Chapter 8 - Analog Circuit Simulator

Design, Features, and Specifications

Analog Simulator allows the design of basic electronic circuits and analysis of voltage, current, and power in the circuit. Spice is the standard library for analog circuit analysis and fortunately there is a Python library called PySpice.

Program Design & Architecture:

- ✓ Modify the Microwave Simulator to create the Analog Simulator
- ✓ Object-Oriented Programming (OOP) - class abstraction, inheritance, & polymorphism
- ✓ KISS - Keep it simple, silly
- ✓ DRY - Don't repeat yourself
- ✓ SOC - Separation of concerns
- ✓ Python Modules & Packages
- ✓ User-interface
    - ✓ TopFrame class
    - ✓ File menu frame
    - ✓ Settings menu frame
    - ✓ Rotation button
    - ✓ Help menu frame
    - ✓ Analysis button
    - ✓ Left Frame Class with Circuit Component Menus
    - ✓ Canvas class
    - ✓ Mouse class
- ✓ Circuit components
    - ✓ Components
        - ✓ Resistor
        - ✓ Capacitor
        - ✓ Inductor
        - ✓ Transistor
    - ✓ Sources
        - ✓ Voltage source
        - ✓ Current source

  - ✓ Wires
    - ✓ Straight Wire Class
    - ✓ Segment Wire Class
    - ✓ Elbow Wire Class
  - ✓ Grid class
- ✓ Analysis
  - ✓ Raw data display
  - ✓ Line graphs

Key Technologies Needed:

- ✓ Analog analysis library - PySpice

Project Setup

Language: Python 3.11
IDE: PyCharm 2023.2.1 (Community Edition)
Project directory: D:/EETools/AnalogSimulator
Graphics library: CustomTkinter (https://customtkinter.tomschimansky.com/)

External libraries:

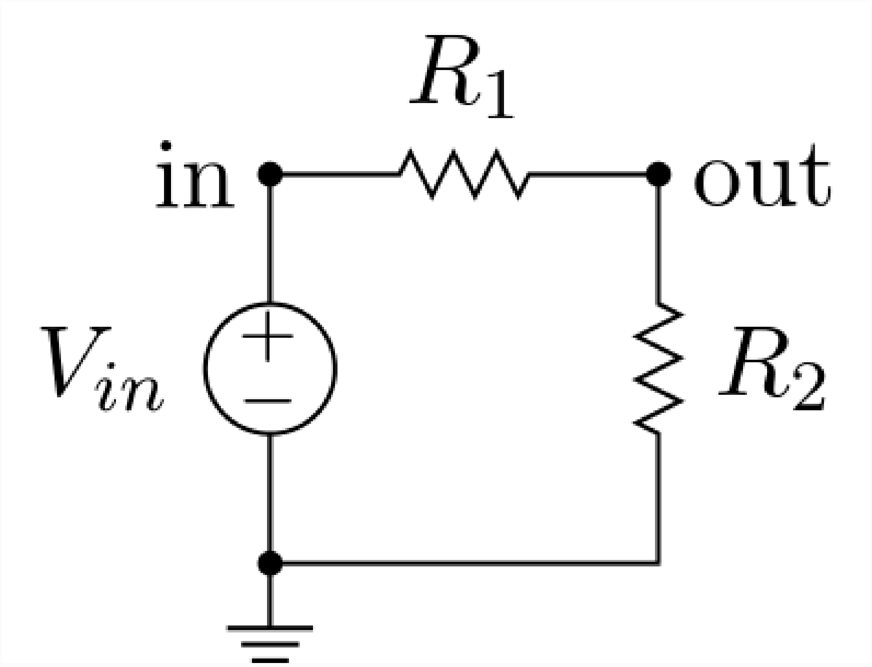
- ✓ pip install customtkinter
- ✓ python.exe -m pip install --upgrade pip
- ✓ pip install ctkcolorpicker
- ✓ pip install tkinter-tooltip
- ✓ pip install pyInstaller - Create .exe file
- ✓ pip install matplotlib
- ✓ Add images and icons directories to the project.

Analog Analysis Library

- PySpice and `Ngspice`
- ✓ pip install `pyspice`
- ✓ `pyspice-post-installation --install-ngspice-dll`

PySpice Evaluation

[Home Page](#)
[Documentation](#)
[GitHub Repository](#)



