Circuit Board Designer - Project Design

Joseph Spear, Jason Rivas, Kenneth Shipley

Design Overview

- Three Primary Features
    a. Graphical User Interface with PyQt
    b. Optimized Path Algorithm with Monte Carlo and A*
    c. Diagram Image Converter
- These features are all integrated together to provide the user with the ability to design their own schematic and convert it from a schematic to an exportable image.
- Along with these features, there are file management features for ease of storage.

Graphical User Interface

- PyQt will be the driving program for developing the GUI. The QTDesigner software will also be used in implementation of the GUI.
- Simplicity is the desired goal of the GUI so reducing the clutter seen in many other programs is key.
- Three primary portions: Workspace, File Menu, Feature Menu.
  - Along with the feature menu, a movable pallet wheel in the workspace will be implemented if time permits as an advance feature.
- Each of the 10 components will be imported to PyQt as an image which will be stored in each of the individual component classes.

Workspace

- Plain White background with faint, gray gridlines
    - These gridlines will increase/decrease in interval with an increase/decrease in zoom level respectively.
- The grid points will intersect at grid points which will be indexed in a 2D array. Each grid point will also be used as the position data for where certain components/wires will be.
- A zoom feature will allow the window to zoom into a specific point on the workspace. All elements will increase in size proportionally.
    - A "Home Zoom" button will be added to bring the zoom back to the start zoom level.
- Hamburger and Features menu buttons will located on the top left and top right respectively.
- When an element is placed on the workspace, a prompt will be set up to ask for the label.

File Menu

- The File menu will be a hamburger menu icon in the top left of the workspace.
- Clicking will pull out a side tab ⅓ the size of the screen and include all file management features.
- File Management features are:
  - Save/Save As
  - Load
  - New Project
- If the user decides not to perform a file management action, clicking the workspace will cancel the pull out bar.
  - This is the case for the feature list as well.

Feature Menu

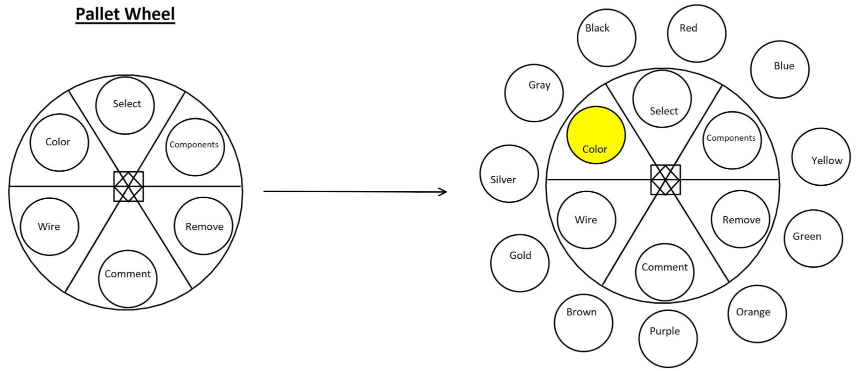
- The feature list for this project will contain all of designing tools. This will be an exhaustive list of designing features.
- Features will include:
    - Component
        - Resistor, Capacitor, Inductor, NPN Transistor, PNP Transistor, Switch, Voltage Source, Switch, Rectifying and Light Emitting Diode, Ground.
    - Color
        - Black, Red, Blue, Green, Purple, Pink, Brown, Yellow, Gold, Silver
    - Wire
    - Wire Snipper
    - Remove
    - Relabel
    - Comment

