Circuit Board Designer

Authors: Joseph Spear, Kenneth Shipley, Jason Rivas

**Introduction**

The development of the theory and practical use of electrical circuits has been the most significant fundamental technology of the modern era. With it has come the development of advanced infrastructure such as lighting, heating and cooling, as well as the creation of microprocessors and computers. Our digital world would not exist without the use of electricity, and the mighty tool of circuit diagrams (also known as circuit schematics). Therefore, the Circuit Board Designer has been developed to be an easy-to-use tool to develop circuit diagrams efficiently and for those who may be lacking experience. The Circuit Board Designer offers a lot of features, but the most important is it’s simple and easy to understand graphical user interface (GUI) which allows the user to generate their own basic circuit within minutes.

While the circuit diagrams are a very useful tool, it can sometimes be very difficult to convert the diagrams, which often have ample amounts of space between components, to a compact, space-efficient printable circuit board (PCB). Therefore, the Circuit Board Designer also provides functionality to convert the circuit diagram to a physical component layout that fits optimally on a PCB. This function allows the user to be able to quickly develop their optimal layout for a compact circuit of the 21st century. This program also has the ability to give the user an image of their optimal PCB layout.

**Technologies**

The technologies used are the follows:

* VS Code editor
* QT Designer
* Python 3
* PyQT Library
* PySide2 (PyQT fork) Library
* Numpy Library
* XML Library
* JSON Library
* PIL (Python Imaging Library)/Pillow (PIL fork)

**Installation and Running the Program**

**Prerequisite: You must be using Windows to run this program properly as there seem to be SVG rendering errors in Linux.**

There are 4 dependencies required to run this program, Python3, PyQt5, Pyside2, Numpy, and Pillow. Here is how to get the program up and running:

* Install the latest version of Python3 and make sure to include python3 in your PATH.
* Install the latest version of Git and make sure to include git in your PATH.
* In cmd run **pip3 install PyQt5**
* In cmd run **pip3 install Pyside2**
* In cmd run **pip3 install numpy**
* In cmd run **pip3 install Pillow**
* Clone the repo to your desired directory by navigating to it in cmd and then run **git clone** [**https://github.com/CinnaKenToast/Circuit-Board-Designer.git**](https://github.com/CinnaKenToast/Circuit-Board-Designer.git)
* To run the program, in cmd inside the src folder, run **python3 main.py**

**GUI Description**

The front end of the program is broken up into 3 different menus: a circuit designer page, circuit converter page, and file management page. The circuit convert page is split between two pages, the converter settings page and PCB image display. When the program starts it will open on the File Management page. The file management page consists of four buttons that allows the use to create a new circuit diagram, open a previous circuit diagram (\*.circ file), and save the current project to a \*.circ file. If the user decides to create a new circuit diagram, it will switch to a blank circuit design page. Here the user will be able to design a circuit as they wish. The right side of the screen consists of tools that the user will be able to use to design their circuit. They can add up to 7 different types of components to their circuit, they can delete a component, add wires between components, cut the wires between components, change the label to their component, and change the color of the next drawn wire. Alongside the tools, at the bottom right corner of the page are zoom options for the design page. The first zoom button returns the screen to the default zoom setting, the second zoom button will zoom in, and the third will zoom out. When a user decides to add a component, a second menu will slide out from the toolbar and allow the user to pick which type of component they would like to add. When they decide on one, the program will prompt the user for a component name. That type of component will then be added to the workspace with the given name in a label. When the user clicks on the label for a component, it will select that component and allow them to move the component around the workspace. If they decide they do not like the name they gave a component, they can select the component by clicking on its label and pressing the “Label” button in the toolbar. After the user adds a second component to the screen, they will be able to connect those two components with a wire. The user can do this by pressing the one of the open circles on each component and then pressing the “Wire” button in the toolbar. This will draw a line between the components and connect them. At this time, if the user decides to move either of the components after they are connected, the line will not follow the component. They can snip the connection between wires by doing the same action of clicking the open circles of each component and then pressing the “Scissors” button in the toolbar. If the user decides they no longer want to keep a component, they can select the component by clicking on the component’s label and then press the “Delete Component” button. At the moment if a component is connected to other components and the user decides to delete said component, it will not remove the connections. The last option the user has in the toolbar is the wire color picker. When the user presses the “Paint” button, it will pull out a similar menu like the components menu, but with color options. Every time components are connected, the wire between the components will have the color they chose, until they choose another color.

When the user presses the hamburger menu in the top left of the screen, a pullout menu will slide out from the left side of the screen and allows the user to choose which screen they would like to go to. The options available are the screens mentioned previously.

The circuit convert page is where all of the PCB magic happens. The page will open with several different settings options for the user to choose from. Of the options are the number of grid points, the number of extra padding grid points, the target layout score, the image scaling option, PCB background color, and PCB wire trace colors. The first four options are what allow the program to run Monte Carlo and A\* to create the PCB layout. These options will be explained in the section below explaining Monte Carlo and A\*. The program will only attempt to find a potential PCB layout if the circuit is closed, and all components have connections. If the circuit does not meet these requirements, a message box will appear saying to fix that. If the program is unable to find a layout for the PCB, another message box will appear stating that the circuit they designed may not fit on a single sided PCB but recommends changing the grid settings and trying again. Assuming the program can find a PCB layout for their circuit, the page will change to show the image along with three buttons. One will go back to the previous page and let the user change the converter settings, the other will run Monte Carlo and A\* again and generate a new layout, and the last button will allow the user to save the image with a desired file name. At the moment, the program will crash after generating many layouts as well as trying to save the PCB image.