Sandbox/pyspice_voltage_divider.py

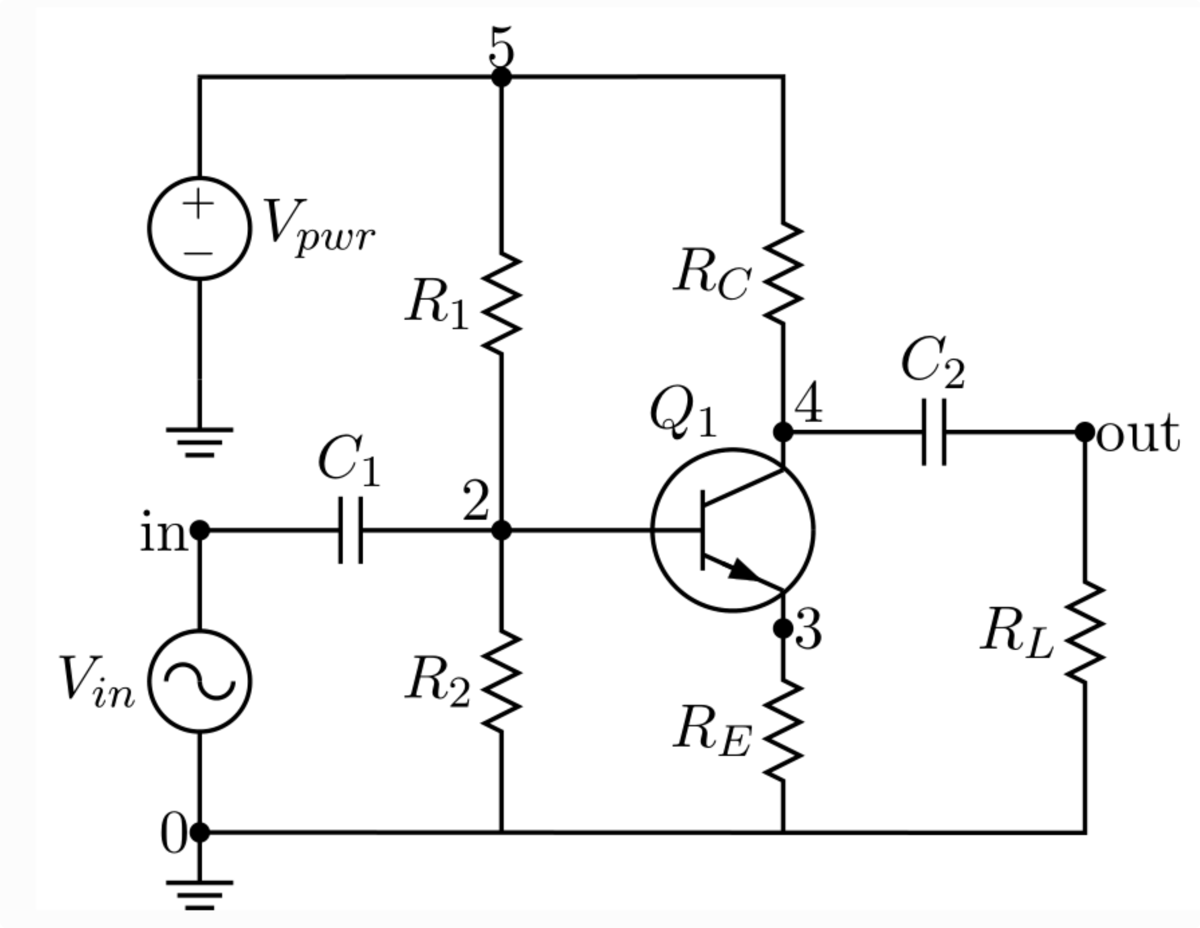
```python
import PySpice.Logging.Logging as Logging
logger = Logging.setup_logging()
from PySpice.Spice.Netlist import Circuit

voltage_unit = u_V = 1
resistance_unit = u_kΩ = 1000

circuit = Circuit('Voltage Divider')

circuit.V('input', 'in', circuit.gnd, 10*u_V)
circuit.R(1, 'in', 'out', 9*u_kΩ)
circuit.R(2, 'out', circuit.gnd, 1*u_kΩ)

simulator = circuit.simulator(temperature=25, nominal_temperature=25)

analysis = simulator.operating_point()
for node in (analysis['in'], analysis.out): # .in is invalid !
```

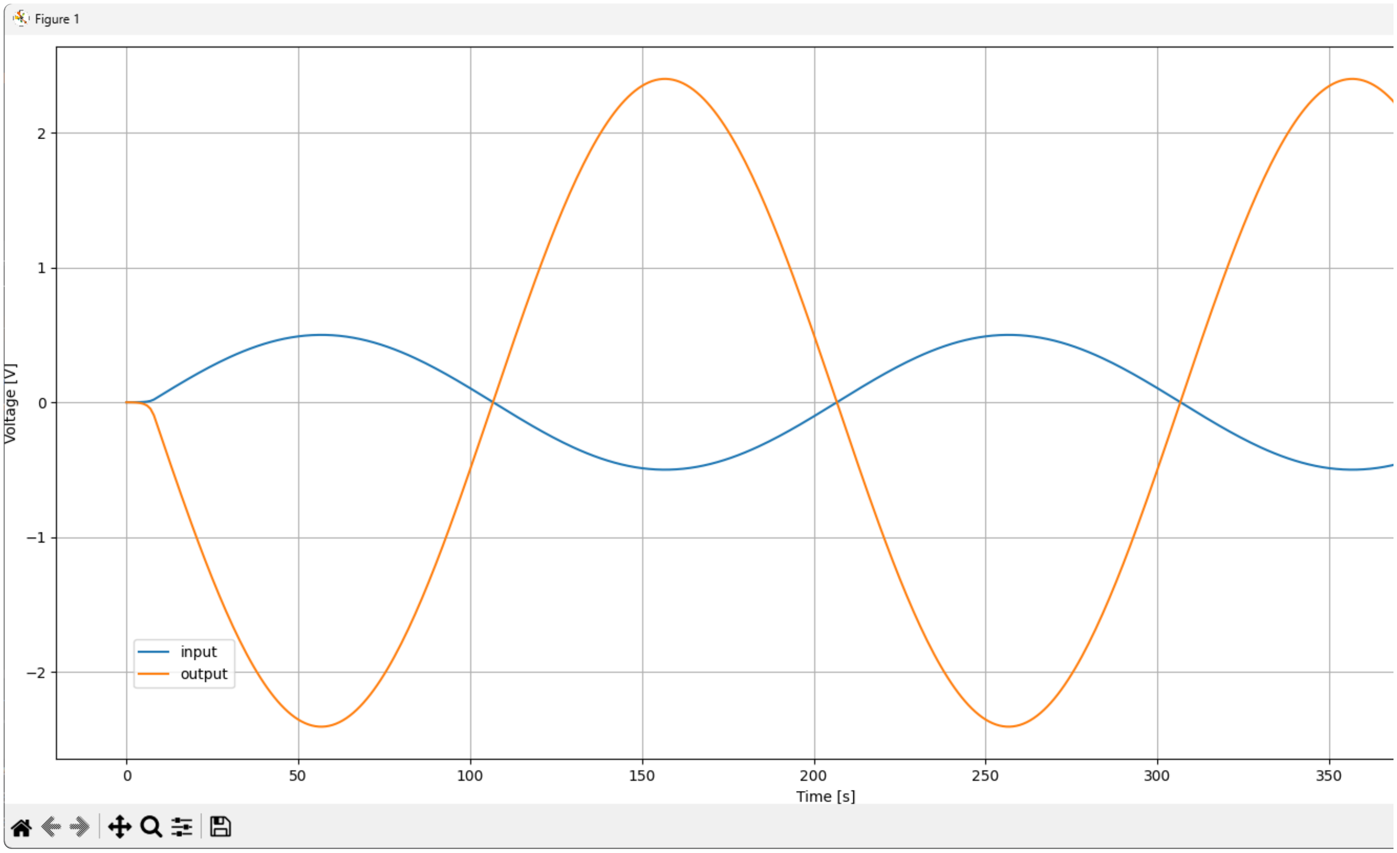
```
    print('Node {}: {} V'.format(str(node), float(node)))

print()

analysis = simulator.dc_sensitivity('v(out)')
for element in analysis.elements.values():
    print(element, float(element))
```