https://www.theaccountingroom.co.nz/assets/frameworks/wf/images/tools-icon.png

Pallet Wheel

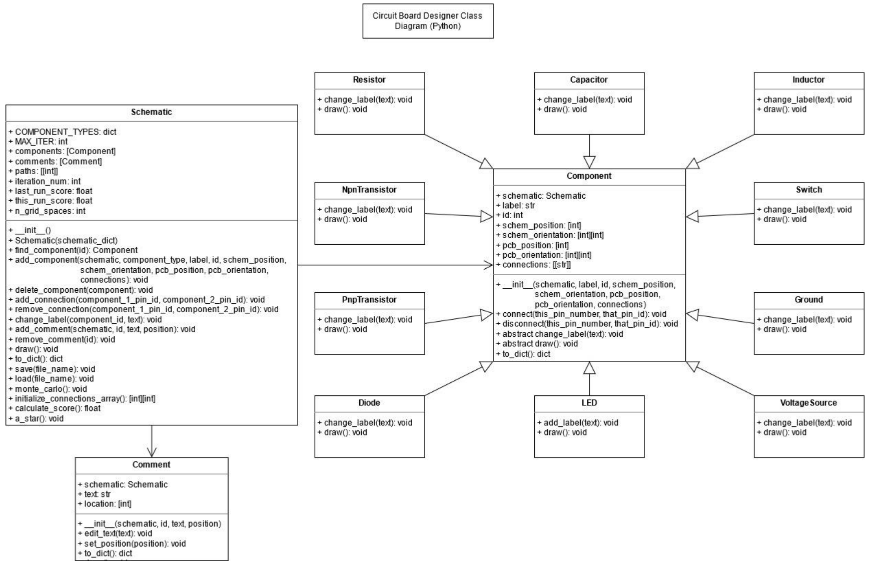
- A Pallet Wheel is a high priority, but in the event implementation fails, the feature pull out tab will be the alternative.
- Features included in the Pallet Wheel will not be exhaustive, and only include 6 of the major buttons, taking out the rebalbel button from the pullout menu.

- The Pallet Wheel will have a **central icon** that when clicked, will **collapse** the entire wheel into a very small, out-of-the-way circle. When click-and-dragged, the entire pallet wheel will move to the desired location on the workspace.
- When **Component** or **Color** is selected from the wheel, 10 bubbles with all of the components or colors will appear circumscribing the pallet wheel.



Classes

- Three main classes will perform the desired operations: **Schematic**, **Component**, and **Comment**.
- **Schematic**: Contains an array of Component objects and an array of Comment objects. Functions to add things to the workspace are found here.
- **Component**: Stores each individual component. It contains members to store names and position/orientation data.
- **Comment**: A small, saveable textbox which can be placed onto the workspace grid.


Circuit Board Designer Class Diagram (Python)
Schematic
+ COMPONENT_TYPES: dict
+ MAX_ITER: int
+ components: [Component]
+ comments: [Comment]
+ paths: [[int]]
+ iteration_num: int
+ last_run_score: float
+ this_run_score: float
+ n_grid_spaces: int
+ __init__()
+ Schematic(schematic_dict)
+ find_component(id): Component
+ add_component(schematic, component_type, label, id, schem_position, schem_orientation, pcb_position, pcb_orientation, connections): void
+ delete_component(component): void
+ add_connection(component_1_pin_id, component_2_pin_id): void
+ remove_connection(component_1_pin_id, component_2_pin_id): void
+ change_label(component_id, text): void
+ add_comment(schematic, id, text, position): void
+ remove_comment(id): void
+ draw(): void
+ to_dict(): dict
+ save(file_name): void
+ load(file_name): void
+ monte_carlo(): void
+ initialize_connections_array(): [int][int]
+ calculate_score(): float
+ a_star(): void
Comment
+ schematic: Schematic
+ text: str
+ location: [int]
+ __init__(schematic, id, text, position)
+ edit_text(text): void
+ set_position(position): void
+ to_dict(): dict
Component
+ schematic: Schematic
+ label: str
+ id: int
+ schem_position: [int]
+ schem_orientation: [int][int]
+ pcb_position: [int]
+ pcb_orientation: [int][int]
+ connections: [[str]]
+ __init__(schematic, label, id, schem_position, schem_orientation, pcb_position, pcb_orientation, connections)
+ connect(this_pin_number, that_pin_id): void
+ disconnect(this_pin_number, that_pin_id): void
+ abstract change_label(text): void
+ abstract draw(): void
+ to_dict(): dict
Resistor
+ change_label(text): void
+ draw(): void
Capacitor
+ change_label(text): void
+ draw(): void
Inductor
+ change_label(text): void
+ draw(): void
NpnTransistor
+ change_label(text): void
+ draw(): void
Switch
+ change_label(text): void
+ draw(): void
PnpTransistor
+ change_label(text): void
+ draw(): void
Ground
+ change_label(text): void
+ draw(): void
Diode
+ change_label(text): void
+ draw(): void
LED
+ add_label(text): void
+ draw(): void
VoltageSource
+ change_label(text): void
+ draw(): void