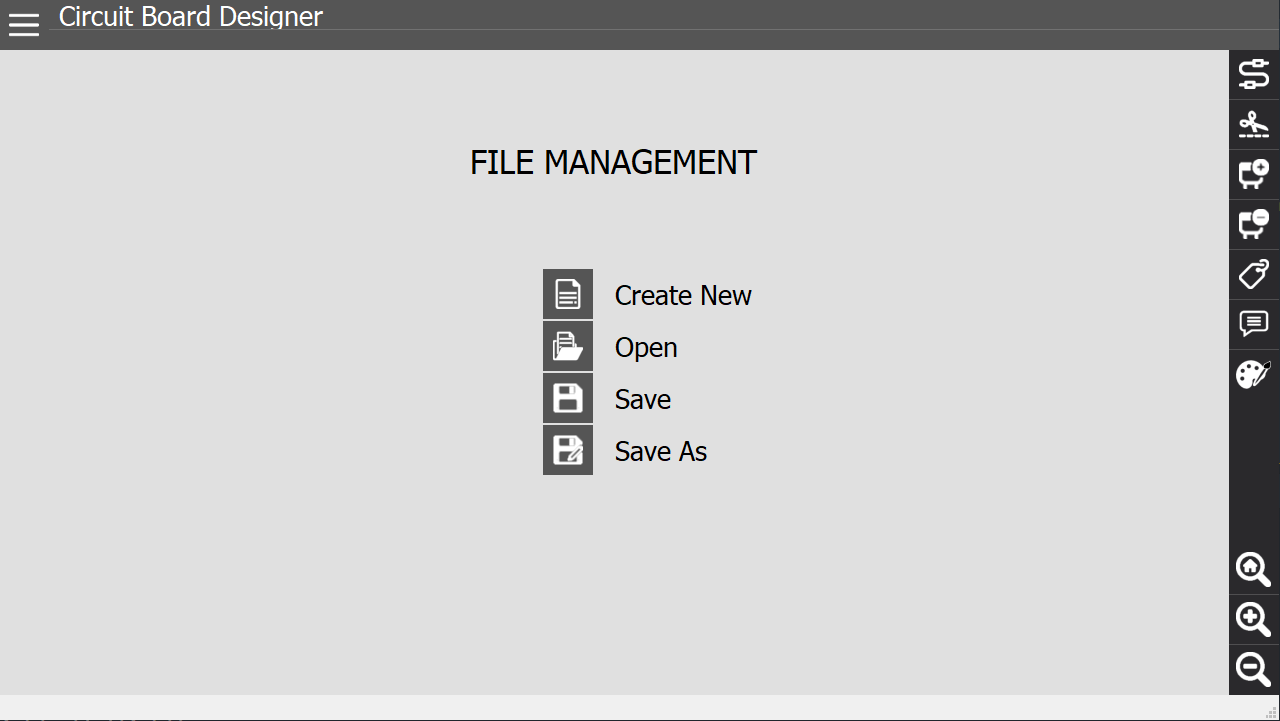


Figure 1: Start Menu

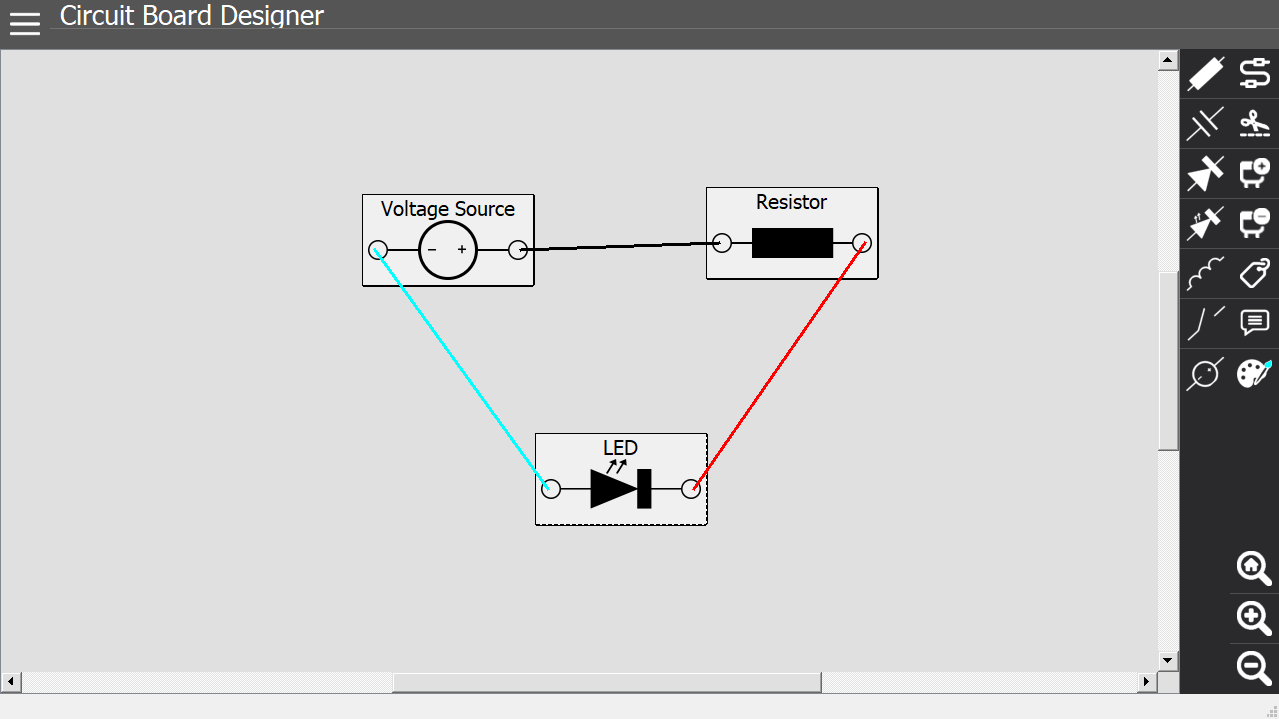


Figure 2: Workspace Example

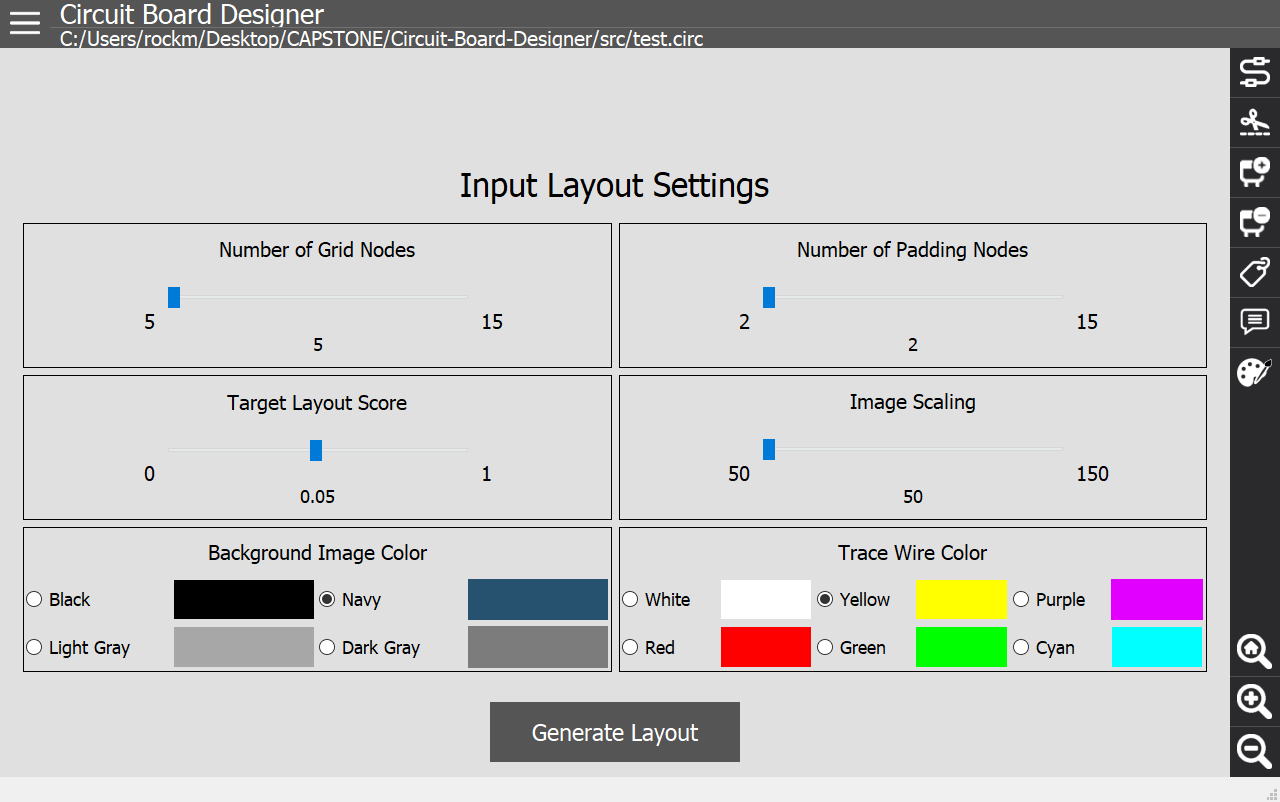


Figure 3: Converter Settings

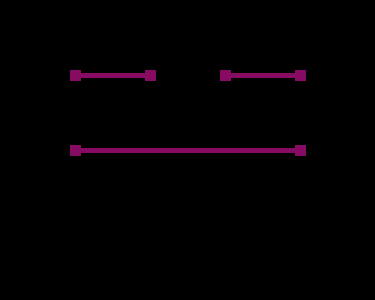
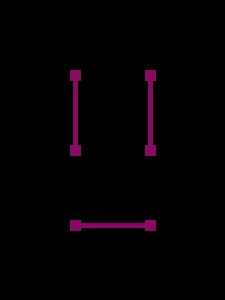


Figure 4: Two Construction Outputs

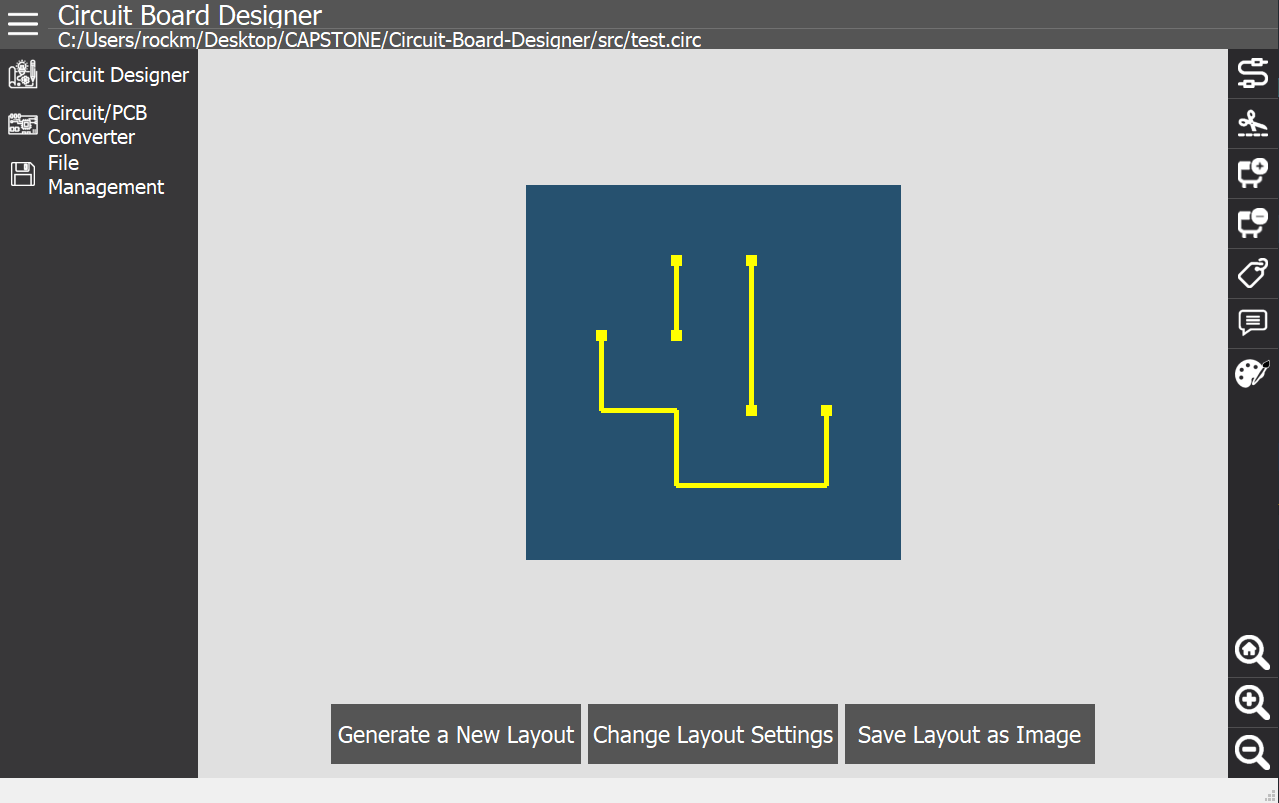


Figure 5: Converted PCB Page

**Features**

The features listed here take into account the listed features from the original features listed provided earlier in the semester. In this list, RED represents items which have been deemed unnecessary and therefore were not implemented. BLUE items represent items which would be implemented in a version 2 of this project.

