinite resistance so that they do not affect the current/flow of electrons in any way. | Pass | 33:56-34:29 |
| 65 | Reset the circuit with the reset button, press backspace, then press tab. | Normal | The circuit board should be reset. Pressing backspace and tab should do nothing as the circuit board has been reset and so past actions stored for undo/redoing are cleared. | Pass | 34:29-34:51 |
| 66 | Following the circuit reset, create three components which are not a cell, then create a cell. Connect wires between the three components and leave the cell outside the circuit. Select each component. | Erroneous | When the user connects the three components together to form a loop without a cell present, the circuit should not become active and should not run. This is due to the cell not being present in the graph cycle and so it is not “powering” the other components. Selecting each component should show that the physics have not been calculated. | Pass | 34:51-35:28 |
| 67 | Press backspace to undo the last wire that was made. Then move the cell into the gap left by the wire and connect it into the circuit cycle. | Normal | Pressing backspace to undo the last wire should remove it from the screen, and the cell should be able to be placed and connected into the circuit. As the cell is now in the circuit, it is powering the other components and so the circuit should be turned on and active. | Pass | 35:28-35:48 |
| 68 | Create a new lamp while the completed circuit is active. Then hover the mouse pointer near to the lamp which is not connected. Move the mouse to the left or right connector and press right click. | Erroneous | The newly created lamp is not part of the circuit and so should not be lit up (have the image with the bulb filled in yellow), as it is not part of the circuit. When the mouse is near the lamp, the user should not be able to create any new wires from the lamp as a cycle is already detected in the circuit. When the mouse is close to the side of the lamp the red proximity indicator circle should not show as wires cannot be dragged. | Pass | 35:48-36:33 |
| 69 | Select a component in the circuit and delete it. Then hover the mouse near the lamp on its own. Hover the mouse near the edge of the lamp and hold right click. Move the mouse pointer around. | Erroneous | When the component is deleted, hovering the mouse near the lamp (which is on its own) should show the red proximity indicator at the sides once again as the circuit is broken and so components can be connected again. Wires should now be able to be dragged again from the sides of the component. | Pass | 36:33-37:10 |

### Screenshot taken during testing.

Here is the screenshot taken during testing in the video:

A screenshot of a computer

Description automatically generated

# Technical Solution

## Contents

|  |  |  |
| --- | --- | --- |
| Line  Number | Code section title | Code description |
| 7 | Component Class | The class for creating, accessing, and updating components. |
| 206 | Wire Class | The class for creating, accessing, and updating wires. |
| 311 | Proximity Indicator Class | The class for creating, accessing, and updating the position of the red circle indicator that shows when the mouse pointer is within close proximity to a component. |
| 331 | Circular Buffer/Stack Class | The class containing the past 10 actions of the user, which can be accessed for adding and undo/redoing actions for creating, moving, and deleting components; and creating wires. |
| 494 | Graph (data structure) Class | The class containing all the components and wires which are implemented into a graph data structure as nodes and edges respectively. This class is also used for running a DFS algorithm to check for a cycle (complete circuit), in which if so vectors and electrons are created to allow electrons to flow and follow the paths of each wire and component. The DFS algorithm used in the circuit test for cycles is a modified version of a python implementation for a directed graph cycle test online (https://www.geeksforgeeks.org/detect-cycle-direct-graph-using-colors/). |
| 737 | Unit Graph Class | The class containing the current axes for the graph that has been selected. It can be accessed to plot all applicable relationships for each component and graph. |
| 794 | Electron Class | The class for creating, accessing, and updating electrons. |
| 820 | Vector Class | The class for creating and accessing vectors for electrons to follow. |
| 850 | FindNearestConnector subroutine. | The subroutine for finding the nearest available connector (left or right of a component) to the mouse position, providing it is within 75px of the mouse. |
| 901 | Connection Reset subroutine | The subroutine for resetting sprite and wire attributes when a component is deleted. |
| 916 | NumberOfSprites subroutine | The subroutine for assigning a new sprite its ID, using the length of the ID-Tracker sprite group and adding 1. |
| 924 | Physics Calculation subroutine | The subroutine for calculating all undetermined circuit values for each component. This uses various physics calculations and formulas to calculate and update each component to have the correct calculated value. |
| 1019 | drawStatsText subroutine | The subroutine for drawing all component circuit attributes into the statistics/selected component box on the right hand side of the screen. This allows the user to see the circuit values which have been calculated. |
| 1073 | resetCircuitBoard subroutine | The subroutine for resetting all objects when the reset button is pressed. This clears all sprite groups, the graph and stack data structures, as well as any other objects. |
| 1087 | Pre-determined global objects and variables are created. | This includes the circular stack and empty graph data structures, any sprite groups, the proximity indicator circle, and each components specific circuit attributes. |
| 1144 | Tkinter thread and Tkinter Window Class | This is where the tkinter thread is run using threading and the window is initiated as a class. All component and graph buttons are created as well as the reset button, which are put into a grid with padding to space out and separate each button type. The tkinter class also contains the methods for each button allowing, each component to be created; the graph to be selected; and the circuit to be reset. |
| 1253 | Pygame thread | This is where the pygame thread is run using threading. The pygame window is initiated and constant images and shapes that show to the screen are created. Any pygame specific variables are also created. |
| 1349 | Event loop and event detection | Includes pygame processes such as closing the program and taking inputs from the keyboard. (As well as other computation of the program). |
| 1540 | Click and drag for components | This is the section of code which allows the components to be picked up and moved. |
| 1610 | Right clicking to select components and drag wires | This is the section of code which allows the user to right click a component and select it to change its attributes and see different graph relationships. Wires can also be dragged using right click when the mouse is on a red proxCircle. |
| 1679 | Showing any constant images and shapes | This is the section of code which, under various conditions, shows the pre-determined/constant images for pygame. For example, if the user has toggled labels so that “labelsOn == True”, the labels text will be shown. Any graph relationships will be shown by calling the specific plot method for the unit graph. |
| 1816 | Showing any components, wires, and the proximity indicator to the screen | This is the section of the code which iterates through each sprite and, depending on the object type, shows them and updates their position if they are being moved. |
| 1872 | Electron position updating | This is the section of the code which iterates through each electron and updates their positon to move around the circuit, using their current vector index and accessing the vectors stored in the circuit’s graph to move by an amount, according to the current in the circuit (to determine speed), and the emfs sign (positive or negative) to determine direction. |
| 1920 | Screenshot saving | This is the section of the code which if run when the user has pressed “s”. It causes a message to be displayed for 400 cycles of the event loop (~1-1.5s) to show a screenshot has been taken, as well as reading the screenshotData file to name the next screenshot by the next number, before saving it in the screenshots folder. |
| 1947 | Threading initialisation | This is where the threads for pygame and tkinter are created and assigned each subroutine before being started. |