Console output

```
Node in: 10.0 V
Node out: 1.0 V

vinput 0.09999999999621426
r1_scale -0.8999991000992625
r1_bv_max -0.0
r1_m 0.8999999999287899
r1_w -0.0
r1_l -0.0
r1 -9.999990001102918e-05
r1:ef -0.0
r1:wf -0.0
r1:lf -0.0
r1:bv_max -0.0
r2_scale 0.899999099855317
r2_bv_max -0.0
r2_m -0.8999999999165925
r2_w -0.0
r2_l -0.0
r2 0.0008999991000504734
r2:ef -0.0
r2:wf -0.0
r2:lf -0.0
r2:bv_max -0.0
```

AC Coupled Transistor Amplifier





Sandbox/pyspice_transistor.py

```
import matplotlib.pyplot as plt
import PySpice.Logging.Logging as Logging
logger = Logging.setup_logging()
from PySpice.Doc.ExampleTools import find_libraries
from PySpice.Spice.Library import SpiceLibrary
from PySpice.Spice.Netlist import Circuit


libraries_path = find_libraries()
spice_library = SpiceLibrary(libraries_path)

voltage_unit = u_V = 1
resistance_unit = u_kΩ = 1000
frequency_unit = u_kHz = 1000
capacitance_unit = u_uF = 1e-6
u_MΩ = 1e6
u_pF = 1e-12

circuit = Circuit('Transistor')

circuit.V('power', 5, circuit.gnd, 15*u_V)
source = circuit.SinusoidalVoltageSource('in', 'in', circuit.gnd,
amplitude=.5*u_V, frequency=1*u_kHz)
circuit.C(1, 'in', 2, 10*u_uF)
circuit.R(1, 5, 2, 100*u_kΩ)
circuit.R(2, 2, 0, 20*u_kΩ)
circuit.R('C', 5, 4, 10*u_kΩ)
circuit.BJT(1, 4, 2, 3, model='bjt') # Q is mapped to BJT !
circuit.model('bjt', 'npn', bf=80, cjc=5*u_pF, rb=100)
circuit.R('E', 3, 0, 2*u_kΩ)
circuit.C(2, 4, 'out', 10*u_uF)
circuit.R('Load', 'out', 0, 1*u_MΩ)


figure, ax = plt.subplots(figsize=(20, 10))

# .ac dec 5 10m 1G

simulator = circuit.simulator(temperature=25, nominal_temperature=25)
analysis = simulator.transient(step_time=source.period/200,
end_time=source.period*2)

ax.set_title('')
ax.set_xlabel('Time [s]')
ax.set_ylabel('Voltage [V]')
ax.grid()
ax.plot(analysis['in'])
ax.plot(analysis.out)
ax.legend(('input', 'output'), loc=(.05,.1))

plt.tight_layout()
plt.show()
```