1. **Circuit Builder:**
   1. GUI
      1. Implemented using PyQT or similar GUI library
      2. Canvas/Workspace
         1. The workspace has a canvas that represents the 2D grid which contains points for position data that is needed.
            1. Infinite/Finite option.

Canvas dynamically change size to enclose only the components that are in view and some amount of padding (space around the outside).

* + - * 1. Zoom in/out

UI button implementation

[ctrl] + [+] and [ctrl] + [-]

[ctrl] + [scroll up] and [ctrl] + [scroll down].(For Future Work)

Zoom option(For Future Work)

Zoom relative to mouse

Zoom relative to center of canvas

* + 1. Pallet Wheel Widget/Tools(Buttons)
       1. Can be expanded and collapsed
          1. When the center of the widget is clicked, it can expand to show the tools available or collapse to keep workspace minimal.

When expanded, each option will also expand with its options in another sub-wheel.

Sub-wheel is concentric to the main wheel.

Option expanded will be highlighted.

* + - 1. Can be dragged around
         1. Can be dragged by pressing and holding while moving mouse
      2. Tools/Functions
         1. Wire

Allow the user to draw a line connecting two components together.

Wires can be colored to differentiate from other wires.

The pin the user first clicks gets the IDs of the following pins the user clicks on which adds these ids to the list of connections in **D.1.1.2.5**

When the pin of a component is pressed, the pin will change colors indicating

* + - * 1. Wire Snipper

Allow the user to cut a wire that they had previously drawn, removing the connection they made.

This will remove connections from the data of each component in **D.1.1.2.5**

* + - * 1. Add Component

Pop up another menu of components available to add

Allow the user to pick which component they want to add, then drag it to the workspace

Will create a new component with the selected one’s type in its “Type” field

Adds it to the list of components

Will create a sprite for the component selected that will follow the mouse until the user lets go.

* + - * 1. Delete Component

Allow the user to pick which component they want to remove from their schematic

Will remove any connections the component had

Will remove the component from the list of components used

* + - * 1. Label

Allow user to label a particular component

* + - * 1. Comments

Comments will be used in order to annotate the board.

* + 1. Hamburger Menu/File Management (Buttons)
       1. Save Project
          1. This will save the project as described in **D.1**
       2. Save As…
          1. This will save the project as described in **D.1**
       3. Load Project
          1. This will load the project as described in **D.2**
       4. Create New Project
          1. Ask user if they want to save then save or don’t save (if current instance is not blank)
          2. Delete current instance of the schematic class (**D.1**)
          3. Instantiate a new schematic
  1. Database of components (Surface Mount Device only)
     1. Will have 8 types of components that the user can choose from
        1. Resistor, Capacitor, Inductor, Switch, Diode (rectifying and light emitting), Voltage Source, and Ground.
  2. Data Management
     1. All components created will create instances of objects with the following fields/member variables, all of which are described in sub-numbers of **D.1.1.2**
        1. Label, Type, ID, Position, Connections, and Paths
     2. All components are added to a list (or dictionary) as they are created, and removed from the list as they are deleted.

1. **Optimization of PCB Layout:**
   1. Grid System
      1. The layout for the PCB will be made on a grid (3D grid if we get time)
         1. All components will be laid on the vertices of the grid
         2. All component connections will run along the grid connecting the components together
         3. 2D vector (or 3D if we have time)
   2. Monte Carlo
      1. Input Arguments
         1. List of Components
            1. List of dictionaries described in **D.1.1.2**
      2. Output Data
         1. List of Components now with path data
            1. List of dictionaries described in **D.1.1.2**
   3. A-Star (A\*)
      1. Input Arguments
         1. List of components
            1. List of dictionaries described in **D.1.1.2**
      2. Output Data
         1. List of components with updated path data
            1. List of dictionaries described in **D.1.1.2**
2. **Diagram-to-Image Exporter:**
   1. Produce an image in either PNG
      1. The exporting software will take in the resulting position data from the PCB Optimization.
         1. Uses Pillow to draw rectangles for solder pads and lines for traces
      2. The program will save the data using functions described in **D.1** before exporting to an image and save the output from **B**
3. **Saving/Loading a Circuit Project:**
   1. Save to \*.circ with json formatted data
      1. This will save the current project to a \*.circ file in order to load the project if the user decides to close the program and wants to open it back up.
         1. Every project will have a base of a Schematic class
         2. Schematic will be made from Components. Components have:
            1. Label

Name given to a component by the user, only there for the user

* + - * 1. Type

Type of component as described in **A.2.1.1**

* + - * 1. ID

Number given to the component for the program to reference

Each of its pins will be: ID\_#

* + - * 1. Position

Circuit Position

Relative to top-left of screen and corresponds to the center of the sprite drawn

Component has a location on the workspace showing where the user placed the component

Just for redrawing the circuit when loading

PCB Position

Component has a location on a grid showing where the component is located on the pcb

Relative to top-left of screen and corresponds to the central location of the component’s solder pads

Sub-Positions

Location of the solder pads relative to the PCB position

Will be null if the circuit has not been converted to a pcb

* + - * 1. Connections

Contains dictionary of connections where the key is the starting pin and the values are the connecting pin(s)

* + - * 1. Paths

Starts out null.