## Code

|  |
| --- |
| 1 #Import everything  2 **import** pygame**,**pickle**,**threading**,**os**,**copy**,**math  3 **import** tkinter **as** tk  4 **from** tkinter **import** **\***  5 **from** tkinter**.***ttk* **import** **\***  6  7 ###Class for all components  8 **class** **Component(**pygame**.***sprite***.***Sprite***):**  9 #Constructor  10 **def** \_\_init\_\_**(**self**,**name**,**ID**,**attributes**,**initialPosition**):**  11 pygame**.***sprite***.***Sprite***.***\_\_init\_\_***(**self**)**  12 #Name used to identify component type  13 self**.***name* **=** name  14 #ID used to identify specific component  15 self**.***ID* **=** ID  16 self**.***pickupStatus* **=** **False**  17 #Sets all existing varibles to be attributes in the component  18 #e.g. if the component has passed in in attributes "voltage" it makes an attribute  19 variables **=** **vars(**self**)**  20 **for** key **in** attributes**:**  21 variables**[**key**]** **=** attributes**[**key**]**  22 self**.***surf* **=** pygame**.***image***.***load***(**f'graphics/{self**.***name*}Base.png'**)**  23 self**.***image* **=** 'base'  24 self**.***rightClicked* **=** **False**  25 self**.***rect* **=** self**.***surf***.***get\_rect***(**center **=** initialPosition**)**  26 self**.***currentPosition* **=** initialPosition  27 self**.***x* **=** self**.***currentPosition***[**0**]**  28 self**.***y* **=** self**.***currentPosition***[**1**]**  29 self**.***left* **=** self**.***rect***.***midleft*  30 self**.***right* **=** self**.***rect***.***midright*  31 self**.***top* **=** self**.***rect***.***midtop*  32 self**.***bottom* **=** self**.***rect***.***midbottom*  33 #Stores connecting points for finding nearest sprite = left/right/top/bottom coords of component image  34 self**.***connectors* **=** **{**'left'**:** self**.***left***,**'right'**:** self**.***right***,**'top'**:** self**.***top***,**'bottom'**:** self**.***bottom***}**  35 #Stores connected wireSprites at left/right/top/bottom  36 #Wires can only be connected to left/right, but top and bottom are shown to allow voltmeters  37 # to find the nearest sprite when left/right are already used.  38 self**.***connected* **=** **{**'left'**:** **None,**'right'**:** **None,**'top'**:** **None,**'bottom'**:** **None}**  39 #Is sprite part of the circuit graph  40 self**.***active* **=** **False**  41 pygame**.***sprite***.***Group***.***add***(**self**)**  42 componentSpriteGroup**.***add***(**self**)**  43 IDSprites**.***add***(**self**)**  44 allSprites**.***add***(**self**)**  45 IDTracker**.***add***(**self**)**  46 #Add conditional attributes for specific sprites  47 **if** self**.***name* **==** 'voltmeter'**:**  48 self**.***voltWires* **=** **{**'left'**:** **None,** 'right'**:** **None}**  49 **elif** self**.***name* **==** 'thermistor'**:**  50 self**.***prevTemp* **=** 21  51 **elif** self**.***name* **==** 'ldr'**:**  52 self**.***prevLI* **=** 500  53 self**.***left* **=** **(**self**.***rect***.***midleft***[**0**],**self**.***rect***.***midleft***[**1**]+**3**)**  54 self**.***right* **=** **(**self**.***rect***.***midright***[**0**],**self**.***rect***.***midright***[**1**]+**3**)**  55  56 #Update sprite when it is being moved  57 **def** update**(**self**):**  58 self**.***left* **=** self**.***rect***.***midleft*  59 self**.***right* **=** self**.***rect***.***midright*  60 self**.***top* **=** self**.***rect***.***midtop*  61 self**.***bottom* **=** self**.***rect***.***midbottom*  62 self**.***x* **=** self**.***currentPosition***[**0**]**  63 self**.***y* **=** self**.***currentPosition***[**1**]**  64 self**.***connectors* **=** **{**'left'**:**self**.***left***,**'right'**:**self**.***right***,**'top'**:**self**.***top***,**'bottom'**:**self**.***bottom***}**  65 **if** self**.***name* **==** 'ldr'**:**  66 self**.***left* **=** **(**self**.***rect***.***midleft***[**0**],**self**.***rect***.***midleft***[**1**]+**3**)**  67 self**.***right* **=** **(**self**.***rect***.***midright***[**0**],**self**.***rect***.***midright***[**1**]+**3**)**  68 **if** self**.***name* **==** 'voltmeter'**:**  69 **if** self**.***voltWires***[**'left'**]** **==** **None:**  70 self**.***pd* **=** 0  71  72 #Get every attribute of the component as a dictionary  73 **def** allAttributes**(**self**):**  74 **return** **vars(**self**)**  75  76 #Toggles image between base (black) and red when called for being selected  77 **def** toggleImage**(**self**):**  78 **if** self**.***image* **==** 'base'**:**  79 self**.***image* **=** 'red'  80 **else:**  81 self**.***image* **=** 'base'  82 image **=** self**.***image***.***title***()**  83 **if** self**.***name* **==** 'lamp'**:**  84 **if** self**.***active* **==** **True:**  85 self**.***surf* **=** pygame**.***image***.***load***(**f'graphics/{self**.***name*}{image}On.png'**)**  86 **else:**  87 self**.***surf* **=** pygame**.***image***.***load***(**f'graphics/{self**.***name*}{image}.png'**)**  88 **elif** self**.***name* **==** 'cell'**:**  89 **if** self**.***resistance* **>** 0**:**  90 self**.***surf* **=** pygame**.***image***.***load***(**f'graphics/{self**.***name*}Resistor{image}.png'**)**  91 self**.***rect* **=** self**.***surf***.***get\_rect***(**midleft **=** self**.***left***)**  92 self**.***update***()**  93 **else:**  94 self**.***surf* **=** pygame**.***image***.***load***(**f'graphics/{self**.***name*}{image}.png'**)**  95 self**.***rect* **=** self**.***surf***.***get\_rect***(**midleft **=** self**.***left***)**  96 self**.***update***()**  97 **else:**  98 self**.***surf* **=** pygame**.***image***.***load***(**f'graphics/{self**.***name*}{image}.png'**)**  99  100 #Change current component image to have a blue tint when the mouse is hovering over it  101 **def** hover**(**self**,**changeType**):**  102 image **=** self**.***image***.***title***()**  103 **if** self**.***name* **==** 'lamp'**:**  104 **if** circuitGraph**.***circuitValid* **==** **True** **and** self**.***active* **==** **True:**  105 self**.***surf* **=** pygame**.***image***.***load***(**f'graphics/{self**.***name*}{image}On{changeType}.png'**)**  106 **else:**  107 self**.***surf* **=** pygame**.***image***.***load***(**f'graphics/{self**.***name*}{image}{changeType}.png'**)**  108 **elif** self**.***name* **==** 'cell'**:**  109 **if** self**.***resistance* **>** 0**:**  110 self**.***surf* **=** pygame**.***image***.***load***(**f'graphics/{self**.***name*}Resistor{image}{changeType}.png'**)**  111 self**.***rect* **=** self**.***surf***.***get\_rect***(**center **=** self**.***currentPosition***)**  112 self**.***update***()**  113 **else:**  114 self**.***surf* **=** pygame**.***image***.***load***(**f'graphics/{self**.***name*}{image}{changeType}.png'**)**  115 self**.***update***()**  116 **else:**  117 self**.***surf* **=** pygame**.***image***.***load***(**f'graphics/{self**.***name*}{image}{changeType}.png'**)**  118  119 #Save current sprite attributes and positions to an actions file for undo/redo  120 **def** save**(**self**,**saveType**):**  121 allAttributes **=** self**.***allAttributes***()**  122 attributesForPickle **=** **[]**  123 **for** attribute **in** allAttributes**:**  124 #Excludes any attributes containing surfaces for serialising  125 **if** attribute **!=** 'surf' **and** attribute **!=** '\_Sprite\_\_g' **and** attribute **!=** 'connected' **and** attribute **!=** 'undoneConnectors' **and** attribute **!=** 'voltWires'**:**  126 attributesForPickle**.***append***({**attribute**:** allAttributes**[**attribute**]})**  127 #Saves the sprite to "actions[filePointer]" text file by pickling  128 **with** **open(**f'actionsFolder/action{actions**.***savedFilePointer*}'**,** 'wb'**)** **as** file**:**  129 pickle**.***dump***(**attributesForPickle**,** file**)**  130 #Move file pointer forwards (if new action)  131 **if** saveType **==** 'new'**:**  132 actions**.***savedFilePointer* **+=** 1  133 actions**.***savedFilePointer* **=** actions**.***savedFilePointer***%**actions**.***maximum*  134  135 #Undo sprite movement by loading previous file and updating positions  136 **def** undo**(**self**):**  137 #Save sprite without moving pointer  138 self**.***save***(**'old'**)**  139 actions**.***savedFilePointer* **-=** 1  140 **if** actions**.***savedFilePointer* **==** **-**1**:**  141 actions**.***savedFilePointer* **+=** actions**.***maximum*  142 #Load previous file  143 **with** **open(**f'actionsFolder/action{actions**.***savedFilePointer*}'**,** 'rb'**)** **as** file**:**  144 unPickled **=** pickle**.***load***(**file**)**  145 #Iterate through unpickled file and re-assign/update all attributes  146 **for** attribute **in** unPickled**:**  147 **for** attributeName **in** attribute**:**  148 **vars(**self**)[**attributeName**]** **=** attribute**[**attributeName**]**  149 self**.***pickupStatus* **=** **False**  150 self**.***x* **=** self**.***currentPosition***[**0**]**  151 self**.***y* **=** self**.***currentPosition***[**1**]**  152 self**.***rect* **=** self**.***surf***.***get\_rect***(**center **=** self**.***currentPosition***)**  153 self**.***left* **=** self**.***rect***.***midleft*  154 self**.***right* **=** self**.***rect***.***midright*  155 self**.***top* **=** self**.***rect***.***midtop*  156 self**.***bottom* **=** self**.***rect***.***midbottom*  157 self**.***connectors* **=** **{**'left'**:**self**.***left***,**'right'**:**self**.***right***,**'top'**:**self**.***top***,**'bottom'**:**self**.***bottom***}**  158 #Iterate through "connected" to find any connected wires and update the ends of any existing wires  159 **for** key **in** self**.***connected***:**  160 **if** self**.***connected***[**key**]:**  161 wire **=** self**.***connected***[**key**]**  162 **if** wire**.***sprites***[**0**]** **==** self**:**  163 wire**.***ends***[**0**]** **=** self**.***connectors***[**key**]**  164 **elif** wire**.***sprites***[**1**]** **==** self**:**  165 wire**.***ends***[**1**]** **=** self**.***connectors***[**key**]**  166  167 #Redo sprite movement by loading next file and updating positions  168 **def** redo**(**self**):**  169 #Save sprite without moving pointer  170 self**.***save***(**'old'**)**  171 actions**.***savedFilePointer* **+=** 1  172 actions**.***savedFilePointer* **=** actions**.***savedFilePointer***%**actions**.***maximum*  173 #Load next file  174 **with** **open(**f'actionsFolder/action{actions**.***savedFilePointer*}'**,** 'rb'**)** **as** file**:**  175 unPickled **=** pickle**.***load***(**file**)**  176 #Re-assign and update all attributes from loaded file  177 **for** attribute **in** unPickled**:**  178 **for** attributeName **in** attribute**:**  179 **vars(**self**)[**attributeName**]** **=** attribute**[**attributeName**]**  180 self**.***pickupStatus* **=** **False**  181 self**.***x* **=** self**.***currentPosition***[**0**]**  182 self**.***y* **=** self**.***currentPosition***[**1**]**  183 self**.***rect* **=** self**.***surf***.***get\_rect***(**center **=** self**.***currentPosition***)**  184 self**.***left* **=** self**.***rect***.***midleft*  185 self**.***right* **=** self**.***rect***.***midright*  186 self**.***top* **=** self**.***rect***.***midtop*  187 self**.***bottom* **=** self**.***rect***.***midbottom*  188 self**.***connectors* **=** **{**'left'**:**self**.***left***,**'right'**:**self**.***right***,**'top'**:**self**.***top***,**'bottom'**:**self**.***bottom***}**  189 #Update wire end positions  190 **for** key **in** self**.***connected***:**  191 **if** self**.***connected***[**key**]:**  192 wire **=** self**.***connected***[**key**]**  193 **if** wire**.***sprites***[**0**]** **==** self**:**  194 wire**.***ends***[**0**]** **=** self**.***connectors***[**wire**.***spriteConnectors***[**0**]]**  195 **elif** wire**.***sprites***[**1**]** **==** self**:**  196 wire**.***ends***[**1**]** **=** self**.***connectors***[**wire**.***spriteConnectors***[**1**]]**  197  198 #Reset component circuit values when circuit is broken  199 **def** reset**(**self**):**  200 **if** self**.***active* **==** **True:**  201 variables **=** **vars(**self**)**  202 atts **=** **globals()[**f'{self**.***name*}Attributes'**]**  203 **for** key **in** atts**:**  204 variables**[**key**]** **=** atts**[**key**]**  205  206 ###Class for Wires  207 **class** **Wire(**pygame**.***sprite***.***Sprite***):**  208 #Constructor  209 **def** \_\_init\_\_**(**self**,**ID**,**start**,**end**,**inputType**):**  210 pygame**.***sprite***.***Sprite***.***\_\_init\_\_***(**self**)**  211 self**.***name* **=** 'wire'  212 self**.***inputType* **=** inputType  213 self**.***ID* **=** ID  214 self**.***end* **=** end  215 #Start and end position  216 self**.***ends* **=** **[**start**,**end**]**  217 #Respective start/end sprites  218 self**.***sprites* **=** **[None,None]**  219 #Respective start/end sprite connectors (left/right)  220 self**.***spriteConnectors* **=** **[None,None]**  221 self**.***prevData* **=** **[]**  222 #Is wire being dragged  223 self**.***placed* **=** **False**  224 self**.***image* **=** 'base'  225 self**.***color* **=** '#000000'  226 allSprites**.***add***(**self**)**  227 IDSprites**.***add***(**self**)**  228 IDTracker**.***add***(**self**)**  229  230 #Update end of wire when it is no longer being dragged  231 **def** updateEnd**(**self**,**nearestPoint**,**makingWire**,**wireOriginSprite**,**wireOriginConnector**,):**  232 #If wire is being dragged, set end to mouse\_pos  233 **if** makingWire **==** **True:**  234 self**.***ends***[**1**]** **=** pygame**.***mouse***.***get\_pos***()**  235 **else:**  236 #If there is a non-used connector in 75px:  237 **if** nearestPoint **!=** **None:**  238 **if** nearestPoint**[**1**]** **!=** 'top' **and** nearestPoint**[**1**]** **!=** 'bottom'**:**  239 #Show proxCircle at nearestPoint  240 self**.***ends***[**1**]** **=** proxCircle**.***pos*  241 self**.***placed* **=** **True**  242 wireOriginCon **=** **None**  243 nearestCon **=** **None**  244 #Find and set [sprites] and [spriteConnectors]  245 **for** key **in** wireOriginSprite**.***connectors***:**  246 **if** wireOriginSprite**.***connectors***[**key**]** **==** self**.***ends***[**0**]:**  247 wireOriginCon **=** key  248 **for** key **in** nearestPoint**[**0**].***connectors***:**  249 **if** nearestPoint**[**0**].***connectors***[**key**]** **==** self**.***ends***[**1**]:**  250 nearestCon **=** key  251 self**.***sprites* **=** **[**wireOriginSprite**,**nearestPoint**[**0**]]**  252 self**.***spriteConnectors* **=** **[**wireOriginCon**,**nearestCon**]**  253 **for** key **in** wireOriginSprite**.***connected***:**  254 **if** wireOriginConnector **==** key**:**  255 wireOriginSprite**.***connected***[**key**]** **=** self  256 **for** key **in** nearestPoint**[**0**].***connected***:**  257 **if** nearestPoint**[**1**]** **==** key**:**  258 nearestPoint**[**0**].***connected***[**key**]** **=** self  259 #Add wire creation to actions stack  260 actions**.***add***([**self**,**'createdWire'**])**  261 #Add edge to circuit graph  262 circuitGraph**.***addEdge***(**self**.***sprites***[**0**].***ID***,**self**.***sprites***[**1**].***ID***)**  263  264 #If component is being moved and wire is placed down, update the end of the wire to follow the component  265 **def** update**(**self**):**  266 **for** sprite **in** self**.***sprites***:**  267 **if** sprite**.***pickupStatus* **==** **True:**  268 **for** key **in** sprite**.***connectors***:**  269 **if** sprite **==** self**.***sprites***[**0**]:**  270 self**.***ends***[**0**]** **=** sprite**.***connectors***[**self**.***spriteConnectors***[**0**]]**  271 **break**  272 **elif** sprite **==** self**.***sprites***[**1**]:**  273 self**.***ends***[**1**]** **=** sprite**.***connectors***[**self**.***spriteConnectors***[**1**]]**  274 **break**  275  276 #Get all attributes  277 **def** allAttributes**(**self**):**  278 **return** **vars(**self**)**  279  280 #Reset self  281 **def** reset**(**self**):**  282 self**.***placed* **=** **False**  283 self**.***sprites* **=** **[None,None]**  284 self**.***spriteConnectors* **=** **[None,None]**  285 allSprites**.***remove***(**self**)**  286 IDSprites**.***remove***(**self**)**  287  288 #Undo wire by saving data to prevData attribute and reset attributes  289 **def** undoCreation**(**self**,**deletedSprite**):**  290 self**.***prevData* **=** **[**copy**.***copy***(**self**.***ends***),**copy**.***copy***(**self**.***sprites***),**copy**.***copy***(**self**.***spriteConnectors***)]**  291 **for** component **in** self**.***sprites***:**  292 **if** component **!=** deletedSprite**:**  293 **for** key **in** component**.***connected***:**  294 **if** component**.***connected***[**key**]** **==** self**:**  295 component**.***connected***[**key**]** **=** **None**  296 self**.***reset***()**  297  298 #Redo wire by re-initialising attirbutes using prevData  299 **def** redoCreation**(**self**):**  300 self**.***placed* **=** **True**  301 self**.***ends* **=** self**.***prevData***[**0**]**  302 self**.***sprites* **=** self**.***prevData***[**1**]**  303 self**.***spriteConnectors* **=** self**.***prevData***[**2**]**  304 **if** self**.***sprites***[**0**].***connected***[**self**.***spriteConnectors***[**0**]]** **==** **None:**  305 self**.***sprites***[**0**].***connected***[**self**.***spriteConnectors***[**0**]]** **=** self  306 **if** self**.***sprites***[**1**].***connected***[**self**.***spriteConnectors***[**1**]]** **==** **None:**  307 self**.***sprites***[**1**].***connected***[**self**.***spriteConnectors***[**1**]]** **=** self  308 allSprites**.***add***(**self**)**  309 IDSprites**.***add***(**self**)**  310  311 ###Class for red proximity indicator (for when the mouse is hovering near an available connector)  312 **class** **ProxCircle(**pygame**.***sprite***.***Sprite***):**  313 #Constructor  314 **def** \_\_init\_\_**(**self**,**name**,**ID**,**pos**):**  315 pygame**.***sprite***.***Sprite***.***\_\_init\_\_***(**self**)**  316 self**.***name* **=** name  317 self**.***ID* **=** ID  318 self**.***pos* **=** pos  319 self**.***surf* **=** pygame**.***image***.***load***(**'graphics/proxCircle.png'**)**  320 self**.***rect* **=** self**.***surf***.***get\_rect***(**center **=** self**.***pos***)**  321 #Set the position of the proxCircle to the closest point available  322 **def** showClosest**(**self**,**nearestPoint**,**wireOrigin**):**  323 **if** nearestPoint**[**2**]** **<=** 75**:**  324 **if** nearestPoint**[**1**]** **==** 'left'**:**  325 self**.***rect* **=** self**.***surf***.***get\_rect***(**center **=** nearestPoint**[**0**].