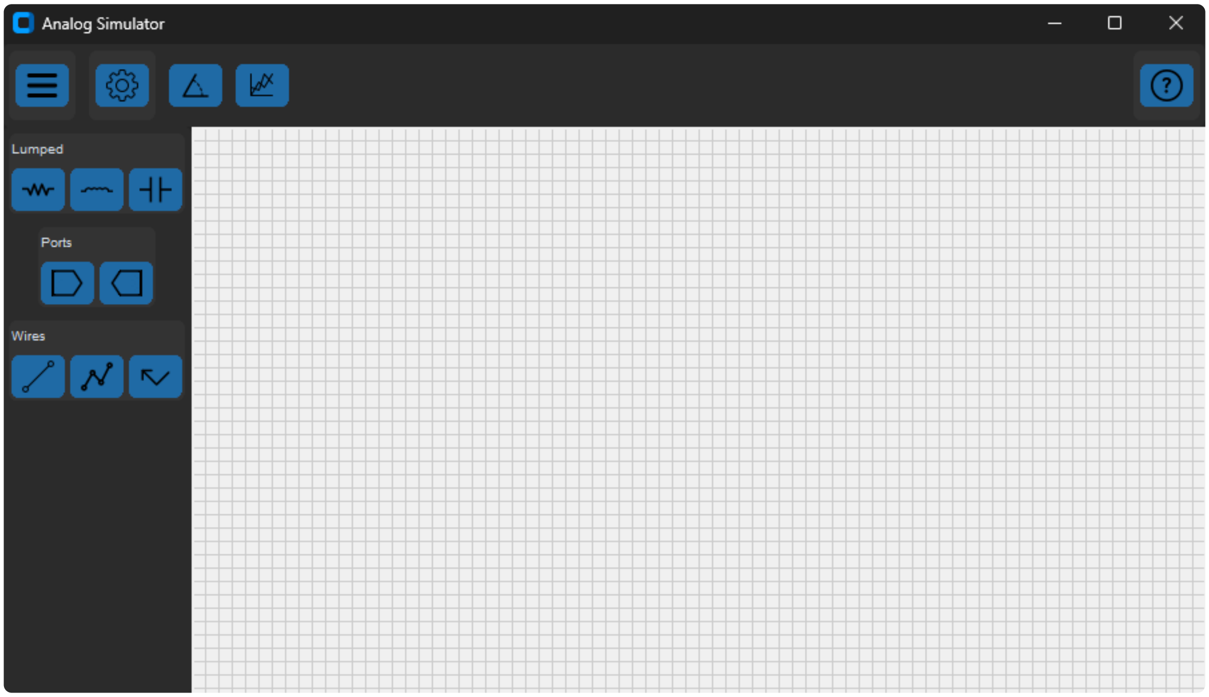
Analog Simulator User Interface

Directories copied from Microwave Simulator

- Circuits
    - Deleted circuit .json files
- Comp_Lib
- Helper_Lib
- icons
- images
- UI_Lib
- `Wire_Lib`

Modifications:

- Changed name of ideal_button_frame.py to ports_button_frame.py
- Changed name of `IdealButtonFrame` class to `PortsButtonFrame`
- Modified left_frame.py to create a `PortsButtonFrame` instead of `IdealButtonFrame`
- Modifications to top_frame.py
    - Removed `skrf` import
    - Replaced all code in `analyze_circuit()` method with pass - this is where we will put PySpice analysis code



Looks a lot like the Microwave Simulator user interface. Changes needed:

- Add voltage and current sources
- Add nodes for testing voltage
- Add a Ground Class
- Add PySpice analysis to Top Frame class

Refactor Classes

Universal Connector Method
Define common connector locations for 1-port and 2-port components

1-Port Components

- Ground

2-Port Components

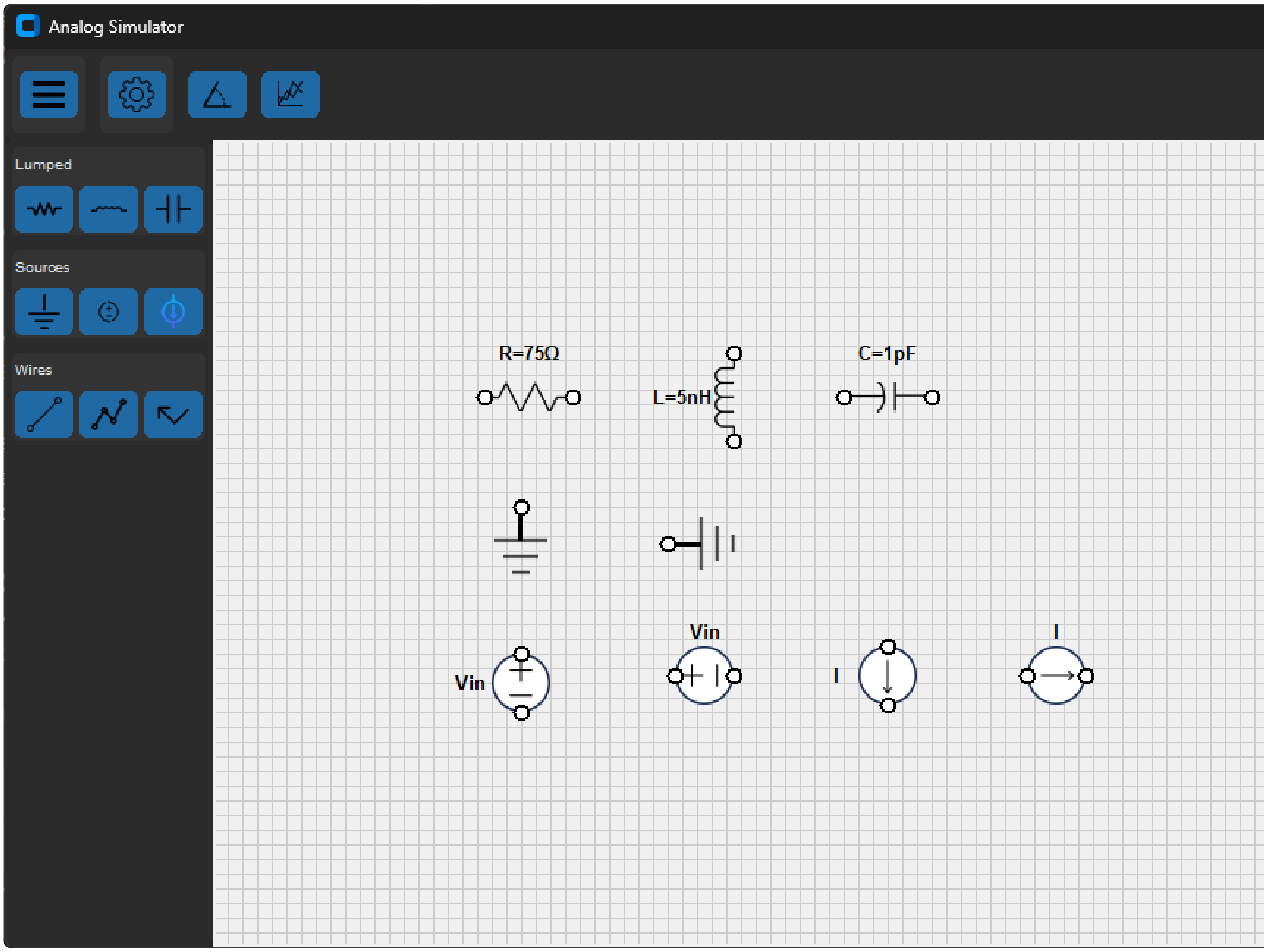
- Resistor
- Capacitor
- Inductor
- Voltage Source