List of points describing path for the connections described in **D.1.1.2.5.1**

* + - 1. Schematics will also have comments, comments have
         1. Comment

A string for written by the user for the comment

* + - * 1. Position

Location of the comment on the workspace

* 1. Load project from .circ
     1. This will load in the .circ file that saved a previous Schematic instance
        1. The loaded schematic object will be assigned to a new instance of the Schematic class

**Class Descriptions (Class Diagrams)**

GUI ([Class Diagram](https://viewer.diagrams.net/?highlight=0000ff&edit=_blank&layers=1&nav=1&title=GUIClassDiagrams.drawio#R7Vtbc9soFP41nnEf0pHkax9jO7eZZOvU3U37lMESltkioUUosfvr93DRzYrjxk0WdacvsTgC9HH4gHMhnd402lxwlKxvWIBpx3OCTac363jeydjx4EdKtlrS7zlaEHISaJFbChbkOzbCvFpGApzWKgrGqCBJXeizOMa%2BqMkQ5%2ByxXm3FaP2rCQpxQ7DwEW1K70gg1lo6Hjil%2FBKTcJ1%2F2XXMmwjllY0gXaOAPVZEvbNOb8oZE%2Fop2kwxlcrL9aLbne95WwDjOBY%2F0uBudTn7eJ74yf2fwnFGixCNvRPTywOimRlwxxtS6G%2BSSMhia%2FQw%2FCeTOCcR4iGJO71TeOskG%2FgLQjVaKT8RLNHv%2BpV3Am%2FECaIkNO18AIx52Sc8heZXfXmZC24Qie9ILLXmOd3bsvgurwqjXe42B1myK1tzOZqcXPmH3f0Yjho9xStRGX6zb1k3I7rKbXV0%2B5G%2FHZDvjEW6EgECWYGQAhWwUYfaP4ifLpTMChwSpJYVwjJBSYynjDJu9DLHsR0sCY5bhcPMza0GZQXKilBD11RYguCzKGEx7KBGHZ3BZJqLOoOZLVDq8CUsLlEVC%2FoaCG0NWeqvMZzExNc1F0XRCpol8r%2BFnGVxcAF%2FzVF5O8mEYLGWWIElOPJxqxBp49DqVpygICBxaBmFgLpYWD%2BmwXozqrC7bHTVTxeTrlLGNNeKeXhnaZLk%2BvkZaC0yT%2B%2FvSUzE%2FX3Xki5hg%2Fa%2FTcvzBHCoJhNw%2BuwgypIACTxnKakBemCwRdnZmTjwqTjJ7OOBjbIwQbqFfaKZH6MI66eEpV%2BKp6%2F6SVa%2BCvQzZUjut9ZHI3HUh%2FN5m%2BAm3n0jsz6AAFMscDkE64AUP%2FIFrTTq5EqUBfcqsA6R44g94ApK64j8NYpDfI2WmNoHI0MFVy1QisTxMWsBpSWQSxZh%2B0g0T%2BbGR4bVBX%2Ftg6IY8RadUNJv%2F5jgFjBYIplyDAZFO7As0EOLkJy2wLoSLAwpvkFx1o3QRsf81VmFY7SkuCX4PoM5nLYToLaXF5QEmP8l8wpdKk8xjTFVYv2sfVvreFMssuSGxWAzIU6ZfQamsBKuIhS2YGEmLMmSM87hZJFZnIN4vBoS7wFz2P8RPdXJn5lKD01MKmimEUzA8OIrqjJjsA9QkKxgMkwm0PVM%2BRxFhMoc4iWmD1j2Ci%2FWIqKyUvHtajYsT20BBrypiEx27ALD%2BS34Fqrkb0d5btLkKt2%2BKT%2BWmb9%2BLltXsn7DoREik20Mi87LhBw8mJzcC%2FJzXpvzc7D3wALu3urf%2F0VS7pzxyM5Ssx7wXMoAG3jgE7Yxysgj%2BJ%2FAKboS2JJijkoStohbRUStHkso4wd6gPJxiCK5QcbLNDmwdt4GbuP7LSvXOKojqyaQYDZ%2BOWDQh%2FMHK%2BmRJijOh1jdo6pyO8xGQXAN9XS2Zc5SWzFXwPFJHmbWgYAppg2ffKnswfFLmxmeNzhsZnjDQdPMcMdvZWb0GzrFcKQvTJFxsWYhixE9K6UTlZDBslc502WdaybVr7T1NxZia1SMMsHqusQbIr7I5u8HpvS18ma2MT2rwtYU9upf4n1W%2BxxTJMhD%2FVrXU4o0TedMncTFrI28QW3Wxrs2X8oy7mPTrHoP68U9aceo0ZOa2WJEx092frWuZlTuTn8cnMrbc1BaUuZ%2Fk%2FMQ59fz3KEunqt1pObl4LQ6z04rzCbfVhrJYkEGWSibqdJBOujJeM6wNnOQJ1gPWeAH%2BVVZvG4%2BoT%2FLuXG%2FzpTh4EjO9dwP9Z4G%2Ff%2BYc4MXcY4lOK5xzK0T0Huacm6VcAX9DlGuSrgK%2F16dcu6PMm5gkXKDD%2FXDqX8s5UpPOafcbk9vTblhg3JTwv2MSJAThjiYjs4Mp2AlSG%2FXmVKUplJEUMgRODtOd76FQw2MRKf3rsHW%2BvH3uCYCLxJ1D2D2CF7K65sNzo4%2Be02rwc0nr2o17Kr99WITT63p1gQny))

The front end of the system works off of four different classes, the backend Schematic, the MainWindow class, a Component class, and a Widget class. The schematic class is explained in the section below, but the other classes were created in order to keep track of all the components in the workspace and have it interact with the backend. There is certainly a better way to keep track of all the components and its data. The MainWindow class keeps track of all the data needed to run the GUI since it is the driver of the program. It contains variables that holds the UI of the program, the scene that runs the design workspace, a list of components, a list of connections, and a schematic that ties everything to run Monte Carlo and A\*. There are several other miscellaneous variables that are stored in order to run the various functions inside of the main window. Below is a list of functions that MainWindow has as well as their functionality.