***left***)**  326 self**.***pos* **=** nearestPoint**[**0**].***left*  327 **elif** nearestPoint**[**1**]** **==** 'right'**:**  328 self**.***rect* **=** self**.***surf***.***get\_rect***(**center **=** nearestPoint**[**0**].***right***)**  329 self**.***pos* **=** nearestPoint**[**0**].***right*  330  331 ###Class for circular buffer/stack which stores the users previous actions for undo/redo  332 **class** **Cstack():**  333 #Constructor  334 **def** \_\_init\_\_**(**self**,**name**,**maximum**):**  335 self**.***name* **=** name  336 self**.***stack* **=** **[None]\***maximum  337 self**.***maximum* **=** maximum  338 self**.***front* **=** 0  339 self**.***rear* **=** **-**1  340 self**.***usedItems* **=** 0  341 self**.***undoneItems* **=** 0  342 #File pointer for loading component positions  343 self**.***savedFilePointer* **=** 0  344 #Reset stack  345 **def** reset**(**self**):**  346 self**.***stack* **=** **[None]\***self**.***maximum*  347 self**.***front* **=** 0  348 self**.***rear* **=** **-**1  349 self**.***usedItems* **=** 0  350 self**.***undoneItems* **=** 0  351 self**.***savedFilePointer* **=** 0  352  353 #Add action to stack  354 **def** add**(**self**,**item**):**  355 #If all slots are used, set front item and update pointers  356 **if** self**.***usedItems* **==** self**.***maximum***:**  357 self**.***stack***[**self**.***front***]** **=** item  358 self**.***front* **+=** 1  359 self**.***rear* **+=** 1  360 self**.***front* **=** self**.***front***%**self**.***maximum*  361 self**.***rear* **=** self**.***rear***%**self**.***maximum*  362 **else:**  363 #If not all slots are "used"  364 self**.***usedItems* **+=** 1  365 self**.***rear* **+=** 1  366 self**.***rear* **=** self**.***rear***%**self**.***maximum*  367 self**.***stack***[**self**.***rear***]** **=** item  368 self**.***undoneItems* **=** 0  369 **if** self**.***stack***[**self**.***rear***][**1**]** **!=** 'createdComponent'**:**  370 self**.***action***()**  371  372 #Undo action in stack  373 **def** undo**(**self**):**  374 actionUndone **=** self**.***stack***[**self**.***rear***]**  375 #If stack not empty  376 **if** self**.***usedItems* **>** 0**:**  377 self**.***usedItems* **-=** 1  378 self**.***undoneItems* **+=** 1  379 self**.***rear* **-=** 1  380 **if** self**.***rear* **==** **-**1**:**  381 self**.***rear* **+=** self**.***maximum*  382 **if** actionUndone **!=** **None:**  383 undoneSprite **=** actionUndone**[**0**]**  384 #Undo component creation  385 **if** actionUndone**[**1**]** **==** 'createdComponent'**:**  386 allSprites**.***remove***(**undoneSprite**)**  387 IDSprites**.***remove***(**undoneSprite**)**  388 componentSpriteGroup**.***remove***(**undoneSprite**)**  389 circuitGraph**.***removeNode***(**undoneSprite**.***ID***)**  390 #Undo component movement  391 **elif** actionUndone**[**1**]** **==** 'movedComponent'**:**  392 **if** actionUndone**[**0**].***name* **==** 'voltmeter'**:**  393 **for** key **in** actionUndone**[**0**].***voltWires***:**  394 IDSprites**.***remove***(**actionUndone**[**0**].***voltWires***[**key**])**  395 allSprites**.***remove***(**actionUndone**[**0**].***voltWires***[**key**])**  396 undoneSprite**.***undo***()**  397 **for** sprite **in** componentSpriteGroup**:**  398 **if** sprite**.***name* **==** 'voltmeter'**:**  399 **if** sprite**.***voltWires***[**'left'**]:**  400 voltConnectedSprite **=** sprite**.***voltWires***[**'left'**].***sprites***[**1**]**  401 **if** voltConnectedSprite **==** actionUndone**[**0**]:**  402 sprite**.***voltWires***[**'left'**].***ends***[**1**]** **=** actionUndone**[**0**].***left*  403 sprite**.***voltWires***[**'right'**].***ends***[**1**]** **=** actionUndone**[**0**].***right*  404 circuitGraph**.***resetVectors***()**  405 #Undo component deletion  406 **elif** actionUndone**[**1**]** **==** 'deletedComponent'**:**  407 allSprites**.***add***(**undoneSprite**)**  408 IDSprites**.***add***(**undoneSprite**)**  409 componentSpriteGroup**.***add***(**undoneSprite**)**  410 circuitGraph**.***addNode***(**undoneSprite**.***ID***)**  411 **for** key **in** undoneSprite**.***connected***:**  412 **if** undoneSprite**.***connected***[**key**]** **!=** **None:**  413 wire **=** undoneSprite**.***connected***[**key**]**  414 wire**.***redoCreation***()**  415 circuitGraph**.***addEdge***(**wire**.***sprites***[**0**].***ID***,**wire**.***sprites***[**1**].***ID***)**  416 circuitGraph**.***circuitTest***()**  417 #Undo wire creation  418 **elif** actionUndone**[**1**]** **==** 'createdWire'**:**  419 circuitGraph**.***removeEdge***(**undoneSprite**)**  420 circuitGraph**.***circuitTest***()**  421 **if** circuitGraph**.***circuitValid* **==** **False:**  422 **for** sprite **in** componentSpriteGroup**:**  423 sprite**.***reset***()**  424 undoneSprite**.***undoCreation***(None)**  425 self**.***action***()**  426  427 #Redo action in stack  428 **def** redo**(**self**):**  429 #If stack not full  430 **if** self**.***usedItems* **<** self**.***maximum* **and** self**.***undoneItems* **!=** 0**:**  431 self**.***usedItems* **+=** 1  432 self**.***undoneItems* **-=** 1  433 self**.***rear* **+=** 1  434 self**.***rear* **=** self**.***rear***%**self**.***maximum*  435 actionRedone **=** self**.***stack***[**self**.***rear***]**  436 **if** actionRedone **!=** **None:**  437 redoneSprite **=** actionRedone**[**0**]**  438 #Redo component creation  439 **if** actionRedone**[**1**]** **==** 'createdComponent'**:**  440 allSprites**.***add***(**redoneSprite**)**  441 IDSprites**.***add***(**redoneSprite**)**  442 componentSpriteGroup**.***add***(**redoneSprite**)**  443 circuitGraph**.***addNode***(**redoneSprite**.***ID***)**  444 #Redo component movement  445 **elif** actionRedone**[**1**]** **==** 'movedComponent'**:**  446 redoneSprite**.***redo***()**  447 **for** sprite **in** componentSpriteGroup**:**  448 **if** sprite**.***name* **==** 'voltmeter'**:**  449 **if** sprite**.***voltWires***[**'left'**]:**  450 voltConnectedSprite **=** sprite**.***voltWires***[**'left'**].***sprites***[**1**]**  451 **if** voltConnectedSprite **==** actionRedone**[**0**]:**  452 sprite**.***voltWires***[**'left'**].***ends***[**1**]** **=** actionRedone**[**0**].***left*  453 sprite**.***voltWires***[**'right'**].***ends***[**1**]** **=** actionRedone**[**0**].***right*  454 circuitGraph**.***resetVectors***()**  455 #Redo component deletion  456 **elif** actionRedone**[**1**]** **==** 'deletedComponent'**:**  457 connectionReset**(**redoneSprite**)**  458 circuitGraph**.***removeNode***(**redoneSprite**.***ID***)**  459 circuitGraph**.***circuitValid* **=** circuitGraph**.***circuitTest***()**  460 **if** circuitGraph**.***circuitValid* **==** **False:**  461 **for** component **in** componentSpriteGroup**:**  462 **if** component**.***name* **==** 'voltmeter'**:**  463 **if** component**.***voltWires***[**'left'**]:**  464 voltConnectedSprite **=** sprite**.***voltWires***[**'left'**].***sprites***[**1**]**  465 **if** voltConnectedSprite **==** redoneSprite**:**  466 component**.***pd* **=** 0  467 IDSprites**.***remove***(**component**.***voltWires***[**'left'**])**  468 IDSprites**.***remove***(**component**.***voltWires***[**'right'**])**  469 allSprites**.***remove***(**component**.***voltWires***[**'left'**])**  470 allSprites**.***remove***(**component**.***voltWires***[**'right'**])**  471 component**.***voltWires* **=** **{**'left'**:** **None,** 'right'**:** **None}**  472 component**.***reset***()**  473 **for** electron **in** electronSprites**:**  474 **for** spriteGroup **in** allGroups**:**  475 **if** spriteGroup **!=** IDTracker**:**  476 spriteGroup**.***remove***(**electron**)**  477 circuitGraph**.***resetVectors***()**  478 **for** spriteGroup **in** allGroups**:**  479 **if** spriteGroup **!=** IDTracker**:**  480 spriteGroup**.***remove***(**redoneSprite**)**  481 #Redo wire creation  482 **elif** actionRedone**[**1**]** **==** 'createdWire'**:**  483 redoneSprite**.***redoCreation***()**  484 circuitGraph**.***addEdge***(**redoneSprite**.***sprites***[**0**].***ID***,**redoneSprite**.***sprites***[**1**].***ID***)**  485 self**.***action***()**  486  487 #Set component images to red when any action is performed (they become unselected)  488 **def** action**(**self**):**  489 **for** sprite **in** IDSprites**:**  490 **if** sprite**.***image* **==** 'red'**:**  491 sprite**.***image* **=** 'base'  492 sprite**.***surf* **=** pygame**.***image***.***load***(**f'graphics/{sprite**.***name*}Base.png'**)**  493  494 ###Class for circuit graph data structure  495 **class** **Graph():**  496 #Constructor  497 **def** \_\_init\_\_**(**self**):**  498 self**.***graph* **=** **{}**  499 self**.***nodeNumber* **=** 0  500 self**.***circuitValid* **=** **False**  501 self**.***vectorsMade* **=** **False**  502 self**.***vectorsCycle* **=** **[]**  503 self**.***emfSign* **=** **None**  504 self**.***currentMultiplier* **=** **None**  505 #Reset graph (reset button pressed)  506 **def** reset**(**self**):**  507 self**.***graph* **=** **{}**  508 self**.***nodeNumber* **=** 0  509 self**.***circuitValid* **=** **False**  510 self**.***vectorsMade* **=** **False**  511 self**.***vectorsCycle* **=** **[]**  512 self**.***emfSign* **=** **None**  513 self**.***currentMultiplier* **=** **None**  514  515 #Add node  516 **def** addNode**(**self**,**node**):**  517 **if** node **in** self**.***graph***:**  518 **pass**  519 **else:**  520 self**.***graph***[**node**]** **=** **[]**  521 self**.***nodeNumber* **+=** 1  522 #Test if circuit is valid  523 self**.***circuitValid* **=** self**.***circuitTest***()**  524 self**.***action***()**  525  526 #Add edge  527 **def** addEdge**(**self**,**node1**,**node2**):**  528 **if** node1 **in** self**.***graph* **and** node2 **in** self**.***graph***:**  529 self**.***graph***[**node1**].***append***(**node2**)**  530 self**.***graph***[**node2**].***append***(**node1**)**  531 #Test if circuit is valid  532 self**.***circuitValid* **=** self**.***circuitTest***()**  533 self**.***action***()**  534  535 #Remove node  536 **def** removeNode**(**self**,**node**):**  537 **for** node2 **in** self**.***graph***[**node**]:**  538 self**.***graph***[**node2**].***remove***(**node**)**  539 #Test if circuit is valid  540 self**.***circuitValid* **=** self**.***circuitTest***()**  541 self**.***graph***.***pop***(**node**)**  542 self**.***nodeNumber* **-=** 1  543 self**.***action***()**  544  545 #Remove edge  546 **def** removeEdge**(**self**,**wireSprite**):**  547 spriteIDs **=** **[]**  548 **for** sprite **in** wireSprite**.***sprites***:**  549 spriteIDs**.***append***(**sprite**.***ID***)**  550 **for** node1 **in** self**.***graph***:**  551 **if** node1 **in** spriteIDs**:**  552 **for** node2 **in** self**.***graph***[**node1**]:**  553 **if** node2 **in** spriteIDs**:**  554 self**.***graph***[**node1**].***remove***(**node2**)**  555 #Test if circuit is valid  556 self**.***circuitValid* **=** self**.***circuitTest***()**  557 self**.***action***()**  558  559 ########Taken from online source and adjusted \*START\* ########  560 # A recursive function that uses  561 # visited[] and parent to detect  562 # cycle in subgraph reachable from node v.  563 **def** isCyclicUtil**(**self**,** v**,** visited**,** parent**):**  564 # Mark the current node as visited  565 visited**[**v**]** **=** **True**  566 # Recur for all the vertices  567 # adjacent to this node  568 **for** i **in** self**.***graph***[**v**]:**  569 # If the node is not  570 # visited then recurse on it  571 **if** visited**[**i**]** **==** **False:**  572 **if(**self**.***isCyclicUtil***(**i**,** visited**,** v**)):**  573 **return** **True**  574 # If an adjacent node is  575 # visited and not parent  576 # of current node,  577 # then there is a cycle  578 **elif** parent **!=** i**:**  579 **return** **True**  580 **return** **False**  581  582 # Returns true if the graph  583 # contains a cycle, else false.  584 **def** isCyclic**(**self**):**  585 # Mark all the vertices  586 # as not visited  587 visited **=** **{}**  588 **for** ID **in** self**.***graph***:**  589 visited**[**ID**]** **=** **False**  590 # Call the recursive helper  591 # function to detect cycle in different  592 # DFS trees  593 #for i in range(self.nodeNumber):  594 **for** node **in** self**.***graph***:**  595 # Don't recur for u if it  596 # is already visited  597 **if** visited**[**node**]** **==** **False:**  598 **if(**self**.***isCyclicUtil***(**node**,** visited**,** **-**1**))** **==** **True:**  599 **return** **True**  600 **return** **False**  601 ##### \*END OF MODIFIED SOURCED CODE\* #####  602  603 #Test circuit for conditions  604 **def** circuitTest**(**self**):**  605 #Test if a cell is present and connected  606 cellFound **=** **False**  607 **for** sprite **in** componentSpriteGroup**:**  608 **if** sprite**.***name* **==** 'cell' **and** sprite**.***ID* **in** self**.***graph***:**  609 numberOfConnections **=** 0  610 **for** key **in** sprite**.***connected***:**  611 **if** sprite**.***connected***[**key**]** **!=** **None:**  612 numberOfConnections **+=** 1  613 **if** numberOfConnections **==** 2**:**  614 cellFound **=** **True**  615 #Test if the circuit is cycle is a cell is found  616 **if** cellFound **==** **True:**  617 **if** self**.***isCyclic***()** **!=** **True:**  618 **return** **False**  619 **else:**  620 **return** **True**  621 **else:**  622 **return** **False**  623  624 #Calculate physics whenever an action is performed and the circuit is valid  625 **def** action**(**self**):**  626 **if** circuitGraph**.***circuitValid* **==** **True:**  627 physicsCalc**()**  628  629 #Create vectors for each component and wire and set electron initial positions  630 **def** makeVectors**(**self**):**  631 #Recursive iterative subprogram to create and store each vector  632 **def** vectorIteration**(**sprite1**,**startType**,**activeSprites**,**run**,**skip**):**  633 run **+=** 1  634 **if** sprite1 **!=** activeSprites**[**0**]** **or** run **==** 1**:**  635 #Find the start and end connectors (left/right) of the wire, and create the sprite vector  636 **if** startType **==** 'left'**:**  637 endType **=** 'right'  638 **if** skip **==** **False:**  639 spriteVector **=** **[**sprite1**,**'left'**,**'right'**,**Vector**(**sprite1**.***left***,**sprite1**.***right***)]**  640 **else:**  641 spriteVector **=** **None**  642 **else:**  643 endType **=** 'left'  644 **if** skip **==** **False:**  645 spriteVector **=** **[**sprite1**,**'right'**,**'left'**,**Vector**(**sprite1**.***right***,**sprite1**.***left***)]**  646 **else:**  647 spriteVector **=** **None**  648 wire **=** sprite1**.***connected***[**endType**]**  649 sprite2 **=** **None**  650 **for** sprite **in** wire**.***sprites***:**  651 **if** sprite **!=** sprite1**:**  652 sprite2 **=** sprite  653 skip **=** **False**  654 #Find the corresponding sprites and connectors for the wire  655 sprite1Index **=** wire**.***sprites***.***index***(**sprite1**)**  656 sprite2Index **=** wire**.***sprites***.***index***(**sprite2**)**  657 sprite1Type **=** wire**.***spriteConnectors***[**sprite1Index**]**  658 sprite2Type **=** wire**.***spriteConnectors***[**sprite2Index**]**  659 pos1 **=** wire**.***ends***[**sprite1Index**]**  660 pos2 **=** wire**.***ends***[**sprite2Index**]**  661 #Create wire vector  662 **if** sprite1**.***name* **==** 'junc' **and** sprite2**.***name* **==** 'junc'**:**  663 wireVector **=** **[**'vector'**,**'center'**,**'center'**,**Vector**(**sprite1**.***currentPosition***,**sprite2**.***currentPosition***)]**  664 skip **=** **True**  665 **elif** sprite1**.***name* **==** 'junc'**:**  666 wireVector **=** **[**'vector'**,**'center'**,**sprite2Type**,**Vector**(**sprite1**.***currentPosition***,**pos2**)]**  667 **elif** sprite2**.***name* **==** 'junc'**:**  668 wireVector **=** **[**'vector'**,**sprite1Type**,**'center'**,**Vector**(**pos1**,**sprite2**.***currentPosition***)]**  669 skip **=** **True**  670 **else:**  671 wireVector **=** **[**'vector'**,**sprite1Type**,**sprite2Type**,**Vector**(**pos1**,**pos2**)]**  672 #Add created vectors to the vectorsCycle attribute and call a new iteration  673 **if** spriteVector **!=** **None:**  674 self**.***vectorsCycle***.***append***(**spriteVector**)**  675 self**.***vectorsCycle***.***append***(**wireVector**)**  676 vectorIteration**(**sprite2**,**sprite2Type**,**activeSprites**,**run**,**skip**)**  677  678 activeSprites **=** **[]**  679 #Find active sprites  680 **for** sprite **in** componentSpriteGroup**:**  681 **if** sprite**.***ID* **in** self**.***graph***:**  682 numberOfConnections **=** 0  683 **for** key **in** sprite**.***connected***:**  684 **if** sprite**.***connected***[**key**]** **!=** **None:**  685 numberOfConnections **+=** 1  686 **if** numberOfConnections **==** 2**:**  687 activeSprites**.***append***(**sprite**)**  688 #Initial iteration call  689 vectorIteration**(**activeSprites**[**0**],**'left'**,**activeSprites**,**0**,False)**  690 self**.***vectorsMade* **=** **True**  691 #Find the total magnitude of each vector  692 self**.***totalMag* **=** 0  693 **for** item **in** self**.***vectorsCycle***:**  694 self**.***totalMag* **+=** item**[**3**].***mag*  695 #Find the number of electrons to create  696 self**.***electronNumber* **=** self**.***totalMag***//**50  697 #Find the exact spacing each electron should be roughly placed at (~50px)  698 self**.***electronSpacing* **=** self**.***totalMag***/**self**.***electronNumber*  699 **if** **len(**electronSprites**)** **>** 0**:**  700 **for** spriteGroup **in** allGroups**:**  701 **if** spriteGroup **!=** IDTracker**:**  702 **for** electron **in** electronSprites**:**  703 spriteGroup**.***remove***(**electron**)**  704 #Temporary position to assign to electrons  705 tempPos **=** **(**self**.***vectorsCycle***[**0**][**3**].***pos1***)**  706 vectIndex **=** 0  707 tempPosx **=** tempPos**[**0**]**  708 tempPosy **=** tempPos**[**1**]**  709 #Iterate though a loop and create each electron  710 **for** i **in** **range(**0**,int(**self**.***electronNumber***)):**  711 Electron**(**numberOfSprites**(),**tempPos**,**vectIndex**)**  712 xleft **=** **abs(**self**.***vectorsCycle***[**vectIndex**][**3**].***pos2***[**0**]** **-** tempPosx**)**  713 yleft **=** **abs(**self**.***vectorsCycle***[**vectIndex**][**3**].***pos2***[**1**]** **-** tempPosy**)**  714 magLeft **=** **(**xleft**\*\***2 **+** yleft**\*\***2**)\*\***0.5  715 **if** magLeft **>** self**.