- Current Source

Universal Component Class
Base class: Component
Universal derived class: `AnalogComponent`

Universal Wire Class
Base class: Wire
Universal derived class: `AnalogWire`

Universal Component Class with Universal Connector Method



Comp_Lib/component.py

```python
import tkinter as tk
from PIL import Image, ImageTk
from pathlib import Path

from Helper_Lib import Point


class Component:
    def __init__(self, canvas, comp_type, x1, y1, value):
        """Base class for component classes"""
        self.canvas = canvas
        self.comp_type = comp_type
        self.x1 = x1
        self.y1 = y1
        self.value = value

        self.id = None
        self.sel_id = None

        self.is_selected = False
        self.is_drawing = False
        self.selector = None
        self.angle = 0
        self.comp_text = None
        self.text = None
        self.text_id = None

        self.filename = None
        self.a_image = None
        self.ph_image = None
        self.bbox = None

        self.out = None
        self.in1 = None

        self.conn_list = []
        self.wire_list = []

        # Define component parameters in a dictionary of dictionaries
        self.params = {
            'filename': {
                'resistor': "../images/lumped/resistor_60x30.png",
                'capacitor': "../images/lumped/capacitor_60x30.png",
                'inductor': "../images/lumped/inductor_60x30.png",
                'ground': "../images/sources/ground_50x40.png",
                'vsource': "../images/sources/voltage_source_40x40.png",
                'isource': "../images/sources/current_source_40x40.png"
            },
            'text': {
                'resistor': 'R=' + str(self.value) + '\u2126',
                'capacitor': 'C=' + str(self.value) + 'pF',
                'inductor': 'L=' + str(self.value) + 'nH',
                'ground': "",
                'vsource': "Vin",
                'isource': "I"
            }
        }

    def set_image_filename(self):
        self.filename = Path(__file__).parent / self.params['filename']
[self.comp_type]

    def create_text(self):
        self.comp_text = self.params['text'][self.comp_type]
        if self.comp_type == 'isource' or self.comp_type == 'vsource':
            text_loc = Point(self.x1-35, self.y1)  # Put text on left side of
symbol
        else:
            text_loc = Point(self.x1, self.y1-30)  # Put text above symbol
        self.text_id = self.canvas.create_text(text_loc.x, text_loc.y,
                                text=self.comp_text, fill="black",
                                font='Helvetica 10 bold',
                                angle=self.angle, tags="text")

    def create_image(self, filename):
        """Initial component image creation"""
        self.a_image = Image.open(filename)
        self.a_image = self.a_image.rotate(self.angle, expand=True)
        self.ph_image = ImageTk.PhotoImage(self.a_image)
        self.id = self.canvas.create_image(self.x1, self.y1, anchor=tk.CENTER,
image=self.ph_image, tags='gate')

    def update_position(self):
        """Update the position when the gate object is moved"""
        self.canvas.coords(self.id, self.x1, self.y1)  # Update position

    def update_image(self, filename):
        """Update the image for gate symbol rotation"""
        self.a_image = Image.open(filename)
        self.a_image = self.a_image.rotate(self.angle, expand=True)  # Update
image rotation
        self.ph_image = ImageTk.PhotoImage(self.a_image)
        self.canvas.itemconfig(self.id, image=self.ph_image)  # Update image

    def update_bbox(self):
        """Update the bounding box to get current gate coordinates"""
        self.bbox = self.canvas.bbox(self.id)

    def update_text(self):
        if self.comp_type == 'isource' or self.comp_type == 'vsource':
            if self.angle == 0 or self.angle == 180:
                self.set_east_text()
            elif self.angle == 90 or self.angle == 270:
                self.set_north_text()
        else:
            if self.angle == 0 or self.angle == 180:
                self.set_north_text()
            elif self.angle == 90 or self.angle == 270:
                self.set_east_text()

    def set_east_text(self):
        self.canvas.coords(self.text_id, self.x1 - 35, self.y1)

    def set_north_text(self):
        self.canvas.coords(self.text_id, self.x1, self.y1 - 30)

    def create_selector(self):
        """Create the red rectangle selector and check to see if the gate is
selected"""
        x1, y1, x2, y2 = self.bbox[0] - 5, self.bbox[1] - 5, self.bbox[2] + 5,
self.bbox[3] + 5
        self.sel_id = self.canvas.create_rectangle(x1, y1, x2, y2, fill=None,
outline="red", width=2)
        self.set_selector_visibility()

    def update_selector(self):
        """Update the red rectangle selector coordinates and check to see if
the gate is selected"""
        x1, y1, x2, y2 = self.bbox[0] - 5, self.bbox[1] - 5, self.bbox[2] + 5,
self.bbox[3] + 5
        self.canvas.coords(self.sel_id, x1, y1, x2, y2)
        self.set_selector_visibility()

    def set_selector_visibility(self):
        """Set the selector visibility state"""
        if self.is_selected:
            self.canvas.itemconfig(self.sel_id, state='normal')
        else:
            self.canvas.itemconfig(self.sel_id, state='hidden')

    def set_connector_visibility(self):
        """Set the connector visibility state"""
        if self.is_drawing:
            for c in self.conn_list:
                self.canvas.itemconfig(c.id, state='normal')
        else:
            for c in self.conn_list:
                self.canvas.itemconfig(c.id, state='hidden')

    def check_connector_hit(self, x, y):
        """Hit test to see if a connector is at the provided x, y
coordinates"""
        for conn in self.conn_list:
            if conn.connector_hit_test(x, y):
                return conn
        return None

    def move_connected_wires(self):
        """Resize connected wires if the shape is moved"""
        for connection in self.wire_list:  # comp_conn, wire_name, wire_end
            for connector in self.conn_list:
                if connector.name == connection.comp_conn:
                    wire_obj = self.canvas.wire_dict[connection.wire_name]
                    if connection.wire_end == "begin":
                        wire_obj.x1 = connector.x
                        wire_obj.y1 = connector.y
                    elif connection.wire_end == "end":
                        wire_obj.x2 = connector.x
                        wire_obj.y2 = connector.y

    def rotate(self):
        """Set the rotation angle to the current angle + 90 deg, reset to 0 deg
if angle > 270 deg"""
        self.angle += 90
        if self.angle > 270:
            self.angle = 0

    def hit_test(self, x, y):
        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        if x1 <= x <= x2 and y1 <= y <= y2:
            self.is_selected = True
        else:
            self.is_selected = False
```