* \_\_init\_\_: Initializes all of the variables and connects the buttons in the UI to their functions.
* updatePosition: Updates the positions of the components when they are moved around the Scene.
* printSchematic: A debugging method that prints out all the components in the schematic along with their connections.
* addComponent: Adds a component to the scene and the schematic
* loadComponent: Only called when loading in a schematic from a file. This loads in the component from the file.
* deleteComponent: Deletes a component from the scene and the schematic.
* addConnection: Adds a connection between two components and draws a line on the scene connecting the two components.
* changeLabel: Changes the label of a selected component.
* zoomIn: Zooms in on the scene.
* zoomOut: Zooms out on the scene.
* zoomHome: Sets the zoom to the default setting.
* changePenColor: Changes the color of the wires that are added.
* clearSchematic: Resets the entire scene and empties the schematic for a new project.
* fileOpen: Open a \*.circ file to be loaded into the scene and schematic.
* fileCreate: Call the clearSchematic function to reset the scene.
* fileSave: Calls the schematic’s save function and saves the schematic to the current file. Calls fileSave if a file does not exist yet.
* fileSaveAs: Prompts the user to choose a location to save the schematic file.
* toggleMenu: Toggles the left slideout menu to change screens.
* toggleTools: Toggles the right slideout menu to view components and colors.
* updateSliderValue: Updates the label beneath the converter setting sliders based on the slider position.
* setupMonteCarlo: Passes the schematic the required values from the converter settings page to run Monte Carlo and A\*.
* saveImage: Saves the PCB layout image to a user chosen location.

The Widget class is a simpler class that contains the UI for the component widget that is seen in the scene when a component is added. It has an \_\_init\_\_ function that initializes the widget to be added to its Component wrapper class. The main function this class has is the addLeftButtonPos and addRightButtonPos which adds the pin that was clicked to a list of button events. This list of button events maintains a length 2 and keeps track of the last two pins that were pressed in order to easily connect the two components when a wire is added between them. The setImage function sets the background image of the component to their respect SVG image.

The Component class is a wrapper class for the Widget class. This allows the widget to be added to the workspace scene. It only has an \_\_init\_\_ function that initializes a Component object. When the object is initialized, it creates a Widget and adds itself to the scene and the Schematic object.

Backend ([Class Diagram](https://viewer.diagrams.net/?highlight=0000ff&edit=_blank&layers=1&nav=1&title=ClassDiagrams.drawio#R7V1bc%2BK4Ev41VDEPSWEbMDwGMpecmplNHaZ2Z%2BfFJWzFaMe2PLK4ZH%2F96uK7HSAJIJEhDwHLt49W6%2BtWq93uWNNw85GAePEFezDomD1v07FuO6Z5NRr32AdveZQt4%2FFANvgEebLJKBpm6F%2BYNqbn%2BUvkwaRyIMU4oCiuNro4iqBLK22AELyuHvaAg%2BpdY%2BCnd%2BwVDTMXBLBx2F%2FIowvZOhqUjv4Ekb%2FI7mz00j1z4P70CV5G6f0iHEG5JwTZZdJDkwXw8LrUZL3vWFOCMZXfws0UBlysmcTkeR%2Be2JtDJjCi%2B5wwG4fuzU9n8Zf55RMh36Ht%2BfdX6VVWIFimouiYw4BdbxJzyPQxldDw15LjnISA%2BCjqWDdsby%2FesP%2BsUfxa3n5FcSz39Uv7KNzQKxAgPz3PZYAhKa7Jvvnpp7jzPGuYuQvIxIjcbA%2F7cfP60awtrrctCAefaVl2H%2BPpW77oxwbwgZZ%2BbfPa%2FNjpH1%2Fu%2F%2Fj6%2Fus3Z%2Fr5ZjZ7P5NndOxJQon8PsVhzNQmotMAJEnHvs0FQfb5scfD7mbAki2g9cIbPolW7tEHbMz4IUM6mDCwHNtAG3jukhCHLKPESVxMoDwese7WBF%2BMIicOgAt5vzoecmm91wcTAXfK%2FzO5Fj1%2FaqUUJgthJsoAJVRc5Fb2OkdqTsWtSanvTwuQGRDgrFPrxo99CDCginqVDQongJHPPjSBFDncZXGSmOla0hgGJ%2B4qJ6GASEAx8DwU%2BdoNTAbQh7TMGgo7jw2%2BFSQUeg4KmQ%2FnzH3HxQFOrUK3IAjx%2F50eIAVCJ2ROtjxVsIQenVuHmjAXWkclrOOkhI3elq7PUKdf9FAAefz93efrO7H5LJlq5Po6DooQdZyuIrEmjIZCzOYajgtIgBljEhBCNvVIulVSF93fxq18x6mRD0HI5lCTaJ7ER9muEPQ7CWWF2SRRSR8tI%2FRryUdmOptwkNdFXgprjnGgGlaoD6h8OtxNsm%2FC8VXeiWy0FD3YLfpSC3AeDCCFrfjyPlWHjo%2FErdCKebYmppUjFpMvnFDEpzVdvpXjzaddqWnVZ0YLPWaMAjCHQUXOEiqPUSlXBjmOsuliCaXhSBlLrEW76VRkrw45gSFewTMFD%2BIYRp5DsVOfqwvVNnLsfMvUAXEq7geCw7PBnJSZziksGOORlvHIWpUjjgkjMAcEQRk3ZY1%2B0tULXKgVtNQdCFOLFmrjCggLUEeWaVwE184TVkC15crZtY5cuUQ9AtbqFY4xN9evDIgIBmvSdQlYwe4DCqATsbmnckkxNSJrgpg%2FrBmwAANPIzgRFtyK19BzYnfuJDGmObPy5Zqml6sqfJ%2F644YwpDudcCXYTF2xrUAgokCYdlt7XPs5DQGRh0P0L2QTm0e8pOq5mIf8EBMrg9RcotMKXsNxluD06NhS3LIbgo2DeNhSufT4wODLhV2xeP3E6Gh%2BUTXEXRC4ywBwa8eDnRXUChelyDJyZLQ550S5%2Bq%2FGeSIoFIxXZRCNe1WulvD5OoctlkzUEwuf3orgUtLlTrEU0wPOBLYQi9mazS32Q2BW7m1y2SM2sm5kQtetSPmapOldt%2FKeE%2B5nshHGs92YUxewFi6KNO%2FPMNPtDyBEAc8Y%2FASDFeRXZTsWNAz4Qfm9yxluWboaXy3blJrSjLePEIeQkkd2SLq3SDFMMxOvzL7cXhd5flaWvrgo5fiN7LQRpLmFfn7xIsmOfUnz7J6Rc2fqnHPXiDefYc4dl%2F5NR2GehiACeVRp8f7EU5hlyD0w1SkrItDH%2FX%2BxVlCxLaoMiISECWJaDgpUykTErVhVQDpFMEpuchPciTtu5Tt4%2Fk9tYWz250ddZMUBpmtjSErs7HM3svCopDQZKc2YRaYuVgf4W0yZeHXKRZNx5CJHeeBfBNciuDL5SG1jI6wUJlQyymXUCEUxmy2J%2F87PNTs9m1mqSxCRS2y5xLql71rMj6qLfjos8XFU6Sh8Ao9qk5L2YZc))