***electronSpacing***:**  716 # +(electronSpacing\*moveBy)  717 tempPosx **+=** self**.***electronSpacing* **\*** self**.***vectorsCycle***[**vectIndex**][**3**].***moveBy***[**0**]**  718 tempPosy **+=** self**.***electronSpacing* **\*** self**.***vectorsCycle***[**vectIndex**][**3**].***moveBy***[**1**]**  719 **else:**  720 #Finds how far past the end of a vector the next position will be by  721 # finding the magnitude left and takes it from the electronSpacing  722 tempPosx **=** self**.***vectorsCycle***[**vectIndex**][**3**].***pos2***[**0**]**  723 tempPosy **=** self**.***vectorsCycle***[**vectIndex**][**3**].***pos2***[**1**]**  724 vectIndex **+=** 1  725 **if** vectIndex **==** **len(**self**.***vectorsCycle***):**  726 vectIndex **=** 0  727 magOverflow **=** self**.***electronSpacing* **-** magLeft  728 tempPosx **+=** magOverflow **\*** self**.***vectorsCycle***[**vectIndex**][**3**].***moveBy***[**0**]**  729 tempPosy **+=** magOverflow **\*** self**.***vectorsCycle***[**vectIndex**][**3**].***moveBy***[**1**]**  730 tempPos **=** **(**tempPosx**,**tempPosy**)**  731  732 #Reset vectors when circuit is broken  733 **def** resetVectors**(**self**):**  734 self**.***vectorsMade* **=** **False**  735 self**.***vectorsCycle* **=** **[]**  736  737 ###Class for the Unit Graph to show unit relationships  738 **class** **UnitGraph():**  739 #Constructor  740 **def** \_\_init\_\_**(**self**,**unit1**,**unit2**):**  741 self**.***unit1* **=** unit1  742 self**.***unit2* **=** unit2  743 self**.***units* **=** **{**unit1**:** **None,**unit2**:** **None}**  744 self**.***unitList* **=** **[**unit1**,**unit2**]**  745 self**.***xmax* **=** **None**  746 self**.***ymax* **=** **None**  747 self**.***sprite* **=** **None**  748 self**.***lines* **=** **[]**  749 #Assign each unit a value that it is being measured in  750 **for** unit **in** self**.***units***:**  751 **if** unit **==** 'V'**:**  752 self**.***units***[**unit**]** **=** 'V'  753 **elif** unit **==** 'I'**:**  754 self**.***units***[**unit**]** **=** 'A'  755 **elif** unit **==** 'R'**:**  756 self**.***units***[**unit**]** **=** 'Ω'  757 **elif** unit **==** 'Q'**:**  758 self**.***units***[**unit**]** **=** 'C'  759 **elif** unit **==** 'LI'**:**  760 self**.***units***[**unit**]** **=** 'Lux'  761 **elif** unit **==** 'T'**:**  762 self**.***units***[**unit**]** **=** '°C'  763 self**.***axies* **=** **{**'unit1'**:** **[**unit1**,**self**.***units***[**unit1**]],**  764 'unit2'**:** **[**unit2**,**self**.***units***[**unit2**]]}**  765 #Add to unitGraphGroup to allow access through tkinter and pygame threads  766 **if** **len(**unitGraphGroup**)** **>** 0**:**  767 unitGraphGroup**.***pop***(**0**)**  768 unitGraphGroup**.***append***(**self**)**  769  770 #Draw N/A for non-applicable relationships for certain components  771 **def** plotNA**(**self**,**screen**,**font**):**  772 surf **=** font**.***render***(**'N/A'**,True,**'#ff3344'**)**  773 rect **=** surf**.***get\_rect***(**center **=** **(**874**,**407**))**  774 screen**.***blit***(**surf**,**rect**)**  775  776 #Draw a linear relationship line  777 **def** plotLinear**(**self**,**screen**):**  778 pygame**.***draw***.***line***(**screen**,**'#ff3333'**,(**787**,**468**),(**960**,**360**),**width**=**3**)**  779  780 #Draw the diode I-V relationship line  781 **def** plotDiodeV**(**self**,**screen**):**  782 pygame**.***draw***.***line***(**screen**,**'#ff3333'**,(**787**,**468**),(**800**,**468**),**width**=**3**)**  783 pygame**.***draw***.***arc***(**screen**,**'#ff3333'**,[**775**,**270**,**50**,**200**],** 4.75**,**5.25**,**width**=**3**)**  784 pygame**.***draw***.***line***(**screen**,**'#ff3333'**,(**810**,**458**),(**865**,**350**),**width**=**3**)**  785  786 #Draw the lamp/thermistor I-V relationship line  787 **def** plotLampThermV**(**self**,**screen**):**  788 pygame**.***draw***.***arc***(**screen**,**'#ff3333'**,[**783**,**368**,**437**,**240**],** 1.75**,**3**,** width**=**3**)**  789  790 #Draw the ldr/thermistor relationship line for light intensity/temperature  791 **def** plotTempLi**(**self**,**screen**):**  792 pygame**.***draw***.***arc***(**screen**,**'#ff3333'**,[**795**,**202**,**300**,**260**],** 3.25**,**4.75**,** width**=**3**)**  793  794 ###Class for electrons  795 **class** **Electron(**pygame**.***sprite***.***Sprite***):**  796 #Constructor  797 **def** \_\_init\_\_**(**self**,**ID**,**initialPos**,**vectorIndex**):**  798 pygame**.***sprite***.***Sprite***.***\_\_init\_\_***(**self**)**  799 self**.***name* **=** 'electron'  800 self**.***ID* **=** ID  801 self**.***currentPos* **=** initialPos  802 self**.***image* **=** 'blue'  803 self**.***surf* **=** pygame**.***image***.***load***(**'graphics/electron.png'**)**  804 self**.***rect* **=** self**.***surf***.***get\_rect***(**center **=** initialPos**)**  805 self**.***shown* **=** **False**  806 #Current vector in the graph's stored vectors that electron is following  807 self**.***vectorIndex* **=** vectorIndex  808 IDSprites**.***add***(**self**)**  809 electronSprites**.***add***(**self**)**  810 allSprites**.***add***(**self**)**  811 IDTracker**.***add***(**self**)**  812  813 #Toggle whether the electron is visible on the screen  814 **def** toggleShow**(**self**):**  815 **if** self**.***shown* **==** **False:**  816 self**.***shown* **=** **True**  817 **else:**  818 self**.***shown* **=** **False**  819  820 ###Class for vectors  821 **class** **Vector():**  822 #Constructor  823 **def** \_\_init\_\_**(**self**,**pos1**,**pos2**):**  824 self**.***pos1* **=** pos1  825 self**.***pos2* **=** pos2  826 self**.***xdiff* **=** pos2**[**0**]-**pos1**[**0**]**  827 self**.***ydiff* **=** pos2**[**1**]-**pos1**[**1**]**  828 #Total translation vector  829 self**.***vect* **=** **[**self**.***xdiff***,**self**.***ydiff***]**  830 self**.***mag* **=** **(**self**.***xdiff***\*\***2 **+** self**.***ydiff***\*\***2**)\*\***0.5  831 #Relative translation vector for small fraction of the total magnitude  832 self**.***moveBy* **=** **[(**self**.***xdiff***/**self**.***mag***),(**self**.***ydiff***/**self**.***mag***)]**  833 **if** self**.***xdiff* **==** 0**:**  834 #If x=0, set infinite gradient  835 **if** self**.***ydiff* **!=** 0**:**  836 self**.***gradient* **=** 'inf'  837 self**.***xchange* **=** 0  838 self**.***ychange* **=** self**.***ydiff*  839 **else:**  840 #If x!=0 and y!=0, set normal gradient  841 **if** self**.***ydiff* **!=** 0**:**  842 self**.***gradient* **=** self**.***ydiff***/**self**.***xdiff*  843 self**.***xchange* **=** self**.***gradient*  844 self**.***ychange* **=** 1  845 #If x!=0 and y=0, set gradient to 0  846 **else:**  847 self**.***gradient* **=** 0  848 self**.***xchange* **=** self**.***xdiff*  849 self**.***ychange* **=** 0  850 ###Subprogram to find the nearest connector to the mouse  851 **def** findNearestConnector**(**wireOrigin**):**  852 #Finds and returns the magnitude of the line between the mouse pointer  853 # and a connector, given that it is less than 75px  854 **def** getMagnitude**(**pos1**,**pos2**):**  855 magnitude **=** **((**pos1**[**0**]-**pos2**[**0**])\*\***2 **+** **(**pos1**[**1**]-**pos2**[**1**])\*\***2**)\*\***0.5  856 **if** magnitude **<=** 75**:**  857 **return** magnitude  858 **else:**  859 **return** **None**  860 inRangeMagnitude **=** **[]**  861 **for** sprite **in** componentSpriteGroup**:**  862 **if** sprite**.***name* **!=** 'voltmeter'**:**  863 #If sprite is not picked up and is not the same sprite the wire was started from  864 **if** sprite**.***pickupStatus* **==** **False** **and** sprite **!=** wireOrigin**:**  865 left **=** sprite**.***left*  866 right **=** sprite**.***right*  867 top **=** sprite**.***top*  868 bottom **=** sprite**.***bottom*  869 leftDist **=** getMagnitude**(**pygame**.***mouse***.***get\_pos***(),**left**)**  870 rightDist **=** getMagnitude**(**pygame**.***mouse***.***get\_pos***(),**right**)**  871 topDist **=** getMagnitude**(**pygame**.***mouse***.***get\_pos***(),**top**)**  872 bottomDist **=** getMagnitude**(**pygame**.***mouse***.***get\_pos***(),**bottom**)**  873 #Checks the connector is not already connected  874 **if** leftDist **!=** **None** **and** sprite**.***connected***[**'left'**]** **==** **None:**  875 inRangeMagnitude**.***append***([**sprite**,**'left'**,**leftDist**])**  876 **if** rightDist **!=** **None** **and** sprite**.***connected***[**'right'**]** **==** **None:**  877 inRangeMagnitude**.***append***([**sprite**,**'right'**,**rightDist**])**  878 **if** topDist **!=** **None** **and** sprite**.***connected***[**'top'**]** **==** **None:**  879 inRangeMagnitude**.***append***([**sprite**,**'top'**,**topDist**])**  880 **if** bottomDist **!=** **None** **and** sprite**.***connected***[**'bottom'**]** **==** **None:**  881 #Add any connectors found less than 75px away to a list  882 inRangeMagnitude**.***append***([**sprite**,**'bottom'**,**bottomDist**])**  883 #Iterate through connectors found to find the shortest distance  884 **if** **len(**inRangeMagnitude**)** **>** 0**:**  885 lowestFound **=** 100000000  886 **for** item **in** inRangeMagnitude**:**  887 **if** item**[**2**]** **<** lowestFound**:**  888 lowestFound **=** item**[**2**]**  889 lowestVector **=** item  890 #If lowest vector is not the same point the wire origionated from, add the proxCircle to be blitted  891 **if** proxCircle **not** **in** allSprites **and** lowestVector**[**0**]** **!=** wireOrigin**:**  892 allSprites**.***add***(**proxCircle**)**  893 **return** lowestVector  894 #If no connectors are in range remove the proxCircle from the screen  895 **else:**  896 **if** proxCircle **in** allSprites**:**  897 allSprites**.***remove***(**proxCircle**)**  898 proxCircle**.***pos* **=** **(**0**,**0**)**  899 **return** **None**  900  901 ###Subprogram for reseting a component's connections  902 **def** connectionReset**(**sprite**):**  903 #Check if any wires are connected to a sprite, and if so, set them to None.  904 **for** key **in** sprite**.***connected***:**  905 **if** sprite**.***connected***[**key**]** **!=** **None:**  906 wire **=** sprite**.***connected***[**key**]**  907 **for** wireSprite **in** wire**.***sprites***:**  908 **if** wireSprite **!=** sprite**:**  909 **for** otherKey **in** wireSprite**.***connected***:**  910 **if** wireSprite**.***connected***[**otherKey**]** **!=** **None:**  911 **if** wireSprite**.***connected***[**otherKey**]** **==** wire**:**  912 wireSprite**.***connected***[**otherKey**]** **=** **None**  913 #Undo creation of the wire to delete it from the screen and reset its values  914 wire**.***undoCreation***(**sprite**)**  915  916 ###Subprogram to assign each sprite an ID  917 **def** numberOfSprites**():**  918 #Gets the number of sprites which have existed during run time and assigns the sprite  919 # the next number as an ID  920 newID **=** **len(**IDTracker**)**  921 newID **+=** 1  922 **return** newID  923  924 ###Subprogram for calculating physics circuit values  925 **def** physicsCalc**():**  926 #Only calculate if circuit is valid  927 **if** circuitGraph**.***circuitValid* **==** **True:**  928 activeSprites **=** **[]**  929 current **=** 0  930 emf **=** 0  931 **for** sprite **in** componentSpriteGroup**:**  932 **if** sprite**.***ID* **in** circuitGraph**.***graph***:**  933 numberOfConnections **=** 0  934 **for** key **in** sprite**.***connected***:**  935 **if** sprite**.***connected***[**key**]** **!=** **None:**  936 numberOfConnections **+=** 1  937 **if** numberOfConnections **==** 2**:**  938 activeSprites**.***append***(**sprite**)**  939 #Find total emf before calculating current and voltage, and find the resistance  940 # of any LDRs and thermistors  941 **for** sprite **in** activeSprites**:**  942 #Add to emf if cell to give total emf  943 **if** sprite**.***name* **==** 'cell'**:**  944 emf **+=** sprite**.***emf*  945 #Calculate resistance of thermistor given its temperature  946 **elif** sprite**.***name* **==** 'thermistor'**:**  947 e **=** math**.***exp***(**1**)**  948 sprite**.***resistance* **=** **(**sprite**.***resistance***)\*(**e**\*\*(**3000**\*((**1**/(**sprite**.***temp***+**273**))** **-** **(**1**/(**sprite**.***prevTemp***+**273**)))))**  949 #Calculate resistance of ldr given its light intensity  950 **elif** sprite**.***name* **==** 'ldr'**:**  951 e **=** math**.***exp***(**1**)**  952 sprite**.***resistance* **=** **(**sprite**.***resistance***)\*(**e**\*\*(**5000**\*((**1**/**sprite**.***LI***)** **-** **(**1**/**sprite**.***prevLI***))))**  953 #Find the emf direction multiplier to determine the direction of electrons  954 **if** **-**0.01 **<** emf **<** 0.01**:**  955 circuitGraph**.***emfSign* **=** 0  956 **elif** emf **>=** 0.01**:**  957 circuitGraph**.***emfSign* **=** 1  958 **elif** emf **<=** **-**0.01**:**  959 circuitGraph**.***emfSign* **=** **-**1  960 #Find the total resistance of any circuit components  961 totalRes **=** 0  962 **for** comp **in** activeSprites**:**  963 **if** 'resistance' **in** comp**.***allAttributes***():**  964 **if** comp**.***resistance***:**  965 totalRes **+=** comp**.***resistance*  966 **if** totalRes **!=** 0**:**  967 #Calculate current with I=V/R  968 current **=** emf**/**totalRes  969 nonPwrSprites **=** **[**'ammeter'**,**'voltmeter'**,**'junc'**]**  970 **for** sprite **in** activeSprites**:**  971 sprite**.***current* **=** current  972 **if** 'resistance' **in** sprite**.***allAttributes***():**  973 #If cell has an internal resistance, divide emf into an equal fraction of its resistance  974 # compared to the total resistance to give the "lost volts" in the cell  975 **if** sprite**.***name* **==** 'cell'**:**  976 sprite**.***lostV* **=** emf**\*(**sprite**.***resistance***/**totalRes**)**  977 #Calulate the voltage for each component by dividing the emf into an equal fraction  978 # of its resistance compared to the total resistance  979 **else:**  980 sprite**.***pd* **=** emf**\*(**sprite**.***resistance***/**totalRes**)**  981 #Calculate power of applicable components using P=IV  982 **if** sprite**.***name* **not** **in** nonPwrSprites**:**  983 **if** sprite**.***name* **==** 'cell'**:**  984 sprite**.***power* **=** current**\***sprite**.***emf*  985 **else:**  986 sprite**.***power* **=** current**\***sprite**.***pd*  987 #If the diode voltage is below the threshhold (minimum) voltage, set current to 0  988 **if** sprite**.***name* **==** 'diode'**:**  989 **if** sprite**.***pd* **<** sprite**.***minThresh***:**  990 current **=** 0  991 **for** sprite **in** activeSprites**:**  992 sprite**.***current* **=** 0  993 #Calculate the currentMultiplier for speed of moving electrons  994 **if** **abs(**current**\***0.5**)** **<=** 30**/**9**:**  995 circuitGraph**.***currentMultiplier* **=** **abs(**current**\***0.5**)**  996 **else:**  997 circuitGraph**.***currentMultiplier* **=** **abs(**30**/**9**)**  998 #Set voltage of any connected voltmeters to their corresponding component  999 **for** sprite **in** componentSpriteGroup**:**  1000 **if** sprite**.***name* **==** 'voltmeter'**:**  1001 **if** sprite**.***voltWires***[**'left'**]:**  1002 voltConnectedSprite **=** sprite**.***voltWires***[**'left'**].***sprites***[**1**]**  1003 **if** voltConnectedSprite**.***name* **==** 'cell'**:**  1004 sprite**.***pd* **=** voltConnectedSprite**.***emf*  1005 **else:**  1006 sprite**.***pd* **=** voltConnectedSprite**.***pd*  1007 #If circuit has 0 resistance, set current to infinity and other circuit values to 0.  1008 **else:**  1009 current **=** math**.***inf*  1010 circuitGraph**.***currentMultiplier* **=** **abs(**30**/**9**)**  1011 **for** sprite **in** activeSprites**:**  1012 sprite**.***current* **=** math**.***inf*  1013 sprite**.***pd* **=** 0  1014 sprite**.***power* **=** 0  1015 **for** sprite **in** componentSpriteGroup**:**  1016 **if** sprite**.***name* **==** 'voltmeter'**:**  1017 sprite**.***pd* **=** 0  1018  1019 ###Subprogram for drawing the selected component's circuit values  1020 **def** drawStatsText**(**screen**,**selectedSprite**,**font**):**  1021 **if** selectedSprite**:**  1022 **if** selectedSprite**.***image* **==** 'red' **and** selectedSprite **in** allSprites**:**  1023 atts **=** selectedSprite**.***allAttributes***()**  1024 showableTexts **=** **[**'current'**,**'pd'**,**'emf'**,**'resistance'**,**'temp'**,**'LI'**,**'minThresh'**,**'power'**,**'lostV'**]**  1025 **if** selectedSprite**.***name* **==** 'voltmeter' **and** 'current' **in** atts**:**  1026 **del(**atts**[**'current'**])**  1027 yPos **=** 60  1028 #Set text values to blit and their unit's symbol  1029 **for** textName **in** showableTexts**:**  1030 **if** textName **in** atts**:**  1031 **if** textName **==** 'emf'**:**  1032 text **=** 'e.m.f'  1033 symbol **=** 'V'  1034 **elif** textName **==** 'pd'**:**  1035 text **=** 'Voltage'  1036 symbol **=** 'V'  1037 **elif** textName **==** 'current'**:**  1038 text **=** 'Current'  1039 symbol **=** 'A'  1040 **elif** textName **==** 'resistance'**:**  1041 text **=** 'Resistance'  1042 symbol **=** 'Ω'  1043 **elif** textName **==** 'temp'**:**  1044 text **=** 'Temperature'  1045 symbol **=** '°C'  1046 **elif** textName **==** 'LI'**:**  1047 text **=** 'Light Int'  1048 symbol **=** ' Lux'  1049 **elif** textName **==** 'minThresh'**:**  1050 text **=** 'Min Voltage'  1051 symbol **=** 'V'  1052 **elif** textName **==** 'power'**:**  1053 text **=** 'Power'  1054 symbol **=** 'W'  1055 **elif** textName **==** 'lostV'**:**  1056 text **=** '"Lost Volts"'  1057 symbol **=** 'V'  1058 #Set the value of the attribute  1059 **if** atts**[**textName**]:**  1060 value **=** **round(**atts**[**textName**],** 2**)**  1061 **else:**  1062 value **=** 0  1063 #Show text and draw a red box around the components customisable value  1064 surf **=** font**.***render***(**f'{text}: {value}{symbol}'**,True,**'#333333'**)**  1065 rect **=** surf**.***get\_rect***(**midleft **=** **(**760**,**yPos**))**  1066 screen**.***blit***(**surf**,**rect**)**  1067 **if** 'customVal' **in** selectedSprite**.***allAttributes***():**  1068 **if** textName **==** selectedSprite**.***customVal***:**  1069 pygame**.***draw***.***rect***(**screen**,**'#ff2222'**,(**755**,**yPos**-**15**,**242**,**32**),**width**=**1**)**  1070 #Add 30px to the y position text is shown at ready for the next iteration  1071 yPos **+=** 30  1072  1073 ###Subprogram for resetting the circuit  1074 **def** resetCircuitBoard**():**  1075 #Clear sprite groups  1076 **for** sprite **in** allSprites**:**  1077 allSprites**.