Comp_Lib/analog_component.py

```python
from Comp_Lib.component import Component
from Comp_Lib.connector import Connector
from Helper_Lib import Point


class AnalogComponent(Component):
    """Universal model for analog components"""
    def __init__(self, canvas, comp_type, x1, y1, value=0):
        super().__init__(canvas, comp_type, x1, y1, value)

        self.conn_params = {
            'ports': {
                'resistor': 2,
                'capacitor': 2,
                'inductor': 2,
                'ground': 1,
                'vsource': 2,
                'isource': 2
            },
            'conn_loc': {
                'resistor': 'ew',
                'capacitor': 'ew',
                'inductor': 'ew',
                'ground': 'n',
                'vsource': 'ns',
                'isource': 'ns'
            }
        }

        self.create()

    def create(self):
        self.set_image_filename()
        self.create_image(self.filename)
        self.update_bbox()
        self.create_text()
        self.create_selector()
        self.create_connectors()
        self.set_connector_visibility()

    def update(self):
        self.update_position()
        self.update_image(self.filename)
        self.update_bbox()
        self.update_text()
        self.update_selector()
        self.update_connectors()
        self.set_connector_visibility()

    def create_connectors(self):
        # Calculate position of connectors from current comp position and size
        center, e, w, n, s = self.get_geometry()

        num_ports = self.conn_params['ports'][self.comp_type]
        if num_ports == 1:
            self.out = Connector(self.canvas, "in1", center.x, center.y)
            self.conn_list = [self.out]
        elif num_ports == 2:
            self.out = Connector(self.canvas, "out", center.x, center.y)
            self.in1 = Connector(self.canvas, "in1", center.x, center.y)
            self.conn_list = [self.in1, self.out]

    def update_connectors(self):
        """Update the position of connectors here"""
        center, e, w, n, s = self.get_geometry()

        conn_loc = self.conn_params['conn_loc'][self.comp_type]
        if conn_loc == 'ew':  # 2-port with ew ports
            self.calc_ew_conn_rotation(n, s, e, w)
        elif conn_loc == 'ns':  # 2-port with ns ports
            self.calc_ns_conn_rotation(n, s, e, w)
        elif conn_loc == 'n':  # 1-port with n port
            self.calc_n_conn_rotation(n, w)

        for c in self.conn_list:
            c.update()

        self.move_connected_wires()

    def calc_ew_conn_rotation(self, n, s, e, w):
        if self.angle == 0 or self.angle == 180:
            self.out.x, self.out.y = w.x, w.y
            self.in1.x, self.in1.y = e.x, e.y
        elif self.angle == 90 or self.angle == 270:
            self.out.x, self.out.y = n.x, n.y
            self.in1.x, self.in1.y = s.x, s.y

    def calc_ns_conn_rotation(self, n, s, e, w):
        if self.angle == 0 or self.angle == 180:
            self.out.x, self.out.y = n.x, n.y
            self.in1.x, self.in1.y = s.x, s.y
        elif self.angle == 90 or self.angle == 270:
            self.out.x, self.out.y = w.x, w.y
            self.in1.x, self.in1.y = e.x, e.y

    def calc_n_conn_rotation(self, n, w):
        if self.angle == 0 or self.angle == 180:
            self.out.x, self.out.y = n.x, n.y
        elif self.angle == 90 or self.angle == 270:
            self.out.x, self.out.y = w.x, w.y

    def get_geometry(self):
        sign = lambda angle: 1 if angle == 0 or angle == 180 else -1

        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        w = x2 - x1
        h = y2 - y1
        center = Point(x1 + w / 2, y1 + h / 2)

        e = Point(center.x - sign(self.angle) * w / 2, center.y)
        w = Point(center.x + sign(self.angle) * w / 2, center.y)
        n = Point(center.x, center.y - sign(self.angle) * h / 2)
        s = Point(center.x, center.y + sign(self.angle) * h / 2)

        return center, e, w, n, s

    def __repr__(self):
        return ("Type: " + self.comp_type + " x1: " + str(self.x1) + " y1: " +
str(self.y1) + " value: " +
                str(self.value) + " wire list: " +
str(self.wire_list.__repr__()))

    def reprJson(self):
        return dict(type=self.comp_type, x1=self.x1, y1=self.y1,
angle=self.angle, value=self.value,
                    wire_list=self.wire_list)
```