There are four major classes that hold up the backend of the program. They are the Schematic, Component, PcbGrid, and GridNode. Schematic is the class that holds all of the information for a schematic. It keeps track of every component, every comment, all Monte Carlo/A\* information, saving and loading, pcb image creation, and so on. Every method in the class is made to hopefully make using the program easier as well as making connecting to the frontend (GUI) easier. In the class diagram, from the top to right after the load method, these are all for importing, editing, and exporting information regarding the schematic. From the load method down to the trim\_pcb\_layout method, is every function that prepares and executes our version of Monte Carlo. The last few methods are used to create the image based on the output nodes from the monte\_carlo method, stored in the paths field for the Schematic class. The Component class keeps track of everything to do with its pins and location. It allows us to easily instantiate a component and keep it updated as a user messes with them on the GUI. The PcbGrid and GridNode classes are used to make the paths, using the A\* pathfinding algorithm, that our Monte Carlo method uses to make a layout.

Monte Carlo and A-Star

Our version of Monte Carlo performs random placement of every pin, belonging to components, in a schematic. The placement is within some set number of grid points (e.g. 3x3 grid). The method then calls a function that controls the A\* method by feeding it pairs of starting points and goal points as well as updating the PcbGrid object’s obstructions field. The obstructions have every path appended to them when they are made by A\*; this is so A\* won’t create two overlapping paths. The A\* algorithm was written following pseudo code provided by a YouTuber named Sebastian Lague who teaches, goes through exercises for, computer science areas using Unity to visualize the topics. The algorithm is given extra spacing on top of what the components were given; the components are also mapped to near the middle. This allows A\* to go around the pins if needed. However, if A\* cannot find a path, the monte carlo method shuffles the pins turn order around so that other pins can go first to see if that makes a difference. The layout is scored soon after that to see how well it has done and if it is better than the best yet, it becomes the new best layout. This score is based 70% on how long the total path length is and 30% on the total area taken up by the components and paths. Our method continues to make layouts until the score of the best layout yet reaches/surpasses some target score or until the maximum number of iterations is reached. Once the method exits, a layout is hopefully in the paths variable for a Schematic, or it was not found and some tweaking to the parameters given is needed.

**Future Works**

There were many things which would have been a huge enhancement to the Circuit Board Designer v1.0 had there been more time. Unfortunately, many of the ambitious features initially desired were not able to be implemented in the time allotted. In the event that Circuit Board Designer v2.0 were to ever be implemented, here would be some of its features.

Pallet Wheel

The most ambitious feature to be discussed, the Pallet Wheel would be a GUI feature which would be a shortcut to the menu options found in the final design. This feature would include nearly all of the same features as the design menu (i.e. components, colors, wires, etc.) but would be a moveable widget with a click-and-drag functionality for ease of function selection. The Pallet Wheel would also have the function of being able to collapse into a smaller moveable widget which would save space on the canvas. Once clicked again, it would then expand showing all of the buttons previously mentioned. This feature was scrapped from v1.0 early on as it was determined to have very little priority over the menu bar and the other important functions.