***remove***(**sprite**)**  1078 **if** sprite **in** IDSprites**:**  1079 IDSprites**.***remove***(**sprite**)**  1080 **if** sprite **in** componentSpriteGroup**:**  1081 componentSpriteGroup**.***remove***(**sprite**)**  1082 #Reset Undo/Redo system  1083 actions**.***reset***()**  1084 #Reset circuit graph  1085 circuitGraph**.***reset***()**  1086  1087 #Set up actions stack  1088 actions **=** Cstack**(**'actions'**,**10**)**  1089 #Set up graph data structure  1090 circuitGraph **=** Graph**()**  1091 #Set up component circuit values/attributes and set the  1092 # initial customisable value for the component type  1093 cellAttributes **=** **{**'current'**:** **None,**  1094 'emf'**:** 5**,**  1095 'resistance'**:** 0**,**  1096 'power'**:** **None,**  1097 'lostV'**:** 0**,**  1098 'customVal'**:** 'emf'**}**  1099 lampAttributes **=** **{**'current'**:** **None,**  1100 'pd'**:** **None,**  1101 'resistance'**:** 5**,**  1102 'power'**:** **None,**  1103 'customVal'**:** 'resistance'**}**  1104 resistorAttributes **=** **{**'current'**:** **None,**  1105 'pd'**:** **None,**  1106 'resistance'**:** 10**,**  1107 'power'**:** **None,**  1108 'customVal'**:** 'resistance'**}**  1109 juncAttributes **=** **{**'current'**:** **None}**  1110 ammeterAttributes **=** **{**'current'**:** **None,}**  1111 voltmeterAttributes **=** **{**'pd'**:** **None}**  1112 thermistorAttributes **=** **{**'current'**:** **None,**  1113 'pd'**:** **None,**  1114 'resistance'**:** 10**,**  1115 'temp'**:** 21**,**  1116 'power'**:** **None,**  1117 'customVal'**:** 'temp'**}**  1118 ldrAttributes **=** **{**'current'**:** **None,**  1119 'pd'**:** **None,**  1120 'resistance'**:** 10**,**  1121 'LI'**:** 500**,**  1122 'power'**:** **None,**  1123 'customVal'**:** 'LI'**}**  1124 diodeAttributes **=** **{**'current'**:** **None,**  1125 'pd'**:** **None,**  1126 'resistance'**:** 10**,**  1127 'minThresh'**:** 1**,**  1128 'power'**:** **None,**  1129 'customVal'**:** 'resistance'**}**  1130 #Unit Graph Group (stores unit graph to be access by both the tkinter and pygame threads  1131 unitGraphGroup **=** **[]**  1132 #Set up sprite groups  1133 componentSpriteGroup **=** pygame**.***sprite***.***Group***()**  1134 allSprites **=** pygame**.***sprite***.***Group***()**  1135 IDSprites **=** pygame**.***sprite***.***Group***()**  1136 electronSprites **=** pygame**.***sprite***.***Group***()**  1137 IDTracker **=** pygame**.***sprite***.***Group***()**  1138 allGroups **=** **[**allSprites**,**IDSprites**,**componentSpriteGroup**,**electronSprites**,**IDTracker**]**  1139 #Set up constant images  1140 proxCircle **=** ProxCircle**(**'proxCircle'**,**0**,(**0**,**0**))**  1141 #Set selectedSprite to None to allow access between both the tkinter and pygame threads  1142 selectedSprite **=** **None**  1143  1144 #####Tkinter Thread  1145 **def** tkBoxRun**():**  1146 ###Class for the tkinter window  1147 **class** **tkWindow():**  1148 #Constructor  1149 **def** \_\_init\_\_**(**self**,**windowName**):**  1150 self**.***window* **=** **None**  1151 self**.***windowName* **=** windowName  1152 self**.***root* **=** tk**.***Tk***()**  1153 self**.***root***.***title***(**'Components'**)**  1154 self**.***embed* **=** Frame**(**self**.***root***,** width**=**640**,** height**=**480**)**  1155 os**.***environ***[**'SDL\_WINDOWID'**]** **=** **str(**self**.***embed***.***winfo\_id***())**  1156 os**.***environ***[**'SDL\_VIDEODRIVER'**]** **=** 'windib'  1157 **if** self**.***windowName* **==** 'boxWin'**:**  1158 self**.***root***.***geometry***(**'275x600+110+120'**)**  1159 #Set component images  1160 self**.***imageCell* **=** PhotoImage**(**file**=**'graphics/cellIcon.png'**)**  1161 self**.***imageLamp* **=** PhotoImage**(**file**=**'graphics/lampIcon.png'**)**  1162 self**.***imageResistor* **=** PhotoImage**(**file**=**'graphics/resistorIcon.png'**)**  1163 self**.***imageJunc* **=** PhotoImage**(**file**=**'graphics/juncIcon.png'**)**  1164 self**.***imageAmmeter* **=** PhotoImage**(**file**=**'graphics/ammeterIcon.png'**)**  1165 self**.***imageVoltmeter* **=** PhotoImage**(**file**=**'graphics/voltmeterIcon.png'**)**  1166 self**.***imageThermistor* **=** PhotoImage**(**file**=**'graphics/thermistorIcon.png'**)**  1167 self**.***imageLDR* **=** PhotoImage**(**file**=**'graphics/ldrIcon.png'**)**  1168 self**.***imageDiode* **=** PhotoImage**(**file**=**'graphics/diodeIcon.png'**)**  1169 self**.***borderColor* **=** tk**.***Frame***(**self**.***root***,**background**=**'red'**)**  1170 #Create and set component buttons into a grid  1171 self**.***cellButton* **=** tk**.***Button***(**self**.***root***,** text**=**'Cell'**,** image**=**self**.***imageCell***,** command**=**self**.***cellCommand***,**relief**=**GROOVE**)**  1172 self**.***cellButton***.***grid***(**row**=**0**,**column**=**0**,**padx**=**2.5**)**  1173 self**.***lampButton* **=** tk**.***Button***(**self**.***root***,** text**=**'Lamp'**,** image**=**self**.***imageLamp***,** command**=**self**.***lampCommand***,**relief**=**GROOVE**)**  1174 self**.***lampButton***.***grid***(**row**=**1**,**column**=**0**,**padx**=**2.5**)**  1175 self**.***resistorButton* **=** tk**.***Button***(**self**.***root***,** text**=**'Resistor'**,** image**=**self**.***imageResistor***,** command**=**self**.***resistorCommand***,**relief**=**RIDGE**)**  1176 self**.***resistorButton***.***grid***(**row**=**0**,**column**=**2**,**padx**=**2.5**)**  1177 self**.***juncButton* **=** tk**.***Button***(**self**.***root***,** text**=**'Junc'**,** image**=**self**.***imageJunc***,** command**=**self**.***juncCommand***,**relief**=**RIDGE**)**  1178 self**.***juncButton***.***grid***(**row**=**0**,**column**=**1**,**padx**=**2.5**)**  1179 self**.***ammeterButton* **=** tk**.***Button***(**self**.***root***,** text**=**'Ammeter'**,** image**=**self**.***imageAmmeter***,** command**=**self**.***ammeterCommand***,**relief**=**RIDGE**)**  1180 self**.***ammeterButton***.***grid***(**row**=**1**,**column**=**1**,**padx**=**2.5**)**  1181 self**.***voltmeterButton* **=** tk**.***Button***(**self**.***root***,** text**=**'Voltmeter'**,** image**=**self**.***imageVoltmeter***,** command**=**self**.***voltmeterCommand***,**relief**=**RIDGE**)**  1182 self**.***voltmeterButton***.***grid***(**row**=**1**,**column**=**2**,**padx**=**2.5**)**  1183 self**.***thermistorButton* **=** tk**.***Button***(**self**.***root***,** text**=**'Thermistor'**,** image**=**self**.***imageThermistor***,** command**=**self**.***thermistorCommand***,**relief**=**RIDGE**)**  1184 self**.***thermistorButton***.***grid***(**row**=**2**,**column**=**0**,**padx**=**2.5**)**  1185 self**.***ldrButton* **=** tk**.***Button***(**self**.***root***,** text**=**'LDR'**,** image**=**self**.***imageLDR***,** command**=**self**.***ldrCommand***,**relief**=**RIDGE**)**  1186 self**.***ldrButton***.***grid***(**row**=**2**,**column**=**1**,**padx**=**2.5**)**  1187 self**.***diodeButton* **=** tk**.***Button***(**self**.***root***,** text**=**'Diode'**,** image**=**self**.***imageDiode***,** command**=**self**.***diodeCommand***,**relief**=**RIDGE**)**  1188 self**.***diodeButton***.***grid***(**row**=**2**,**column**=**2**,**padx**=**2.5**)**  1189 #Set Unit Graph images  1190 self**.***imageiv* **=** PhotoImage**(**file**=**'graphics/ivIcon.png'**)**  1191 self**.***imagert* **=** PhotoImage**(**file**=**'graphics/rtIcon.png'**)**  1192 self**.***imagerli* **=** PhotoImage**(**file**=**'graphics/rliIcon.png'**)**  1193 #Create and set Unit Graph buttons into the grid  1194 self**.***ivButton* **=** tk**.***Button***(**self**.***root***,** text**=**'I/V'**,** image**=**self**.***imageiv***,** command**=**self**.***ivCommand***,**relief**=**GROOVE**)**  1195 self**.***ivButton***.***grid***(**row**=**6**,**column**=**0**,**padx**=**2.5**,**pady**=**50**)**  1196 self**.***rliButton* **=** tk**.***Button***(**self**.***root***,** text**=**'R/LI'**,** image**=**self**.***imagerli***,** command**=**self**.***rliCommand***,**relief**=**GROOVE**)**  1197 self**.***rliButton***.***grid***(**row**=**6**,**column**=**1**,**padx**=**2.5**,**pady**=**50**)**  1198 self**.***rtButton* **=** tk**.***Button***(**self**.***root***,** text**=**'R/T'**,** image**=**self**.***imagert***,** command**=**self**.***rtCommand***,**relief**=**GROOVE**)**  1199 self**.***rtButton***.***grid***(**row**=**6**,**column**=**2**,**padx**=**2.5**,**pady**=**50**)**  1200 #Set Reset button image and create button, setting it into the grid  1201 self**.***imageReset* **=** PhotoImage**(**file**=**'graphics/resetIcon.png'**)**  1202 self**.***resetButton* **=** tk**.***Button***(**self**.***root***,** text**=**'Reset'**,** image**=**self**.***imageReset***,** command**=**self**.***resetCommand***,**relief**=**GROOVE**)**  1203 self**.***resetButton***.***grid***(**row**=**7**,**column**=**0**,**padx**=**2.5**,**pady**=**0**)**  1204 self**.***root***.***mainloop***()**  1205  1206 **def** addLastAction**(**self**,**sprite**):**  1207 #Add component creation to actions stack  1208 lastAction **=** **[**sprite**,**'createdComponent'**]**  1209 actions**.***add***(**lastAction**)**  1210 #Add new component to the graph data structure  1211 circuitGraph**.***addNode***(**sprite**.***ID***)**  1212 ##Button command methods for creating a each component  1213 **def** cellCommand**(**self**):**  1214 newCell **=** Component**(**'cell'**,**numberOfSprites**(),**cellAttributes**,(**500**,**300**))**  1215 self**.***addLastAction***(**newCell**)**  1216 **def** lampCommand**(**self**):**  1217 newLamp **=** Component**(**'lamp'**,**numberOfSprites**(),**lampAttributes**,(**500**,**300**))**  1218 self**.***addLastAction***(**newLamp**)**  1219 **def** resistorCommand**(**self**):**  1220 newResistor **=** Component**(**'resistor'**,**numberOfSprites**(),**resistorAttributes**,(**500**,**300**))**  1221 self**.***addLastAction***(**newResistor**)**  1222 **def** juncCommand**(**self**):**  1223 newJunc **=** Component**(**'junc'**,**numberOfSprites**(),**juncAttributes**,(**500**,**300**))**  1224 self**.***addLastAction***(**newJunc**)**  1225 **def** ammeterCommand**(**self**):**  1226 newammeter **=** Component**(**'ammeter'**,**numberOfSprites**(),**ammeterAttributes**,(**500**,**300**))**  1227 self**.***addLastAction***(**newammeter**)**  1228 **def** voltmeterCommand**(**self**):**  1229 newvoltmeter **=** Component**(**'voltmeter'**,**numberOfSprites**(),**voltmeterAttributes**,(**500**,**300**))**  1230 self**.***addLastAction***(**newvoltmeter**)**  1231 **def** thermistorCommand**(**self**):**  1232 newThermistor **=** Component**(**'thermistor'**,**numberOfSprites**(),**thermistorAttributes**,(**500**,**300**))**  1233 self**.***addLastAction***(**newThermistor**)**  1234 **def** ldrCommand**(**self**):**  1235 newLDR **=** Component**(**'ldr'**,**numberOfSprites**(),**ldrAttributes**,(**500**,**300**))**  1236 self**.***addLastAction***(**newLDR**)**  1237 **def** diodeCommand**(**self**):**  1238 newDiode **=** Component**(**'diode'**,**numberOfSprites**(),**diodeAttributes**,(**500**,**300**))**  1239 self**.***addLastAction***(**newDiode**)**  1240 #Reset button command method for resetting the circuit  1241 **def** resetCommand**(**self**):**  1242 resetCircuitBoard**()**  1243 ##Unit Graph button command methods for creating and setting a new unit graph  1244 **def** ivCommand**(**self**):**  1245 UnitGraph**(**'I'**,** 'V'**)**  1246 **def** rtCommand**(**self**):**  1247 UnitGraph**(**'R'**,** 'T'**)**  1248 **def** rliCommand**(**self**):**  1249 UnitGraph**(**'R'**,** 'LI'**)**  1250 tkWindow**(**'boxWin'**)**  1251 root **=** tk**.***Tk***()**  1252  1253 #####Pygame thread  1254 **def** pygameRun**():**  1255 ##Setup pygame  1256 pygame**.***init***()**  1257 os**.***environ***[**'SDL\_VIDEO\_WINDOW\_POS'**]** **=** "%d,%d" **%** **(**400**,**150**)**  1258 screen **=** pygame**.***display***.***set\_mode***((**1000**,**600**))**  1259 pygame**.***display***.***set\_caption***(**'Circuit Board'**)**  1260 pygame**.***display***.***set\_icon***(**pygame**.***image***.***load***(**'graphics/bolt.png'**))**  1261 clock **=** pygame**.***time***.***Clock***()**  1262 ##Set label surfaces and rectangles  1263 font **=** pygame**.***font***.***SysFont***(**'malgungothic'**,**20**)**  1264 fontsmall **=** pygame**.***font***.***SysFont***(**'malgungothic'**,**18**)**  1265 fontHuge **=** pygame**.***font***.***SysFont***(**'malgungothic'**,**40**)**  1266 statsLabelSurf1 **=** fontsmall**.***render***(**'Shows stats of selected component'**,True,**'#bb3333'**)**  1267 statsLabelRect1 **=** statsLabelSurf1**.***get\_rect***(**midleft **=** **(**450**,**40**))**  1268 statsLabelSurf2 **=** fontsmall**.***render***(**'(right clicked) ------------------>'**,True,**'#bb3333'**)**  1269 statsLabelRect2 **=** statsLabelSurf2**.***get\_rect***(**midleft **=** **(**450**,**70**))**  1270 boxLabelSurf1 **=** fontsmall**.***render***(**'Click component to add it to'**,True,**'#bb3333'**)**  1271 boxLabelRect1 **=** boxLabelSurf1**.***get\_rect***(**midright **=** **(**238**,**40**))**  1272 boxLabelSurf2 **=** fontsmall**.***render***(**'<--------------- the board'**,True,**'#bb3333'**)**  1273 boxLabelRect2 **=** boxLabelSurf2**.***get\_rect***(**midright **=** **(**238**,**70**))**  1274 graphLabelSurf1 **=** fontsmall**.***render***(**'Shows graph of selected type for selected'**,True,**'#bb3333'**)**  1275 graphLabelRect1 **=** graphLabelSurf1**.***get\_rect***(**midleft **=** **(**385**,**390**))**  1276 graphLabelSurf2 **=** fontsmall**.***render***(**'component or whole circuit ------------>'**,True,**'#bb3333'**)**  1277 graphLabelRect2 **=** graphLabelSurf2**.***get\_rect***(**midleft **=** **(**385**,**420**))**  1278 gboxLabelSurf1 **=** fontsmall**.***render***(**'Click on graph type to show the'**,True,**'#bb3333'**)**  1279 gboxLabelRect1 **=** gboxLabelSurf1**.***get\_rect***(**midright **=** **(**265**,**330**))**  1280 gboxLabelSurf2 **=** fontsmall**.***render***(**'<------------------ graph box'**,True,**'#bb3333'**)**  1281 gboxLabelRect2 **=** gboxLabelSurf2**.***get\_rect***(**midright **=** **(**265**,**360**))**  1282 resetLabelSurf1 **=** fontsmall**.***render***(**'Click reset to clear the circuit board'**,True,**'#bb3333'**)**  1283 resetLabelRect1 **=** resetLabelSurf1**.***get\_rect***(**midright **=** **(**293**,**460**))**  1284 resetLabelSurf2 **=** fontsmall**.***render***(**'<------------- (cannot be undone)'**,True,**'#bb3333'**)**  1285 resetLabelRect2 **=** resetLabelSurf2**.***get\_rect***(**midright **=** **(**293**,**490**))**  1286 incLabelSurf1 **=** fontsmall**.***render***(**'Amount to increment selected quantity'**,True,**'#bb3333'**)**  1287 incLabelRect1 **=** incLabelSurf1**.***get\_rect***(**midleft **=** **(**417**,**250**))**  1288 incLabelSurf2 **=** fontsmall**.***render***(**'by for selected sprite ----------------->'**,True,**'#bb3333'**)**  1289 incLabelRect2 **=** incLabelSurf2**.***get\_rect***(**midleft **=** **(**417**,**280**))**  1290 toggleLabelsSurf **=** font**.***render***(**'Toggle Labels: 1,'**,True,**'#bb3333'**)**  1291 toggleLabelsRect **=** toggleLabelsSurf**.***get\_rect***(**midleft **=** **(**5**,**580**))**  1292 toggleElectronsSurf **=** font**.***render***(**'Toggle Electrons: 2,'**,True,**'#3333bb'**)**  1293 toggleElectronsRect **=** toggleElectronsSurf**.***get\_rect***(**midleft **=** **(**165**,**580**))**  1294 toggleControlsSurf **=** font**.***render***(**'Toggle Controls: 3'**,True,**'#800080'**)**  1295 toggleControlsRect **=** toggleControlsSurf**.***get\_rect***(**midleft **=** **(**348**,**580**))**  1296 #Set Stats Box title text  1297 statsTxtSurf **=** font**.***render***(**'Selected Component:'**,True,**'#222222'**)**  1298 statsTxtRect **=** statsTxtSurf**.***get\_rect***(**midleft **=** **(**760**,**20**))**  1299 ##Set Controls text surfaces and rectangles  1300 controlsSurf **=** font**.***render***(**'Controls:'**,True,**'#800080'**)**  1301 controlsRect **=** controlsSurf**.***get\_rect***(**midleft **=** **(**760**,**20**))**  1302 escControlSurf **=** font**.***render***(**'Esc: Close Program'**,True,**'#800080'**)**  1303 escControlRect **=** escControlSurf**.***get\_rect***(**midleft **=** **(**760**,**60**))**  1304 lClickControlSurf **=** font**.***render***(**'L Click: Drag Component'**,True,**'#800080'**)**  1305 lClickControlRect **=** lClickControlSurf**.***get\_rect***(**midleft **=** **(**760**,**90**))**  1306 rClickControlSurf **=** font**.***render***(**'R Click: Select/Drag wire'**,True,**'#800080'**)**  1307 rClickControlRect **=** rClickControlSurf**.***get\_rect***(**midleft **=** **(**760**,**120**))**  1308 delControlSurf **=** font**.***render***(**'Delete: Del Selected'**,True,**'#800080'**)**  1309 delControlRect **=** delControlSurf**.***get\_rect***(**midleft **=** **(**760**,**150**))**  1310 sControlSurf **=** font**.***render***(**'S: Save Screenshot'**,True,**'#800080'**)**  1311 sControlRect **=** sControlSurf**.***get\_rect***(**midleft **=** **(**760**,**180**))**  1312 undoControlSurf **=** font**.***render***(**'Backspace: Undo'**,True,**'#800080'**)**  1313 undoControlRect **=** undoControlSurf**.***get\_rect***(**midleft **=** **(**760**,**210**))**  1314 redoControlSurf **=** font**.***render***(**'Tab: Redo'**,True,**'#800080'**)**  1315 redoControlRect **=** redoControlSurf**.***get\_rect***(**midleft **=** **(**760**,**240**))**  1316 upControlSurf **=** font**.***render***(**'↑ Incr Custom Value'**,True,**'#800080'**)**  1317 upControlRect **=** upControlSurf**.***get\_rect***(**midleft **=** **(**760**,**270**))**  1318 downControlSurf **=** font**.***render***(**'↓ Decr Custom Value'**,True,**'#800080'**)**  1319 downControlRect **=** downControlSurf**.***get\_rect***(**midleft **=** **(**760**,**300**))**  1320 stepControlSurf **=** font**.***render***(**'← → Inc/Dec Step'**,True,**'#800080'**)**  1321 stepControlRect **=** stepControlSurf**.***get\_rect***(**midleft **=** **(**760**,**330**))**  1322 cControlSurf **=** font**.***render***(**'C: Change Custom Value'**,True,**'#800080'**)**  1323 cControlRect **=** cControlSurf**.