UI_Lib/lump_button_frame.py

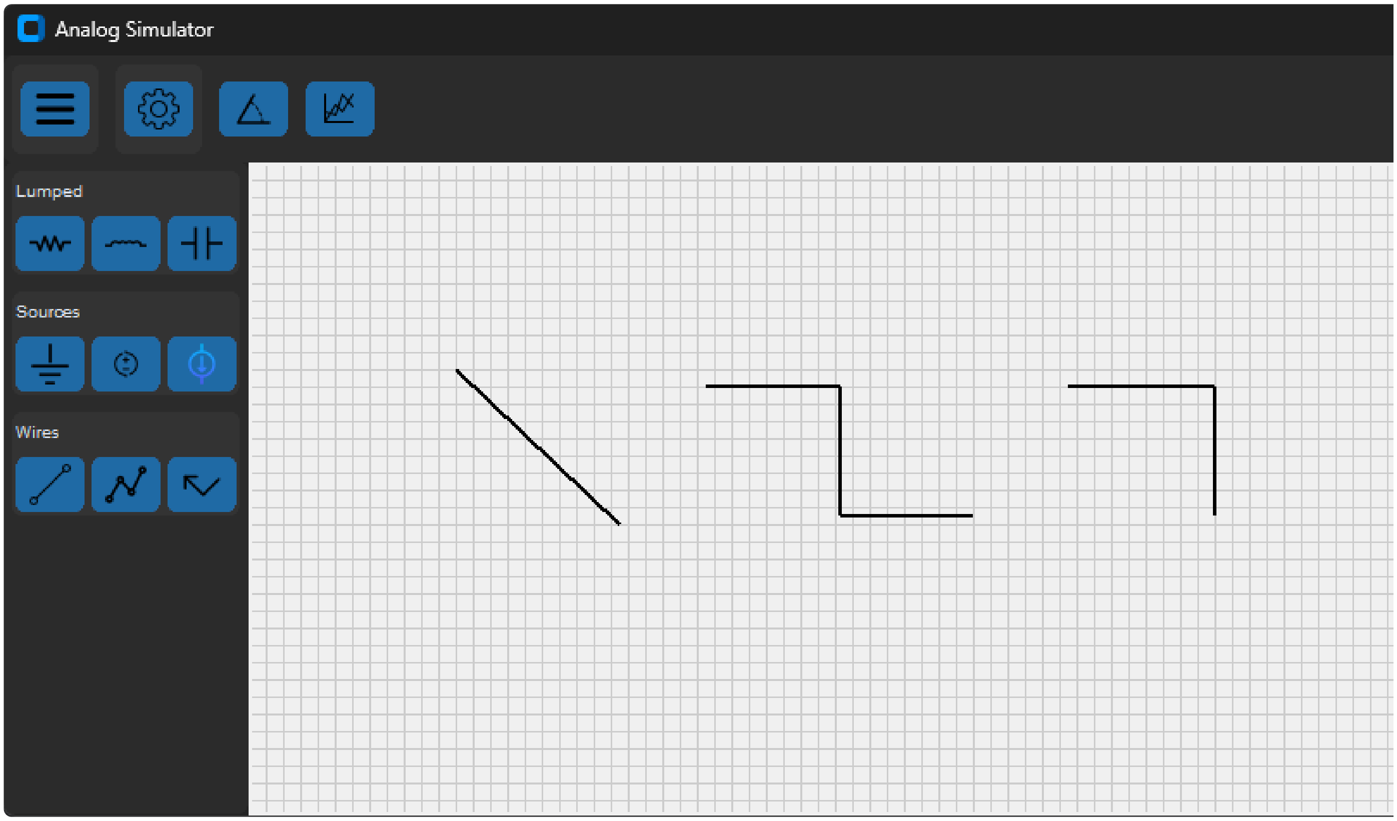
```
import customtkinter as ctk
from tktooltip import ToolTip
from pathlib import Path
from PIL import Image

from Comp_Lib import AnalogComponent


class LumpButtonFrame(ctk.CTkFrame):
    def __init__(self, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas
        self.button_id = None

        self.button_list = [("resistor", "../icons/resistor.png"),
                            ("inductor", "../icons/inductor.png"),
                            ("capacitor", "../icons/capacitor.png")]

        self.init_frame_widgets()

    def init_frame_widgets(self):
        frame_name_label = ctk.CTkLabel(self, text="Lumped", font=("Helvetica",
10), height=20)
        frame_name_label.grid(row=0, column=0, columnspan=3, sticky=ctk.W,
padx=2, pady=2)

        row_num, col_num = 1, 0
        for button in self.button_list:
            a_image = ctk.CTkImage(light_image=Image.open
                                    (Path(__file__).parent / button[1]),
                                    dark_image=Image.open
                                    (Path(__file__).parent / button[1]),
                                    size=(24, 24))
            self.button_id = ctk.CTkButton(self, text="", image=a_image,
width=30,
                                           command=lambda
a_name=button[0]:self.create_events(a_name))
            self.button_id.grid(row=row_num, column=col_num, sticky=ctk.W,
padx=2, pady=2)
            ToolTip(self.button_id, msg=button[0])
            row_num, col_num = self.update_grid_numbers(row_num, col_num)

    def create_events(self, name):
        comp = None
        if name == "resistor":
            comp = AnalogComponent(self.canvas, 'resistor', 100, 100, 75)
        elif name == "inductor":
            comp = AnalogComponent(self.canvas, 'inductor', 100, 100, 5)
        elif name == "capacitor":
            comp = AnalogComponent(self.canvas, 'capacitor',100, 100, 1)
        self.canvas.comp_list.append(comp)
        self.canvas.redraw()
        self.canvas.mouse.move_mouse_bind_events()

    @staticmethod
    def update_grid_numbers(row, column):
        column += 1
        if column > 2:
            column = 0
            row += 1
        return row, column
```