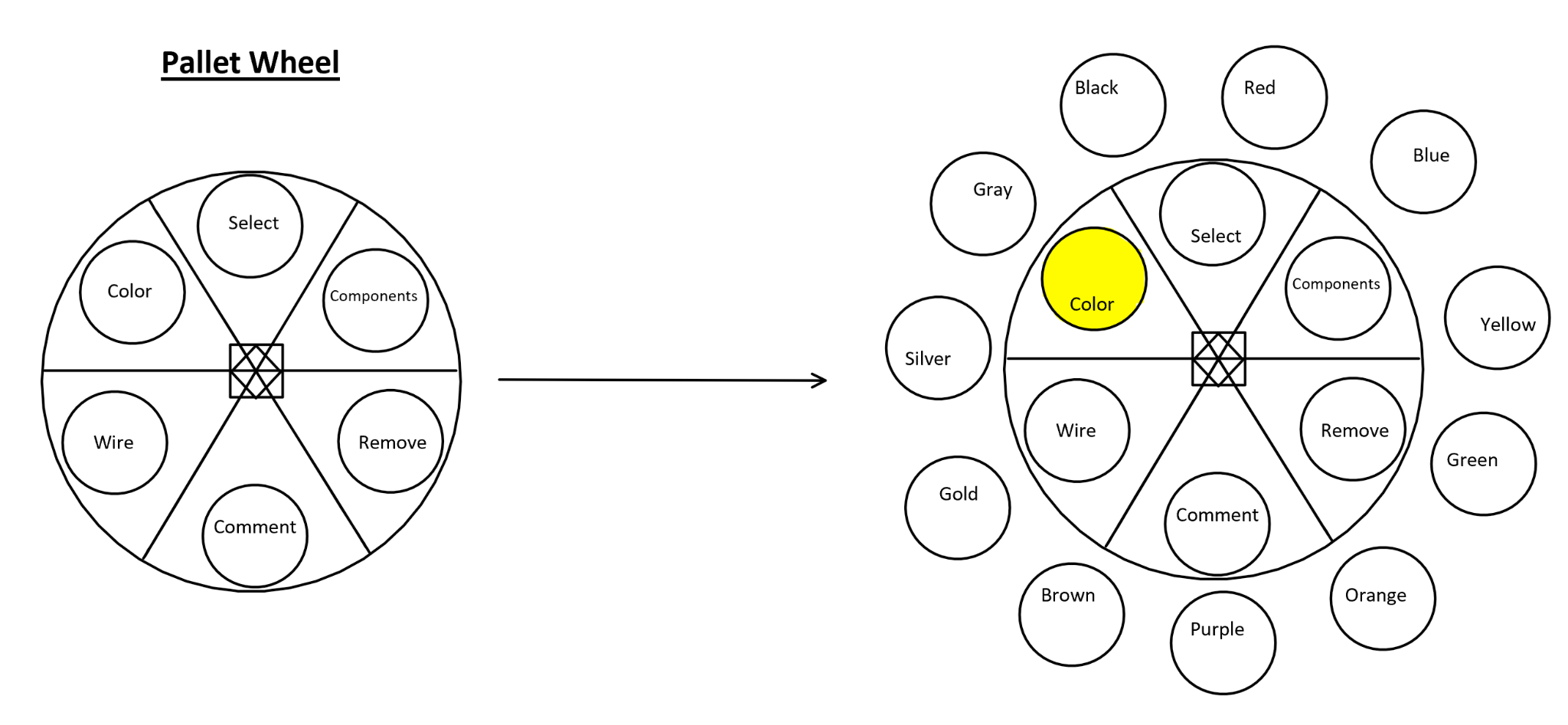


Figure 6: Pallet Wheel Idea

Transistors and Ground

A fundamental component to modern electronics is the transistor. The transistor is used in a plethora of applications ranging from large scale industrial work to smaller applications such as the computer. Therefore, the transistor is a must have for future versions. The inclusion of an NPN and PNP transistor was discussed but due to the increased complexity with needing three terminals instead of two, the idea was scrapped for v1.0.

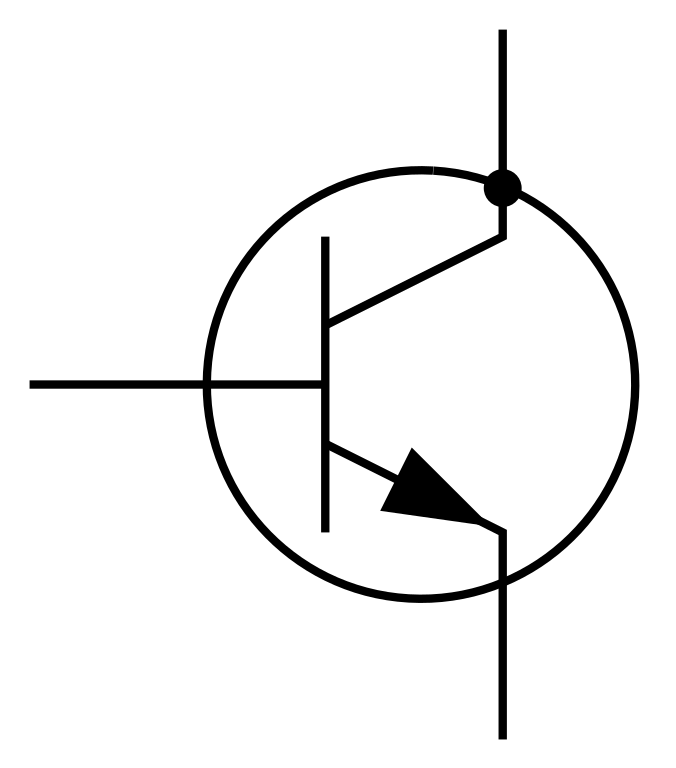


Figure 7: Transistor Symbol

A similar thing could be said about the Ground component. This component was left out due to it being a single pin component. The main difficulty of a ground component is that it represents a ground wire that multiple components would connect to for ground. Having all the components be a two pin component vastly simplified the PCB conversion process.

Follower Wires

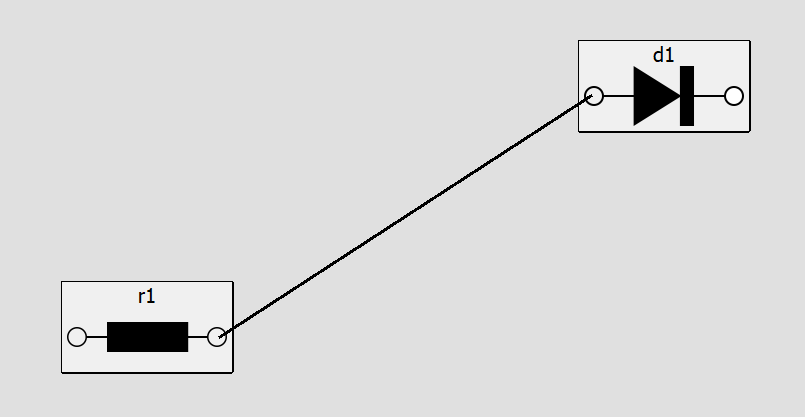
“Follower wires” would be the feature in which wire poles would essentially “stick” to the component poles and be dragged along to connect to the component wherever it went. The wire would stretch and move around relative to the components it was attached to and where they are being moved to. The idea of follower wires was pushed off due to time-constraints. 

Figure 8: Wire Following

Real time Image Update on Button Press

One big feature that we wanted to add to the program was a real time update to a component’s image when one of its pins is clicked. This is why the background image is a SVG. The idea was to click on the pin and the open circle would get filled in. This would be a visual indicator to the user that the pin was clicked. This feature could not get implemented due to time constraints.

**Known Bugs**

* Program crashes when generating PCB layouts or saving the layout - The cause for this is unknown as the program crashes with no error and there is no way we can figure out how to figure out what causes the error.
* Can double connect the same two pins - Since the pin connections are stored as single QGraphicsLine objects, when two components are connected multiple times, the pin connection is overwritten, losing access to the old line. This could be resolved by not allowing the same pins to be connected together more than once.
* Lose access to connection lines when multiple connections are made to the same pin - This is caused for the same reason as the bug above. This can be resolved by storing a list of pin connection lines.
* Labels on the PCB image may not appear in the right location - The cause for this is currently unknown.