***get\_rect***(**midleft **=** **(**760**,**360**))**  1324 ##Set initial predetermined variables  1325 pickup **=** **False**  1326 wireOriginSprite **=** **None**  1327 makingWire **=** **False**  1328 wireOriginConnector **=** **None**  1329 rightClickSinglePress **=** **False**  1330 selectedSprite **=** **None**  1331 customOrder **=** 1  1332 leftWire **=** **None**  1333 rightWire **=** **None**  1334 unit1 **=** **None**  1335 unit2 **=** **None**  1336 symbol1 **=** **None**  1337 symbol2 **=** **None**  1338 labelsOn **=** **False**  1339 controlsOn **=** **False**  1340 screenshotData **=** **[]**  1341 screenshotTaken **=** **False**  1342 screenshotIteration **=** 0  1343 saveScreenshot **=** **False**  1344 #Set current screenshot number by loading the screenshotData file containing the number of past screenshots  1345 **with** **open(**'screenshots/screenshotData.txt'**,** 'r'**)** **as** file**:**  1346 **for** line **in** file**:**  1347 screenshotData**.***append***(**line**)**  1348 screenshotNumber **=** **int(**screenshotData**[**0**])**  1349 ###Event loop  1350 **while** **True:**  1351 **if** selectedSprite**:**  1352 **if** **len(**allSprites**)** **==** 0**:**  1353 selectedSprite **=** **None**  1354 screen**.***fill***(**'#999999'**)**  1355 ##Check for events and key presses  1356 **for** event **in** pygame**.***event***.***get***():**  1357 **if** event**.type** **==** pygame**.***QUIT***:**  1358 pygame**.quit()**  1359 **exit()**  1360 **if** event**.type** **==** pygame**.***KEYDOWN***:**  1361 #If the 'c' key is pressed, change the customisable value of the selected sprite if applicable  1362 **if** event**.***key* **==** pygame**.***K\_c***:**  1363 **if** selectedSprite **!=** **None:**  1364 **if** selectedSprite**.***name* **==** 'cell'**:**  1365 **if** selectedSprite**.***customVal* **==** 'emf'**:**  1366 selectedSprite**.***customVal* **=** 'resistance'  1367 **else:**  1368 selectedSprite**.***customVal* **=** 'emf'  1369 **elif** selectedSprite**.***name* **==** 'diode'**:**  1370 **if** selectedSprite**.***customVal* **==** 'minThresh'**:**  1371 selectedSprite**.***customVal* **=** 'resistance'  1372 **else:**  1373 selectedSprite**.***customVal* **=** 'minThresh'  1374 ##If the backspace key is pressed, undo the last action  1375 **if** event**.***key* **==** pygame**.***K\_BACKSPACE***:**  1376 actions**.***undo***()**  1377 selectedSprite **=** **None**  1378 ##If the tab key is pressed, redo the last undone action  1379 **if** event**.***key* **==** pygame**.***K\_TAB***:**  1380 actions**.***redo***()**  1381 selectedSprite **=** **None**  1382 ##If the delete key is pressed, delete the selected sprite  1383 **if** event**.***key* **==** pygame**.***K\_DELETE***:**  1384 **for** sprite **in** IDSprites**:**  1385 **if** sprite **in** componentSpriteGroup**:**  1386 **if** sprite**.***rightClicked* **==** **True:**  1387 #If a voltmeter, remove the wires connecting it to any components  1388 **if** sprite**.***name* **==** 'voltmeter'**:**  1389 **if** sprite**.***voltWires***[**'left'**]:**  1390 sprite**.***pd* **=** 0  1391 IDSprites**.***remove***(**sprite**.***voltWires***[**'left'**])**  1392 IDSprites**.***remove***(**sprite**.***voltWires***[**'right'**])**  1393 allSprites**.***remove***(**sprite**.***voltWires***[**'left'**])**  1394 allSprites**.***remove***(**sprite**.***voltWires***[**'right'**])**  1395 sprite**.***voltWires* **=** **{**'left'**:** **None,** 'right'**:** **None}**  1396 sprite**.***rightClicked* **=** **False**  1397 #Add component deletion to actions stack  1398 actions**.***add***([**sprite**,**'deletedComponent'**])**  1399 #Reset deleted components wires  1400 connectionReset**(**sprite**)**  1401 #Remove component from graph data structure  1402 circuitGraph**.***removeNode***(**sprite**.***ID***)**  1403 #Test if circuit is still valid  1404 circuitGraph**.***circuitValid* **=** circuitGraph**.***circuitTest***()**  1405 **if** circuitGraph**.***circuitValid* **==** **False:**  1406 #Reset all components if circuit is broken  1407 **for** component **in** componentSpriteGroup**:**  1408 #If component is voltmeter reset its connecting wires  1409 **if** component**.***name* **==** 'voltmeter'**:**  1410 **if** component**.***voltWires***[**'left'**]:**  1411 voltConnectedSprite **=** component**.***voltWires***[**'left'**].***sprites***[**1**]**  1412 **if** voltConnectedSprite **==** sprite**:**  1413 component**.***pd* **=** 0  1414 IDSprites**.***remove***(**component**.***voltWires***[**'left'**])**  1415 IDSprites**.***remove***(**component**.***voltWires***[**'right'**])**  1416 allSprites**.***remove***(**component**.***voltWires***[**'left'**])**  1417 allSprites**.***remove***(**component**.***voltWires***[**'right'**])**  1418 component**.***voltWires* **=** **{**'left'**:** **None,** 'right'**:** **None}**  1419 component**.***reset***()**  1420 #Delete all electrons is circuit is broken  1421 **for** electron **in** electronSprites**:**  1422 **for** spriteGroup **in** allGroups**:**  1423 **if** spriteGroup **!=** IDTracker**:**  1424 spriteGroup**.***remove***(**electron**)**  1425 circuitGraph**.***resetVectors***()**  1426 #Remove deleted component from sprite groups  1427 **for** spriteGroup **in** allGroups**:**  1428 **if** spriteGroup **!=** IDTracker**:**  1429 spriteGroup**.***remove***(**sprite**)**  1430 selectedSprite **=** **None**  1431 #If left arrow key is pressed, change the increment step by a factor of 1/10  1432 **if** event**.***key* **==** pygame**.***K\_LEFT***:**  1433 customOrder **/=** 10  1434 #Loop back to 1 if nessessary  1435 **if** customOrder **<** 0.01**:**  1436 customOrder **=** 1  1437 #If right arrow key is pressed, change the increment step by a factor of 10  1438 **if** event**.***key* **==** pygame**.***K\_RIGHT***:**  1439 customOrder **\*=** 10  1440 #Loop back to 0.01 if nessessary  1441 **if** customOrder **>** 1**:**  1442 customOrder **=** 0.01  1443 #If up arrow key is pressed, increase the customisable value for the selected component by the increment step  1444 **if** event**.***key* **==** pygame**.***K\_UP***:**  1445 **if** selectedSprite **!=** **None:**  1446 **if** selectedSprite**.***name* **!=** 'voltmeter' **and** selectedSprite**.***name* **!=** 'ammeter' **and** selectedSprite**.***name* **!=** 'junc'**:**  1447 #Use copy to prevent variable automatic updates for certain values  1448 prevOrder **=** copy**.***copy***(**customOrder**)**  1449 **if** selectedSprite**.***customVal* **==** 'temp'**:**  1450 selectedSprite**.***prevTemp* **=** copy**.***copy***(**selectedSprite**.***temp***)**  1451 **elif** selectedSprite**.***customVal* **==** 'LI'**:**  1452 selectedSprite**.***prevLI* **=** copy**.***copy***(**selectedSprite**.***LI***)**  1453 customOrder **\*=** 10  1454 #Add to customisable value  1455 **vars(**selectedSprite**)[**selectedSprite**.***customVal***]** **+=** customOrder  1456 customOrder **=** prevOrder  1457 ##If component is an LDR/thermistor, calculate its resistance as the circuit does not need to be active for this  1458 **if** selectedSprite**.***name* **==** 'thermistor'**:**  1459 e **=** math**.***exp***(**1**)**  1460 **try:**  1461 selectedSprite**.***resistance* **=** **(**selectedSprite**.***resistance***)\*(**e**\*\*(**3000**\*((**1**/(**selectedSprite**.***temp***+**273**))** **-** **(**1**/(**selectedSprite**.***prevTemp***+**273**)))))**  1462 **except:**  1463 **pass**  1464 **elif** selectedSprite**.***name* **==** 'ldr'**:**  1465 e **=** math**.***exp***(**1**)**  1466 **try:**  1467 selectedSprite**.***resistance* **=** **(**selectedSprite**.***resistance***)\*(**e**\*\*(**5000**\*((**1**/**selectedSprite**.***LI***)** **-** **(**1**/**selectedSprite**.***prevLI***))))**  1468 **except:**  1469 **pass**  1470 #Re-calculate physics with updated values if circuit is complete  1471 **if** circuitGraph**.***circuitValid* **==** **True:**  1472 physicsCalc**()**  1473 #If down arrow key is pressed, decrease the customisable value for the selected component by the increment step  1474 **if** event**.***key* **==** pygame**.***K\_DOWN***:**  1475 **if** selectedSprite **!=** **None:**  1476 **if** selectedSprite**.***name* **!=** 'voltmeter' **and** selectedSprite**.***name* **!=** 'ammeter' **and** selectedSprite**.***name* **!=** 'junc'**:**  1477 prevOrder **=** copy**.***copy***(**customOrder**)**  1478 changeVal **=** selectedSprite**.***customVal*  1479 **if** changeVal **==** 'temp'**:**  1480 selectedSprite**.***prevTemp* **=** copy**.***copy***(**selectedSprite**.***temp***)**  1481 **elif** changeVal **==** 'LI'**:**  1482 selectedSprite**.***prevLI* **=** copy**.***copy***(**selectedSprite**.***LI***)**  1483 customOrder **\*=** 10  1484 **vars(**selectedSprite**)[**changeVal**]** **-=** customOrder  1485 customOrder **=** prevOrder  1486 #In addition to the up arrow functionalites for decreasing values,  1487 # if temperature falls below -273, do not change it and  1488 # then if any custom values other than emf fall below 0, do not change it  1489 **if** changeVal **==** 'temp'**:**  1490 **if** selectedSprite**.***temp* **<** **-**273**:**  1491 selectedSprite**.***temp* **=** **-**273  1492 **elif** changeVal **!=** 'emf' **and** **vars(**selectedSprite**)[**changeVal**]** **<** 0**:**  1493 **vars(**selectedSprite**)[**changeVal**]** **=** 0  1494 **if** selectedSprite**.***name* **==** 'thermistor'**:**  1495 e **=** math**.***exp***(**1**)**  1496 **try:**  1497 selectedSprite**.***resistance* **=** **(**selectedSprite**.***resistance***)\*(**e**\*\*(**3000**\*((**1**/(**selectedSprite**.***temp***+**273**))** **-** **(**1**/(**selectedSprite**.***prevTemp***+**273**)))))**  1498 **except:**  1499 **pass**  1500 **elif** selectedSprite**.***name* **==** 'ldr'**:**  1501 e **=** math**.***exp***(**1**)**  1502 **try:**  1503 selectedSprite**.***resistance* **=** **(**selectedSprite**.***resistance***)\*(**e**\*\*(**5000**\*((**1**/**selectedSprite**.***LI***)** **-** **(**1**/**selectedSprite**.***prevLI***))))**  1504 **except:**  1505 **pass**  1506 **if** circuitGraph**.***circuitValid* **==** **True:**  1507 physicsCalc**()**  1508 #If '1' key is pressed, toggle labels to show and stop showing any controls  1509 **if** event**.***key* **==** pygame**.***K\_1***:**  1510 **if** labelsOn **==** **True:**  1511 labelsOn **=** **False**  1512 **else:**  1513 labelsOn **=** **True**  1514 controlsOn **=** **False**  1515 #If '2' key is pressed, toggle electrons to show  1516 **if** event**.***key* **==** pygame**.***K\_2***:**  1517 **if** pickup **==** **False:**  1518 **for** electron **in** electronSprites**:**  1519 **if** electron**.***shown* **==** **False:**  1520 electron**.***shown* **=** **True**  1521 **else:**  1522 electron**.***shown* **=** **False**  1523 #If '3' key is pressed, toggle controls to show and stop showing any labels  1524 **if** event**.***key* **==** pygame**.***K\_3***:**  1525 **if** controlsOn **==** **True:**  1526 controlsOn **=** **False**  1527 **else:**  1528 controlsOn **=** **True**  1529 labelsOn **=** **False**  1530 #If escape is pressed, close pygame  1531 **if** event**.***key* **==** pygame**.***K\_ESCAPE***:**  1532 pygame**.quit()**  1533 **exit()**  1534 #If the 's' key is pressed set screenshot values to take screenshot further into the event loop  1535 **if** event**.***key* **==** pygame**.***K\_s***:**  1536 saveScreenshot **=** **True**  1537 screenshotTaken **=** **True**  1538  1539 ###Check is mouse buttons are pressed  1540 #Check if left or middle click is pressed  1541 **if** pygame**.***mouse***.***get\_pressed***()[**0**]** **==** **True:**  1542 #Check if a component is already picked up  1543 **if** pickup **==** **False:**  1544 sprite **=** **None**  1545 spriteList **=** **[]**  1546 #Check that user is not dragging a wire  1547 **if** makingWire **==** **False:**  1548 **for** sprite **in** componentSpriteGroup**:**  1549 spriteList**.***append***(**sprite**)**  1550 **for** sprite **in** **reversed(**spriteList**):**  1551 #If mouse is on a component and not the proxCircle, pickup component and set it to follow the mouse  1552 **if** sprite**.***rect***.***collidepoint***(**pygame**.***mouse***.***get\_pos***()):**  1553 sprite**.***pickupStatus* **=** **True**  1554 pickup **=** **True**  1555 #Make any visible electrons invisible  1556 **for** electron **in** electronSprites**:**  1557 **if** electron**.***shown* **==** **True:**  1558 electron**.***shown* **=** **False**  1559 #If the component being moved is a voltmeter, remove any existing wires connecting it to a component  1560 **if** sprite**.***name* **==** 'voltmeter'**:**  1561 **for** key **in** sprite**.***voltWires***:**  1562 IDSprites**.***remove***(**sprite**.***voltWires***[**key**])**  1563 allSprites**.***remove***(**sprite**.***voltWires***[**key**])**  1564 sprite**.***voltWires* **=** **{**'left'**:** **None,**'right'**:** **None}**  1565 #Create next component movement action to be added to the stack  1566 lastAction **=** **[**sprite**,**'movedComponent'**]**  1567 #Save the sprites position to an action file to be loaded for undo/redo  1568 sprite**.***save***(**'new'**)**  1569 sprite**.***rect* **=** sprite**.***surf***.***get\_rect***(**center **=** pygame**.***mouse***.***get\_pos***())**  1570 **break**  1571 **else:**  1572 #If component is being moved, set its center position to the mouse pointer  1573 **for** sprite **in** componentSpriteGroup**:**  1574 **if** sprite**.***pickupStatus* **==** **True:**  1575 sprite**.***rect* **=** sprite**.***surf***.***get\_rect***(**center **=** pygame**.***mouse***.***get\_pos***())**  1576 sprite**.***currentPosition* **=** pygame**.***mouse***.***get\_pos***()**  1577 #If component is a voltmeter and within 75px of another component, create wires between the voltmeter and component  1578 **if** sprite**.***name* **==** 'voltmeter'**:**  1579 **if** nearestPoint**:**  1580 **if** nearestPoint**[**0**].***name* **!=** 'ammeter'**:**  1581 #If wires do not exist between the voltmeter and other component, create them  1582 **if** **not(**sprite**.***voltWires***[**'left'**]** **and** sprite**.***voltWires***[**'right'**]):**  1583 leftWire **=** Wire**(**numberOfSprites**(),**sprite**.***left***,**nearestPoint**[**0**].***left***,**'voltLeft'**)**  1584 rightWire **=** Wire**(**numberOfSprites**(),**sprite**.***right***,**nearestPoint**[**0**].***right***,**'voltRight'**)**  1585 sprite**.***voltWires* **=** **{**'left'**:** leftWire**,**'right'**:** rightWire**}**  1586 leftWire**.***sprites* **=** **[**sprite**,**nearestPoint**[**0**]]**  1587 rightWire**.***sprites* **=** **[**sprite**,**nearestPoint**[**0**]]**  1588 **else:**  1589 #Draw temporary red wires to show where wires will be created when voltmeter is put down  1590 pygame**.***draw***.***aaline***(**screen**,**'#ff'**,**sprite**.***left***,**nearestPoint**[**0**].***left***,**2**)**  1591 pygame**.***draw***.***aaline***(**screen**,**'#ff'**,**sprite**.***right***,**nearestPoint**[**0**].***right***,**2**)**  1592 sprite**.***voltWires***[**'left'**].***ends* **=** **[**sprite**.***left***,**nearestPoint**[**0**].***left***]**  1593 sprite**.***voltWires***[**'right'**].***ends* **=** **[**sprite**.***right***,**nearestPoint**[**0**].***right***]**  1594 sprite**.***voltWires***[**'left'**].***sprites* **=** **[**sprite**,**nearestPoint**[**0**]]**  1595 sprite**.***voltWires***[**'right'**].***sprites* **=** **[**sprite**,**nearestPoint**[**0**]]**  1596 sprite**.***voltWires***[**'left'**].***spriteConnectors* **=** **[**'left'**,**'left'**]**  1597 sprite**.***voltWires***[**'right'**].***spriteConnectors* **=** **[**'right'**,**'right'**]**  1598 **else:**  1599 #Delete existing voltWires if component is not within 75px  1600 sprite**.***voltWires* **=** **{**'left'**:** **None,**'right'**:** **None}**  1601 lw **=** sprite**.***voltWires***[**'left'**]**  1602 rw **=** sprite**.***voltWires***[**'right'**]**  1603 **if** lw **and** lw **in** IDSprites**:**  1604 IDSprites**.***remove***(**lw**)**  1605 allSprites**.***remove***(**lw**)**  1606 **if** rw **and** rw **in** IDSprites**:**  1607 IDSprites**.***remove***(**rw**)**  1608 allSprites**.***remove***(**rw**)**  1609 #Check if right click is pressed  1610 **elif** pygame**.***mouse***.***get\_pressed***()[**2**]** **==** **True:**  1611 #Check that component is not picked up  1612 **if** pickup **==** **False:**  1613 sprite **=** **None**  1614 spriteList **=** **[]**  1615 #Check that user is not dragging a wire  1616 **if** makingWire **==** **False:**  1617 **for** sprite **in** componentSpriteGroup**:**  1618 spriteList**.***append***(**sprite**)**  1619 **for** sprite **in** **reversed(**spriteList**):**  1620 #If mouse is on the proxCircle, create and start dragging a wire  1621 **if** sprite**.***name* **!=** 'junc' **and** proxCircle**.***rect***.***collidepoint***(**pygame**.***mouse***.***get\_pos***())** **and** circuitGraph**.***isCyclic***()** **==** **False:**  1622 makingWire **=** **True**  1623 selectedSprite **=** **None**  1624 wireOriginConnector **=** nearestPoint**[**1**]**  1625 wireOriginSprite **=** nearestPoint**[**0**]**  1626 newWire **=** Wire**(**numberOfSprites**(),**proxCircle**.***pos***,**pygame**.***mouse***.***get\_pos***(),**'newWire'**)**  1627 **break**  1628 #If right click was not pressed in the previous event loop iteration and a wire is not being dragged  1629 **if** rightClickSinglePress **==** **False** **and** makingWire **==** **False:**  1630 **for** sprite **in** IDSprites**:**  1631 **if** sprite **in** componentSpriteGroup**:**  1632 #If mouse is on a component toggle the image to be red or black  1633 **if** sprite**.***rect***.***collidepoint***(**pygame**.***mouse***.***get\_pos***()):**  1634 **if** sprite**.***image* **==** 'base'**:**  1635 **for** redImageSprite **in** componentSpriteGroup**:**  1636 **if** redImageSprite**.***image* **==** 'red'**:**  1637 redImageSprite**.***toggleImage***()**  1638 redImageSprite**.***rightClicked* **=** **False**  1639 selectedSprite **=** sprite  1640 #If a unit graph is selected set the sprite stored to the selected sprite  1641 **if** **len(**unitGraphGroup**)** **>** 0**:**  1642 unitGraphGroup**[**0**].***sprite* **=** sprite  1643 sprite**.***rightClicked* **=** **True**  1644 sprite**.***toggleImage***()**  1645 **elif** sprite**.***image* **==** 'red'**:**  1646 selectedSprite **=** **None**  1647 sprite**.***toggleImage***()**  1648 sprite**.***rightClicked* **=** **False**  1649 rightClickSinglePress **=** **True**  1650 #If mouse clicks not pressed  1651 **else:**  1652 #Is component is picked up  1653 **if** pickup **==** **True:**  1654 #Deselect any components  1655 selectedSprite **=** **None**  1656 #Reset circuit vectors and electrons  1657 circuitGraph**.***resetVectors***()**  1658 #Add last movement to actions stack  1659 actions**.