Data Management

- Due to the scale of the project, a database will not be needed. Since there are only 10 components to choose from, and potentially a limited number of components allowed, all data can be stored internally.
- This will allow for the data management to be kept simple.
- If there ever comes a point where more components would be added to choose from, the data management could be reconsidered.
  - Additional components is on the list of potential features to be added if time permits, but is lower in priority.
- The “database” of components is two things: a .json file named “types.json” and a sprite sheet called “component_sprites.png”.
- The .json file will contain the following fields
  - the central sprite position for the component
  - the positions of the pins, for connecting wires in the schematic builder
  - the sprite index, for getting the sprite for this component from the sprite sheet
  - the positions of the solder pads relative to the center of the component
  - the size of the solder pad, the dimensions of a square
- The .png file has square images that are of the components.
- Both are loaded on startup.

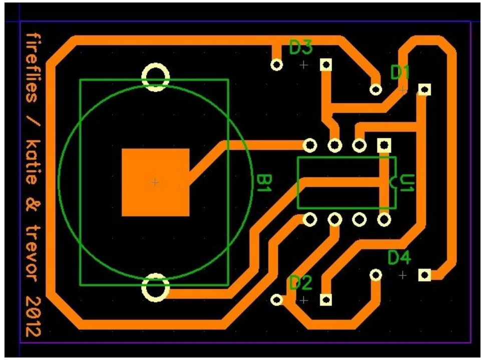
Save and load

- Save
  - A Schematic dictionary is created, which is a concatenation of the following lists and dictionaries:
    - “Components”:
      - List of Components where each component is a dictionary with the field names as the keys, and their values as the values to the dict.
    - “Comments”:
      - List of Comments where each comment is a dictionary with the field names as the keys, and their values as the values to the dict.
    - Monte Carlo dictionary where each field used for Monte Carlo is the key, and their values as the values to the dict.
  - The dictionary is converted to JSON object and written to file.json using the json module(file is a string chosen by the user)
- Load
  - Schematic dictionary extraction
    - JSON file loaded into dictionary which is sent to a constructor for the Schematic class that unpacks and sets its fields to the ones just loaded.

Monte Carlo for Component Placement

1. If on a first run
   a. Initialize the path list to an empty list, randomize the component placement, check for overlaps on placement and redo if necessary
   b. If not on a first run, randomly place a non locked component, check for overlaps on components and path
2. Call the A* function to find paths between the connections without overlaps
3. Calculate the score of the board to determine if it is a good layout
   a. The score will primarily be dependent on the total lengths of all paths and the surface area of the board used. These two factors can be weighted based on importance and a score will given (ie .75 * SA + .25 * PL)
4. Accept or deny the layout based on the score in relation to a previous score, if one exists
5. If the layout is accepted
   a. Lock the shortest path component and the path itself, then run to find a new layout of non locked components
   b. Otherwise rerun Monte Carlo using the last iteration's locked components and path
6. Once all components have been locked into place, that will be the best layout until a new one is found. Then Monte Carlo is run again to find a new layout.
7. The function will end when a better score cannot be found after a certain number of runs and plateaus

Diagram Image Converter



- The main purpose of this function is to take a circuit design that has gone through Monte Carlo and A* and an optimal PCB layout has been found. These functions will result in components having particular locations on a PCB as well as the optimal paths chosen.
- Using this data, a PCB can be drawn using the Python Pillow library. The function will output an image that consists of a black rectangle representing the PCB with the following inside the rectangle
  - Red lines representing the traces (these lines will only be able to change direction at 90 degrees for simplicity)
  - Yellow squares with a hole to represent the pin holes for a component to be soldered onto
  - Green labels for the components, green + sign for any components that require directionality (capacitor), and potentially outlines for a component (diode)