***add***(**lastAction**)**  1660 #If wire is being dragged, stop making wire and update the wire end  1661 **if** makingWire **==** **True:**  1662 makingWire **=** **False**  1663 newWire**.***updateEnd***(**nearestPoint**,**makingWire**,**wireOriginSprite**,**wireOriginConnector**)**  1664 #Reset pickup/dragging variables  1665 pickup **=** **False**  1666 wireOriginSprite **=** **None**  1667 wireOriginConnector **=** **None**  1668 makingWire **=** **False**  1669 rightClickSinglePress **=** **False**  1670 #Re-calculate physics when a voltmeter is connected  1671 **for** sprite **in** componentSpriteGroup**:**  1672 **if** sprite**.***pickupStatus* **==** **True:**  1673 **if** sprite**.***name* **==** 'voltmeter'**:**  1674 **if** sprite**.***voltWires***[**'left'**]** **and** sprite**.***voltWires***[**'right'**]:**  1675 sprite**.***voltWires***[**'left'**].***placed* **=** **True**  1676 sprite**.***voltWires***[**'right'**].***placed* **=** **True**  1677 physicsCalc**()**  1678 sprite**.***pickupStatus* **=** **False**  1679 ###Draw any shapes and text if applicable  1680 **if** labelsOn **==** **True:**  1681 pygame**.***draw***.***rect***(**screen**,**'#999999'**,(**750**,**0**,**600**,**600**))**  1682 pygame**.***draw***.***rect***(**screen**,**'#000000'**,(**750**,**0**,**1200**,**300**),**width**=**3**)**  1683 pygame**.***draw***.***line***(**screen**,**'#000000'**,(**750**,**40**),(**1200**,**40**),**3**)**  1684 #Show increment step with the current set value to increase/decrease by  1685 **if** selectedSprite **!=** **None:**  1686 **if** selectedSprite**.***name* **==** 'ldr'**:**  1687 incrementOrderSurf **=** font**.***render***(**f'Increment Step: < {**str(round(**customOrder**\***10**,**2**))**} >'**,True,**'#222222'**)**  1688 **else:**  1689 incrementOrderSurf **=** font**.***render***(**f'Increment Step: < {**str(round(**customOrder**,**2**))**} >'**,True,**'#222222'**)**  1690 **else:**  1691 incrementOrderSurf **=** font**.***render***(**f'Increment Step: < {**str(round(**customOrder**,**2**))**} >'**,True,**'#222222'**)**  1692 incrementOrderRect **=** incrementOrderSurf**.***get\_rect***(**midleft **=** **(**760**,**280**))**  1693 screen**.***blit***(**statsTxtSurf**,**statsTxtRect**)**  1694 screen**.***blit***(**incrementOrderSurf**,**incrementOrderRect**)**  1695 drawStatsText**(**screen**,**selectedSprite**,**font**)**  1696 #Show basic toggle key controls  1697 screen**.***blit***(**toggleLabelsSurf**,**toggleLabelsRect**)**  1698 screen**.***blit***(**toggleElectronsSurf**,**toggleElectronsRect**)**  1699 screen**.***blit***(**toggleControlsSurf**,**toggleControlsRect**)**  1700 #Draw/blit labels if toggled  1701 **if** labelsOn **==** **True:**  1702 screen**.***blit***(**statsLabelSurf1**,**statsLabelRect1**)**  1703 screen**.***blit***(**statsLabelSurf2**,**statsLabelRect2**)**  1704 screen**.***blit***(**boxLabelSurf1**,**boxLabelRect1**)**  1705 screen**.***blit***(**boxLabelSurf2**,**boxLabelRect2**)**  1706 screen**.***blit***(**graphLabelSurf1**,**graphLabelRect1**)**  1707 screen**.***blit***(**graphLabelSurf2**,**graphLabelRect2**)**  1708 screen**.***blit***(**gboxLabelSurf1**,**gboxLabelRect1**)**  1709 screen**.***blit***(**gboxLabelSurf2**,**gboxLabelRect2**)**  1710 screen**.***blit***(**resetLabelSurf1**,**resetLabelRect1**)**  1711 screen**.***blit***(**resetLabelSurf2**,**resetLabelRect2**)**  1712 screen**.***blit***(**incLabelSurf1**,**incLabelRect1**)**  1713 screen**.***blit***(**incLabelSurf2**,**incLabelRect2**)**  1714 #Draw/blit controls if toggled  1715 **if** controlsOn **==** **True:**  1716 pygame**.***draw***.***rect***(**screen**,**'#999999'**,(**750**,**0**,**1200**,**1200**))**  1717 pygame**.***draw***.***rect***(**screen**,**'#000000'**,(**750**,**0**,**1200**,**700**),**width**=**3**)**  1718 pygame**.***draw***.***line***(**screen**,**'#000000'**,(**750**,**40**),(**1200**,**40**),**3**)**  1719 screen**.***blit***(**controlsSurf**,**controlsRect**)**  1720 screen**.***blit***(**upControlSurf**,**upControlRect**)**  1721 screen**.***blit***(**downControlSurf**,**downControlRect**)**  1722 screen**.***blit***(**stepControlSurf**,**stepControlRect**)**  1723 screen**.***blit***(**cControlSurf**,**cControlRect**)**  1724 screen**.***blit***(**undoControlSurf**,**undoControlRect**)**  1725 screen**.***blit***(**redoControlSurf**,**redoControlRect**)**  1726 screen**.***blit***(**lClickControlSurf**,**lClickControlRect**)**  1727 screen**.***blit***(**rClickControlSurf**,**rClickControlRect**)**  1728 screen**.***blit***(**sControlSurf**,**sControlRect**)**  1729 screen**.***blit***(**escControlSurf**,**escControlRect**)**  1730 screen**.***blit***(**delControlSurf**,**delControlRect**)**  1731 #Draw graph box  1732 pygame**.***draw***.***line***(**screen**,**'#000000'**,(**751**,**300**),(**751**,**600**),**width**=**3**)**  1733 graphTitleSurf **=** font**.***render***(**'Graph'**,True,**'#222222'**)**  1734 graphTitleRect **=** graphTitleSurf**.***get\_rect***(**center **=** **(**875**,**320**))**  1735 #If controls not toggled, draw unit graph  1736 **if** controlsOn **==** **False:**  1737 pygame**.***draw***.***line***(**screen**,**'#000000'**,(**785**,**345**),(**785**,**470**),**width**=**2**)**  1738 pygame**.***draw***.***line***(**screen**,**'#000000'**,(**785**,**470**),(**963**,**470**),**width**=**2**)**  1739 #If unit graph is selected, show units and values  1740 **if** **len(**unitGraphGroup**)** **>** 0**:**  1741 ugraph **=** unitGraphGroup**[**0**]**  1742 unit1 **=** ugraph**.***axies***[**'unit1'**][**0**]**  1743 symbol1 **=** ugraph**.***axies***[**'unit1'**][**1**]**  1744 unit2 **=** ugraph**.***axies***[**'unit2'**][**0**]**  1745 symbol2 **=** ugraph**.***axies***[**'unit2'**][**1**]**  1746 ytitleSurf **=** font**.***render***(**f'{unit1}'**,True,**'#222222'**)**  1747 ytitleRect **=** ytitleSurf**.***get\_rect***(**midright **=** **(**780**,**355**))**  1748 yunitSurf **=** font**.***render***(**f'/{symbol1}'**,True,**'#222222'**)**  1749 yunitRect **=** yunitSurf**.***get\_rect***(**midright **=** **(**783**,**375**))**  1750 xtitleSurf **=** font**.***render***(**f'{unit2}'**,True,**'#222222'**)**  1751 xunitSurf **=** font**.***render***(**f'/{symbol2}'**,True,**'#222222'**)**  1752 xunitRect **=** xunitSurf**.***get\_rect***(**midright **=** **(**963**,**485**))**  1753 **if** unit2 **==** 'LI'**:**  1754 xtitleRect **=** xtitleSurf**.***get\_rect***(**midright **=** **(**920**,**485**))**  1755 **elif** unit2 **==** 'T'**:**  1756 xtitleRect **=** xtitleSurf**.***get\_rect***(**midright **=** **(**933**,**485**))**  1757 **else:**  1758 xtitleRect **=** xtitleSurf**.***get\_rect***(**midright **=** **(**938**,**485**))**  1759 graphTitleSurf **=** font**.***render***(**f'Graph: {unit1} against {unit2}'**,True,**'#222222'**)**  1760 graphTitleRect **=** graphTitleSurf**.***get\_rect***(**center **=** **(**874**,**320**))**  1761 #Show selected sprite name under graph  1762 **if** selectedSprite**:**  1763 showText **=** ''  1764 **if** selectedSprite**.***name* **==** 'junc'**:**  1765 showText **=** 'Junction'  1766 **elif** selectedSprite**.***name* **==** 'ldr'**:**  1767 showText **=** 'LDR'  1768 **elif** selectedSprite**.***name* **==** 'cell'**:**  1769 showText **=** 'Cell/Power Source'  1770 **else:**  1771 showText **=** selectedSprite**.***name***.***title***()**  1772 graphComponentSurf **=** font**.***render***(**f'({showText})'**,True,**'#222222'**)**  1773 graphComponentRect **=** graphComponentSurf**.***get\_rect***(**center **=** **(**874**,**520**))**  1774 screen**.***blit***(**graphComponentSurf**,**graphComponentRect**)**  1775 screen**.***blit***(**graphTitleSurf**,**graphTitleRect**)**  1776 screen**.***blit***(**ytitleSurf**,**ytitleRect**)**  1777 screen**.***blit***(**yunitSurf**,**yunitRect**)**  1778 screen**.***blit***(**xtitleSurf**,**xtitleRect**)**  1779 screen**.***blit***(**xunitSurf**,**xunitRect**)**  1780 **else:**  1781 screen**.***blit***(**graphTitleSurf**,**graphTitleRect**)**  1782 ##If a graph and component is selected, plot lines  1783 **if** selectedSprite **!=** **None** **and** **len(**unitGraphGroup**)** **>** 0**:**  1784 unitGraph **=** unitGraphGroup**[**0**]**  1785 **if** unitGraph**.***unit2* **==** 'LI'**:**  1786 **if** selectedSprite**.***name* **==** 'ldr'**:**  1787 #Draw r-li relationship  1788 unitGraph**.***plotTempLi***(**screen**)**  1789 **else:**  1790 #Draw n/a  1791 unitGraph**.***plotNA***(**screen**,**font**)**  1792 **elif** unitGraph**.***unit2* **==** 'T'**:**  1793 **if** selectedSprite**.***name* **==** 'thermistor'**:**  1794 #Draw r-t relationship  1795 unitGraph**.***plotTempLi***(**screen**)**  1796 **else:**  1797 #Draw n/a  1798 unitGraph**.***plotNA***(**screen**,**font**)**  1799 **elif** unitGraph**.***unit2* **==** 'V'**:**  1800 **if** selectedSprite**.***name* **==** 'diode'**:**  1801 #Draw i-v relationship (diode)  1802 unitGraph**.***plotDiodeV***(**screen**)**  1803 **elif** selectedSprite**.***name* **==** 'lamp' **or** selectedSprite**.***name* **==** 'thermistor'**:**  1804 #Draw i-v relationship (lamp/thermistor)  1805 unitGraph**.***plotLampThermV***(**screen**)**  1806 **elif** selectedSprite**.***name* **==** 'resistor' **or** selectedSprite**.***name* **==** 'ldr'**:**  1807 #Draw linear line relationship (resistor/ldr)  1808 unitGraph**.***plotLinear***(**screen**)**  1809 **else:**  1810 #Draw n/a  1811 unitGraph**.***plotNA***(**screen**,**font**)**  1812 #If circuit is complete and vectors haven't been made, create vectors and electrons  1813 **if** circuitGraph**.***circuitValid* **==** **True:**  1814 **if** circuitGraph**.***vectorsMade* **==** **False:**  1815 circuitGraph**.***makeVectors***()**  1816 #Get the nearest point as [nearestComponent, nearestConnector (L/R for dragging, and T/B for voltmeter), distanceFromMouse]  1817 nearestPoint **=** findNearestConnector**(**wireOriginSprite**)**  1818 #Iterate through sprite group and blit sprites  1819 **for** sprite **in** allSprites**:**  1820 #Show any components on the screen  1821 **if** sprite **in** componentSpriteGroup**:**  1822 #If mouse hovering on/off component change image to hover/non-hover  1823 **if** sprite**.***rect***.***collidepoint***(**pygame**.***mouse***.***get\_pos***()):**  1824 sprite**.***hover***(**'Hov'**)**  1825 **else:**  1826 sprite**.***hover***(**''**)**  1827 #If circuit is valid, set connected components to be active  1828 **if** circuitGraph**.***circuitValid* **==** **True:**  1829 **for** component **in** componentSpriteGroup**:**  1830 **if** sprite**.***ID* **in** circuitGraph**.***graph***:**  1831 numberOfConnections **=** 0  1832 **for** key **in** sprite**.***connected***:**  1833 **if** sprite**.***connected***[**key**]** **!=** **None:**  1834 numberOfConnections **+=** 1  1835 **if** numberOfConnections **==** 2**:**  1836 sprite**.***active* **=** **True**  1837 **else:**  1838 sprite**.***active* **=** **False**  1839 #Show component and update its stored side positions  1840 screen**.***blit***(**sprite**.***surf***,**sprite**.***rect***)**  1841 sprite**.***update***()**  1842 **else:**  1843 #Show proxCircle at left/right connector if nearest point is found  1844 **if** sprite**.***name* **==** 'proxCircle' **and** pickup **==** **False** **and** circuitGraph**.***isCyclic***()** **==** **False:**  1845 **if** nearestPoint**[**0**].***name* **!=** 'voltmeter'**:**  1846 **if** nearestPoint**[**0**].***name* **==** 'junc'**:**  1847 #Set smaller image for proxCircle if nearest component is a junction  1848 proxCircle**.***surf* **=** pygame**.***image***.***load***(**'graphics/proxCircleSmall.png'**)**  1849 **else:**  1850 #Reset to normal image for proxCircle when nearest component is not a junction  1851 proxCircle**.***surf* **=** pygame**.***image***.***load***(**'graphics/proxCircle.png'**)**  1852 sprite**.***showClosest***(**nearestPoint**,**wireOriginSprite**)**  1853 **if** nearestPoint**[**1**]** **!=** 'top' **and** nearestPoint**[**1**]** **!=** 'bottom'**:**  1854 screen**.***blit***(**sprite**.***surf***,**sprite**.***rect***)**  1855 ##Show wires  1856 **elif** sprite**.***name* **==** 'wire'**:**  1857 ##If wire being dragged, set the end to the current mouse position to follow it  1858 **if** makingWire **==** **True** **and** sprite **==** newWire**:**  1859 sprite**.***updateEnd***(**nearestPoint**,**makingWire**,**wireOriginSprite**,**wireOriginConnector**)**  1860 pygame**.***draw***.***aaline***(**screen**,**sprite**.***color***,**sprite**.***ends***[**0**],**sprite**.***ends***[**1**],**5**)**  1861 **else:**  1862 #If not dragging wire, delete it if not connected  1863 **if** sprite**.***placed* **==** **False:**  1864 **if** sprite**.***sprites***[**0**]:**  1865 **if** sprite**.***sprites***[**0**].***name* **!=** 'voltmeter'**:**  1866 allSprites**.***remove***(**sprite**)**  1867 IDSprites**.***remove***(**sprite**)**  1868 #If wire connected, draw it  1869 **else:**  1870 sprite**.***update***()**  1871 pygame**.***draw***.***aaline***(**screen**,**sprite**.***color***,**sprite**.***ends***[**0**],**sprite**.***ends***[**1**],**2**)**  1872 ##If circuit is valid, show electrons and update their positions  1873 **if** circuitGraph**.***circuitValid* **==** **True:**  1874 **for** electron **in** electronSprites**:**  1875 **if** circuitGraph**.***vectorsMade* **==** **True:**  1876 ex **=** electron**.***currentPos***[**0**]**  1877 ey **=** electron**.***currentPos***[**1**]**  1878 vects **=** circuitGraph**.***vectorsCycle*  1879 emfSign **=** circuitGraph**.***emfSign*  1880 currentMult **=** circuitGraph**.***currentMultiplier*  1881 #Add to electron x position by moveBy vector x value  1882 #Emf sign sets the direction (pos/neg/0), currentMultiplier sets the number of times to add "moveBy" for speed  1883 ex **=** ex **+** currentMult**\*(**emfSign**\*(**vects**[**electron**.***vectorIndex***][**3**].***moveBy***[**0**]))**  1884 **if** vects**[**electron**.***vectorIndex***][**0**]** **==** 'vector'**:**  1885 #Add to electron y position by moveBy vector y value  1886 ey **=** ey **+** currentMult**\*(**emfSign**\*(**vects**[**electron**.***vectorIndex***][**3**].***moveBy***[**1**]))**  1887 ##If electron has reached the end of its current vector, increment the current vectorIndex to follow the next one  1888 **if** emfSign **==** 1**:**  1889 **if** vects**[**electron**.***vectorIndex***][**3**].***pos2***[**0**]** **-** vects**[**electron**.***vectorIndex***][**3**].***pos1***[**0**]** **>** 0**:**  1890 **if** electron**.***currentPos***[**0**]** **>=** vects**[**electron**.***vectorIndex***][**3**].***pos2***[**0**]:**  1891 ex **=** vects**[**electron**.***vectorIndex***][**3**].***pos2***[**0**]**  1892 ey **=** vects**[**electron**.***vectorIndex***][**3**].***pos2***[**1**]**  1893 electron**.***vectorIndex* **+=** 1  1894 **else:**  1895 **if** electron**.***currentPos***[**0**]** **<=** vects**[**electron**.***vectorIndex***][**3**].***pos2***[**0**]:**  1896 ex **=** vects**[**electron**.***vectorIndex***][**3**].***pos2***[**0**]**  1897 ey **=** vects**[**electron**.***vectorIndex***][**3**].***pos2***[**1**]**  1898 electron**.***vectorIndex* **+=** 1  1899 **elif** emfSign **==** **-**1**:**  1900 **if** vects**[**electron**.***vectorIndex***][**3**].***pos2***[**0**]** **-** vects**[**electron**.***vectorIndex***][**3**].***pos1***[**0**]** **>** 0**:**  1901 **if** electron**.***currentPos***[**0**]** **<=** vects**[**electron**.***vectorIndex***][**3**].***pos1***[**0**]:**  1902 ex **=** vects**[**electron**.***vectorIndex***][**3**].***pos1***[**0**]**  1903 ey **=** vects**[**electron**.***vectorIndex***][**3**].***pos1***[**1**]**  1904 electron**.***vectorIndex* **-=** 1  1905 **else:**  1906 **if** electron**.***currentPos***[**0**]** **>=** vects**[**electron**.***vectorIndex***][**3**].***pos1***[**0**]:**  1907 ex **=** vects**[**electron**.***vectorIndex***][**3**].***pos1***[**0**]**  1908 ey **=** vects**[**electron**.***vectorIndex***][**3**].***pos1***[**1**]**  1909 electron**.***vectorIndex* **-=** 1  1910 **if** electron**.***vectorIndex* **>=** **len(**vects**):**  1911 electron**.***vectorIndex* **=** 0  1912 **if** electron**.***vectorIndex* **<** 0**:**  1913 electron**.***vectorIndex* **=** **len(**vects**)-**1  1914 #Update new electron position and rectangle  1915 electron**.***currentPos* **=** **(**ex**,**ey**)**  1916 electron**.***rect* **=** electron**.***surf***.***get\_rect***(**center **=** electron**.***currentPos***)**  1917 #Show electron if toggled to show  1918 **if** electron**.***shown* **==** **True:**  1919 screen**.***blit***(**electron**.***surf***,**electron**.***rect***)**  1920 #If 's' has been pressed in the last 400 event loops  1921 **if** screenshotTaken **==** **True** **and** screenshotIteration **<=** 400**:**  1922 #If 's' has been pressed in the current event loop, take screenshot  1923 **if** saveScreenshot **==** **True:**  1924 saveScreenshot **=** **False**  1925 pygame**.***image***.***save***(**screen**,** f'screenshots/screenshot{screenshotNumber}.png'**)**  1926 **with** **open(**'screenshots/screenshotData.txt'**,** 'w'**)** **as** file**:**  1927 screenshotNumber **+=** 1  1928 file**.***write***(str(**screenshotNumber**))**  1929 #Show screenshot message to notify to user  1930 pygame**.***draw***.***rect***(**screen**,**'#999999'**,[**200**,**200**,**400**,**100**])**  1931 pygame**.***draw***.***rect***(**screen**,**'#ff4444'**,[**200**,**200**,**400**,**100**],**width**=**3**)**  1932 ssTakenSurf **=** fontHuge**.***render***(**'Screenshot Taken'**,True,**'#3333ee'**)**  1933 ssTakenRect **=** ssTakenSurf**.***get\_rect***(**midleft **=** **(**220**,**235**))**  1934 ssSavedSurf **=** font**.***render***(**'Saved to screenshots folder'**,True,**'#3333ee'**)**  1935 ssSavedRect **=** ssSavedSurf**.***get\_rect***(**center **=** **(**400**,**270**))**  1936 screen**.***blit***(**ssTakenSurf**,**ssTakenRect**)**  1937 screen**.***blit***(**ssSavedSurf**,**ssSavedRect**)**  1938 screenshotIteration **+=** 1  1939 **else:**  1940 screenshotIteration **=** 0  1941 screenshotTaken **=** **False**  1942  1943 #Update display and clock  1944 pygame**.***display***.***update***()**  1945 clock**.***tick***(**300**)**  1946  1947 #####Threading  1948 #Create pygame and tkinter threads  1949 pygameThread **=** threading**.***Thread***(**target**=**pygameRun**)**  1950 tkBoxThread **=** threading**.***Thread***(**target**=**tkBoxRun**)**  1951 #Start pygame and tkinter threads  1952 pygameThread**.***start***()**  1953 tkBoxThread**.***start***()** |

# Evaluation

## Meeting objectives

### Objective 1:

Objective 1 states that the program should include most theories and aspects of A-level Physics OCR Module 4.1-4.3. This is the current electricity module and overall the program shows many and if not most of the concepts for circuits and electricity.

Objectives 1.1 and 1.1.1 are complete, as the program includes all circuit components which are relevant to a circuit builder (cell/power source, junction, fixed resistor, filament lamp, ammeter, voltmeter, thermistor, light dependant resistor (LDR), and diode), and each component correctly shows the symbol which represents it.

Objective 1.1.2 however is not fully complete as although all components can be connected together to form a series circuit, the finished program does not allow the user to connect components together as a parallel circuit. This means that some concepts and ideas around current electricity have not been represented successfully. Thought it had been attempted, creating an algorithm to detect multiple cycles and individual loops proved too difficult for the timescale of the programming, meaning I made the decision to not include parallel circuits and focus on making the series circuits and other features work to the best standard, allowing the user to creating any series circuit that they wish with any combination of components as well as operating the rest of the program smoothly with ease and gaining an improved understanding of the physics behind circuits. In future and with more time, I would make sure to add the ability to create parallel circuits to the program as it would allow some of the final areas of module 4.1-4.3 to be represented.

Objective 1.2 and all subsequent objectives are completed to a full standard as each component has its correct circuit attributes and all listed attributes are included.

* The cell component has e.m.f., internal resistance, current, power, and (lost volts to go with its internal resistance).
* The junction component only has current as this acts like a buffer for the circuit, allowing the user to add a point for wires to connect to without changes to any calculations.
* The (fixed) resistor and (filament) lamp have current, voltage, resistance, and power.
* The ammeter has only current as in circuits it acts as a device to measure the current at a given point in the circuit.
* The voltmeter has only voltage as in circuits it acts as a device to measure the voltage (or potential difference) across a given component.
* The thermistor has current, voltage, resistance, power, and temperature. This is implemented as the standard thermistor in the physics specification (NTC thermistor) where temperature and resistance are inversely proportional.
* The LDR is the same as the thermistor except having light intensity rather than temperature, which follows the same relationship that temperature and resistance had in the thermistor.
* The diode has current, voltage, resistance, power, and minimum voltage.

Objective 1.3 has been fully completed as the circuit calculates all physics values correctly when a circuit has been completed (see a breakdown of the calculations made under the design section). In addition to this for 1.3.6-1.3.8, the program allows users to change certain values of each component to then re-calculate the physics unless the circuit is broken in which all values for components are reset. Coding the physics calculations was particularly difficult as it required implementing a section of code to take existing values of each component before correctly calculating each value in the correct order and updating them for each component.

### Objective 2:

Objective 2 states that the program should allow the user to create circuits freely, which overall has been completed successfully as any number of components can be connected into a circuit in any format and have their values updated change and alter the rest of the circuits’ properties.

Objective 2.1 has been fully completed as the program correctly detects when a circuit is complete, checking for a cycle and testing if a cell is present in the circuit. Objective 2.1.2 is no longer relevant however as parallel circuits have not been implemented. This means that there is no need to detect which components are in any loops. In future if parallel circuits were implemented, this could be added to the program to simply count each component which is in the cycle as the traversal algorithm runs. For detecting the cycle in my program, creating an algorithm from scratch proved to be too difficult within my timeframe, so I decided to find and research for existing algorithms for cycle detection in a graph and found one that used a DFS search that could be modified to fit into my program specifically.

Objective 2.2 has been fully completed correctly as the user can drag and drop each component in a smooth motion, placing them anywhere in the pygame window. When components are hovered over their image changes to notify the user, and the user can right click a component to select it and see its physics attributes. This was necessary to implement as it allowed the user to make the circuit truly unique and change any values to manipulate the physics calculations in any way.

Objective 2.3 has been fully completed as the user is given to ability to drag wires from each side of each component and connect multiple components together. Designing the way to implement this ability into the program proved difficult as it required a way to allow connections between components to be made both vertically and horizontally, and each connection had to be registered within each component so that cycles could be detecting in the circuit. Initially, I thought about implementing this by allowing the user to connect two components directly next to each other, however I quickly realised that this would not work well due to this only letting the user connect components horizontally and wires would be needed for vertical connections, meaning connecting wires between *every* point would be much more beneficial. When wires are connected, moving a component updates the wire ends to follow it, which was found to be difficult to implement as it was difficult to store and access each wire and its various connected components.

Objective 2.4 is fully complete as the user is able to create a screenshot of their circuit. A message box pops up to notify them, and it is saved with a unique name into the screenshots folder.

### Objective 3:

Objective 3 states that the circuits created should be stored as a graph data structure with nodes and edges.

Objectives 3.1-3.6 are fully completed correctly as each component is represented as a node and each wire is represented as an edge. The graph is also implemented with memory efficiency in mind as it uses an adjacency list rather than an adjacency matrix as the number of component is always greater than or equal to the number of edges, since the graph is also unidirectional and unweighted.

Objective 3.7 is fully complete as the graph contains a cycle detection algorithms to, with the test for a cell being present, allow the circuit to be considered complete. Initially this proved to be particularly difficult to program from scratch and so after taking and modifying a dfs cycle detection algorithm online, I was able to create an algorithm and method to successfully test for a complete circuit.

### Objective 4:

Objective 4 states that the program should have a simple easy to use UI and be accessible for new users with less technological experience.

Objectives 4.1-4.3 have been fully completed as the user is shown by a static label which always shows that they can press 1 to toggle on the labels which point to various areas of the windows to explain what each section is for. The program is entirely accessed through the pygame and tkinter GUI and does not require the user to input data into python’s shell. This was particularly important as inputting data into the shell whilst attempting to operate the program multiple windows if very difficult for perhaps newer users.

Objective 4.4 was particularly important to implement as it allows the user to undo and redo their actions when creating circuits. This is important as without this, making a mistake during circuit creation would require either resetting the entire circuit or having to re-create and connect component and/or wires again. When designing my undo-redo system, I initiaially planned on creating two stacks (1 for undo, and 1 for redo), however I realised with the help of a circular queue-like implementation that I had created, I could modify it to store the users actions without having to delete them when undoing and redoing (as pointers can be used to point at specific actions when they are to be undone and redone. When implementing initially I also deicided that the only the users last 5 actions could be stored, however I decided it would be better after programming to increase the number of actions which could be stored to 10, as this gives users much more leeway for making incorrect movements or connections when building the circuit. The user can undo and redo: creating a component, moving a component, deleting a component, and creating a wire. When implementing undo and redo for movement of components, I decided that it would be beneficial to save the attributes of the component to a file so that it could be stored externally and then re-opened for components to be updated later. I decided to use the pickle module to serialise the component data so it could be easily saved and loaded.

Objective 4.5 has been fully completed as the user is able to select a unit graph and see how the relationship between different components is different for different units. The graph is able to be selected in the tkinter window and proceeds to change the axes in the pygame window in the correct box. The graph is able to show multiple types of relationships, for example linear, curved/polynomial, and reciprocal graphs for various components. The three graphs which are applicable for a circuit builder simulation are correctly implemented and show the correct relationships: I against V, R against Li, and R against T (more details on the graph under the design section). I thought this was important to implement as it allows the user to learn more about the relationships between these units which are necessary in the physics course.

### Objective 5:

Objective 5 states that the program should be able to show electrons flowing around the circuit along each wire and component.

Objective 5.1 has been fully completed as the user can press 2 to toggle whether the electrons are visible or not. Objective 5.2 however, has only been mostly completed as the electrons flow direction is not always in the correct direction. Although sometimes they may not flow in the correct direction initially, when the total emf of the circuit becomes negative the direction of flow still switches to be opposite from what it was before. In addition to this, when the total emf is 0, the electrons remain stationary and do not move in any direction.

Objective 5.3 has been fully completed as when the magnitude of the total emf increases (and therefore current) the electrons move much faster and speed up. In addition to this, when the current increases to a given point and the electrons are moving very fast, the currentMultiplier used to move them at a given speed will no longer increase. This was important to implement sussessfully as it is important for the user to be able to interpret the electron flow direction (which may prove difficult on extremely high speeds).

Objective 5.4 has been fully completed as when the circuit is complete, a method is called in the graph to traverse through the components and wires (nodes and edges), creating vectors along each in the correct order in the cycle. These vectors are created and stored as objects so that when electrons are moved, they can move along each vector’s path in proportion to its total size (meaning they will always be travelling at the same speed). When the circuit is complete, the electrons do correctly move around the circuit in the correct order.

Objectives 5.5-5.8 have also been fully completed. The electrons are implemented as a class to allow ease of access and updates. The program correctly calculates the number of electrons that must be created to allow for a 50px gap, and then calculates the actual spacing that each electron which will be close to 50px as a decimal. When the user moves a component new vectors and electrons are created to allow for the new shape of the circuit. These objectives were particularly important to implement successfully as they allow for the visual experience of the flowing electrons to be the same no matter the size or shape of the circuit.

### Objective 6:

Objective 6 states that the program should represent all non-static elements as a class e.g. components or electrons. For each visual object pygame’s in-built sprite creation should be used to allow different objects to be easily grouped in sprite groups and allow methods to be performed on them.

Objective 6 has been fully completed as various sprite groups are used to allow different types of objects to be easily found and accessed. Each component also has the same methods and attributes so that they can be cross references no matter what the type is. Any attributes which are specific to certain components are created using conditions. Implementing certain elements as objects and sprites was necessary as it allows the program to run smoothly when updating/calling methods/attributes.

## Successes and improvements

### Classes and object creation

Classes and objects have been implemented well into my code because they allow for easy access to attributes and methods for different elements. Components and wires can be stored together easily whilst calling update methods when needed. The graph can be easily created and used to store the circuit and test for when it is complete, as well as creating vectors and electrons to then allow them to cycle through the circuit and update their positions. The circular buffer/stack can also be implemented easily to call undo/redo functions and store the user’s actions.

To improve the way classes have been implemented, I could use inheritance between classes, as this would allow for components to be made in specific classes and have their own methods and attributes on top of an inherited generic component class, rather than using conditions to add attributes depending on the type.

### Circuit Graph, cycle detection, and electrons/vectors

I found lots of success when programming the graph data structure and implementing it into my program. I first designed a basic class object for the graph with an adjacency list to store its nodes and edges, and methods such as adding and removing nodes. This initial basic graph setup then allowed me to further implement the graph to manipulate my specific program.

When implementing the cycle detection algorithm for testing if the circuit is complete, I originally tried to implement it myself from scratch which I struggled with understanding where to start. Due to the short time frame, I decided it would be better to take an existing dfs cycle detection algorithm and modify it to fit into my code, in which I was successful and the circuit was now able to be tested if it was complete by combining this with a test for whether a cell was connected to the circuit.

To improve the graph and its cycle detection algorithm in the future, I could create my own cycle detection algorithm specifically for the program itself as this would allow a much more efficient section of code to be used as it would be coded in such a way that it could communicate directly with certain components or wires, allowing them to be easily stored in an attribute when they are in the cycle for later processes.

I also found lots of success when programming the electron and vector creation for moving the electrons around the circuit when it is complete. To get this to work, I first coded a basic version to understand every detail and process the algorithm required. After understanding this, I could then go through and modify my basic algorithm to create a fully working one, using a recursive algorithm to loop around the circuit graph creating vectors to follow through each component and wire. These vectors could then be used to find the number of electrons and the exact spacing that they should be apart (~50px).

To allow the electrons to move, I implemented them as objects to have a current vector pointer which pointed to a specific vector stored in a list of vector objects in the correct cycle order. Electrons could then easily move and have their x and y positions updated in proportion to its current vectors total magnitude, have its direction calculated using the “emfSign” (1, -1, or 0), and the speed using the “currectMultiplier” to multiply the default x and y values to add.

To improve the electrons and vectors, I could use a method within the graph class to update their positions rather than update them directly in the event loop as it would make the code slightly more organised.

### Undo-redo system

The undo-redo system is implemented well because it allows the user’s last 10 actions to be stored which can then be shifted through to be undone and redone, executing different lines depending on the action.

When moving a component, the component’s position is saved to a file and serialised, this can then be loaded when un/redoing the movement using a pointer specifically for these files. To improve the undo and redo system, it could be changed so that all actions are stored in files rather than just the movement of a component. This would mean there could be universal pointers for all actions rather than them being specifically for one type of action.

To further improve the undo-redo system, I could implement an extra slot could be added to the stack to store the most recent or latest position when the stack is full. This would allow the first action to not be overridden when the stack is full, and an action is saved after being undone.

### Physics calculations

The physics calculations area of the program is implemented well because it allows for a minimum number of pre-determined values (emf, resistance, temperature, light intensity, and minimum voltage) to be used to calculate all other values. The calculations are implemented as a subroutine procedure which calculates everything in one run, allowing for quick updates when values are changed. Each value is calculated correctly to follow how physics works in real circuits and allows the user to correctly see how each value is affected by others.

To improve the physics calculations, I could make it so when the function for calculating is called some values may not change and the procedure could have additional lines to only attempt to update values which have been changed.

The user’s ability to change the pre-determined values has also been implemented well as they are able to change the precision of how much they increment with each press of the arrow keys and change the pre-determined value to increment when applicable. This allows the user to fully customise their circuit with the components they have and set its values to change others accordingly.

### Threading for tkinter and pygame

The use of threading for tkinter and pygame has been implemented well as it allows for both tkinter and pygame to run at the same time. This means the built-in buttons for tkinter can be easily used whilst creating objects within pygame itself.

To improve the ability for tkinter and pygame to run simultaneously, I could multiple python files could be used rather than threading as they may also allow for less global variables required to pass information between the tkinter and pygame threads.

### Dragging and creating wires

The ability to drag and create wires has been implemented well as the user is able to create wires between components in any direction and of any length. When the user is dragging a wire they can release left click without having to directly connect it, as it snaps on to the other component. The ability to drag wires has also been implemented well as dragging components and dragging wires uses separate buttons (left and right click) so that when creating the circuit the user does not accidentally perform the wrong action.

### Screenshotting

The ability to screenshot the circuit has been implemented well as it allows the user to simply create a screenshot with one button which can then be saved into a folder with a unique name. A message box shows the user that they have taken a screenshot.

To improve screenshotting, a time based counter could be used rather than relying on the event loop cycles which may be slower if the device runs slightly slower.

## User Feedback

For user feedback, I asked people who may use my program to try it out and give feedback on their experience and share what they thought by answering some questions.

1. What do you think has been implemented well into my program?

One of the things that multiple people said I had implemented well was the number of components and therefore number of unique circuits that could be made. They said that that this would allow them to create whatever circuits that they needed for example to understand (or help others to understand if using as a teacher) how certain components act when connected together.

I agree that this is one of the largest strengths of my program as the ability to make 1000s of complex circuits with various values is important for the user to be able to understand and improve their knowledge on the areas that they feel they need to.

1. Do you think my program represents physics well and in an understandable way?

For this question people were majoritively happy with the way my program shows and represents physics, saying that the ability to select and change components’ circuit values was implemented well: it allowed them to change the circuit easily. People also said that the ability to see electrons flowing through the circuit showed a good representation of physics as it allows for people to understand and see what might be happening behind the scenes in a circuit rather than just visible changes.

People also said however that some physics cannot be represented as parallel circuits cannot be made.

I agree that my program implements physics well because it allow the users to further customise their circuit past simple choosing which components to connect together. The electrons are also very useful and represent how real electrons flow flow around a circuit depending on the current/emf supplied to it. I also agree that some of the physics could not be represented by my program as parallel circuits can’t be made, however if I had had and time or if I were to re-make the program in future I would be sure to add parallel circuits to fill this gap.

1. What do you think could be improved in my program?

Unit graph show real time data

For this question people said that my program could be improved by making the unit graph change what it plots based on the calculated values in the circuit (as currently it only shows generic relationships). People also said that I could try to attempt making time-based unit graphs for example charge against time for the equation Q = It.

I agree that this would be a good improvement and if I were to code my program again, I would make sure to add this as a feature so that the unit graph can show the values that have been calculated in real time. I would also try my best and come up with a way to implement more unit graphs to show time based relationships, for example a graph of charge (Q) against time (t).

People also said that my program could be improved by (as spoken about before) giving the ability to create parallel circuits. I agree that this would be an improvement as it would allow the user to make even more complex circuits and fill the final gaps for my program to represent all physics that it possibly could.

1. Do you think you would be able to understand and use my program to either help improve your own understanding or help improve others understanding?

For this question most students said that they would use my program to help them understand current electricity and circuits more, using it as a revision resource. I agree that my program would be useful for some people to revise from and improve their understanding because it can provide a visual representation for someone to make and create circuits in whichever way they need, however everybody studies and understands information in different ways which is why some people may choose not to use my program and stick to descriptions of physics.

Teachers said that my program may be useful when teaching their lessons on current electricity as it would (similar to students using it themselves), create a visual and interactive representation of a physics circuit which can then be changed and have its values updated to show how the internal physics works and how to calculate it.

## General improvements

One large improvement I would make to my code if I had had the time would be to fully implement the ability to create parallel circuits as well as series circuits. This would allow the user to create and understand more about current electricity as although series circuits covers almost all the content required, parallel circuits would mean that the final parts of content would be able to be covered so that the user could learn and understand circuits even more.

Another improvement I would make to my code is that I could change the colour scheme of the GUI so that it is more vibrant rather than a simple black grey and white colour scheme (sometimes using red or blue for selecting components and electrons). I even decided to see what other background colours may have looked like and found a much more vibrant and easier to look at for extended periods of time.

Here is an example of part of this improvement where I have replaced the grey background colour for the circuit build area with a light pastel purple colour (Hex code: #b3a1c1):

A screenshot of a computer

Description automatically generated

Another improvement I would make is to implement an area of code to make the unit graph change depending on what the calculated values are (rather than only showing a general relationship between the two units). This would be useful for my program because it would allow users to understand how changing values affects different relationships rather than just understanding the generic relationship for each unit.

Finally, to improve my code, I would also re-design the way objects are created and certain attributes are stored. For example, I would use inheritance for components to allow each component to easily have specific attributes and methods. Certain attributes could also be improved in the way they are implemented for example a wire could store all its connected component data together rather than in separate attributes.

## Overall evaluation

Overall, I think that my program meets its objectives and aims well as it allows the user to create and customise a unique circuit with many different types of components connected in various ways. It allows them to then customise each component further by changing their values which then updates and calculates each other value accordingly.

I believe I have successfully solved the problem that I aimed to solve (defined at the beginning), which was the lack of resources for A-level physics students who are studying the current electricity module. This module involves various circuit components in which my program has been designed to have, as well as showing the different ways components interact together with Kirchoff’s laws for splitting voltage into a series circuit and Ohms law shown by the unit graph, along with many other relationships.

My program allows students to practice their knowledge and skills that they feel they need to work on for current electricity and improve their understanding. It can also allow teachers to use it as an aid in teaching to help students understand better.

To fully reach my programs objectives and fully solve the problem, I would need to implement the ability for the user to create parallel circuits. To scope this out how I may be able to implement this is that I could allow the user to create an infinite amount of wires from the junction component as this would allow them to create multiple loops within the same circuit without connecting two wires between one component.