UI_Lib/sources_button_frame.py

```
import customtkinter as ctk
from tktooltip import ToolTip
from pathlib import Path
from PIL import Image

from Comp_Lib import AnalogComponent


class SourcesButtonFrame(ctk.CTkFrame):
    def __init__(self, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas
        self.button_id = None

        self.button_list = [("ground", "../icons/ground.png"),
                            ("vsource", "../icons/voltage_source.png"),
                            ("isource", "../icons/current_source.png")]

        self.init_frame_widgets()

    def init_frame_widgets(self):
        frame_name_label = ctk.CTkLabel(self, text="Sources", font=
("Helvetica", 10), height=20)
        frame_name_label.grid(row=0, column=0, columnspan=3, sticky=ctk.W,
padx=2, pady=2)
```

```
        row_num, col_num = 1, 0
        for button in self.button_list:
            a_image = ctk.CTkImage(light_image=Image.open
                                    (Path(__file__).parent / button[1]),
                                    dark_image=Image.open
                                    (Path(__file__).parent / button[1]),
                                    size=(24, 24))
            self.button_id = ctk.CTkButton(self, text="", image=a_image,
width=30,
                                           command=lambda
a_name=button[0]:self.create_events(a_name))
            self.button_id.grid(row=row_num, column=col_num, sticky=ctk.W,
padx=2, pady=2)
            ToolTip(self.button_id, msg=button[0])
            row_num, col_num = self.update_grid_numbers(row_num, col_num)

    def create_events(self, name):
        comp = None
        if name == "ground":
            comp = AnalogComponent(self.canvas, 'ground', 100, 100)
        elif name == "vsource":
            comp = AnalogComponent(self.canvas, 'vsource',100, 100)
        elif name == "isource":
            comp = AnalogComponent(self.canvas, 'isource', 100, 100)
        self.canvas.comp_list.append(comp)
        self.canvas.redraw()
        self.canvas.mouse.move_mouse_bind_events()

    @staticmethod
    def update_grid_numbers(row, column):
        column += 1
        if column > 2:
            column = 0
            row += 1
        return row, column
```

Universal Wire Class



Wire_lib/wire.py

```python
from Wire_Lib.wire_selector import WireSelector


class Wire:
    """Base class for wire classes"""
    def __init__(self, canvas, wire_type, x1, y1, x2, y2):
        self.canvas = canvas
        self.wire_type = wire_type
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2

        self.fill_color = "black"
        self.border_width = 2
        self.line_direction = "horizontal"

        self.name = None
        self.id = None
        self.is_selected = False
        self.selector = None
        self.width = 2
        self.bbox = None

        # Connections for wire list
        self.in_cnx = None
        self.out_cnx = None
        self.cnx = []

        self.sel1, self.sel2 = None, None

    def update_bbox(self):
        self.bbox = self.canvas.bbox(self.id)

    def create_selectors(self):
        self.sel1 = WireSelector(self.canvas, "begin", self.x1, self.y1)
        self.sel2 = WireSelector(self.canvas, "end", self.x2, self.y2)

    def update_selectors(self):
        self.sel1.x, self.sel1.y = self.x1, self.y1
        self.sel2.x, self.sel2.y = self.x2, self.y2
        self.sel1.update()
        self.sel2.update()

    def update_selection(self):
        if self.is_selected:
            self.canvas.itemconfigure(self.id, fill="red")
            self.canvas.itemconfigure(self.sel1.id, state='normal')
            self.canvas.itemconfigure(self.sel2.id, state='normal')
        else:
            self.canvas.itemconfigure(self.id, fill="black")
            self.canvas.itemconfigure(self.sel1.id, state='hidden')
            self.canvas.itemconfigure(self.sel2.id, state='hidden')

    def hit_test(self, x, y):
        x1, y1 = self.bbox[0], self.bbox[1]
        x2, y2 = self.bbox[2], self.bbox[3]
        if x1 <= x <= x2 and y1 <= y <= y2:
            self.is_selected = True
        else:
            self.is_selected = False

    def sel_hit_test(self, x, y):
        if self.sel1.selector_hit_test(x, y):
            self.selector = self.sel1.name
            return self.sel1
        elif self.sel2.selector_hit_test(x, y):
            self.selector = self.sel2.name
            return self.sel2
        else:
            return None

    def resize(self, offsets, event):
        offset_x1, offset_y1, offset_x2, offset_y2 = offsets
        if self.selector == "end":
            x2 = event.x - offset_x2
            y2 = event.y - offset_y2
            self.x2, self.y2 = x2, y2
            self.x2, self.y2 = self.canvas.grid.snap_to_grid(self.x2, self.y2)
        elif self.selector == "begin":
            x1 = event.x - offset_x1
            y1 = event.y - offset_y1
            self.x1, self.y1 = x1, y1
            self.x1, self.y1 = self.canvas.grid.snap_to_grid(self.x1, self.y1)

    def create_wire_list_cnx(self, comp_type, wire_end):
        if wire_end == 'out':
            if comp_type == "inport":
                self.out_cnx = (comp_type, 0)
            else:
                self.out_cnx = (comp_type, 1)
        elif wire_end == 'in1':
            self.in_cnx = (comp_type, 0)
        if self.in_cnx and self.out_cnx:
            self.cnx = [self.out_cnx, self.in_cnx]
            self.canvas.conn_list.append(self.cnx)
```

Wire_Lib/analog_wire.py

```
from Wire_Lib.wire import Wire


class AnalogWire(Wire):
    def __init__(self, canvas, wire_type, x1, y1, x2, y2):
        super().__init__(canvas, wire_type, x1, y1, x2, y2)

        self.seg1, self.seg2, self.seg3 = None, None, None
        self.segment_list = []

        self.create_wire()
        self.update_bbox()
        self.create_selectors()
        self.update_selection()

    def create_wire(self):
        if self.wire_type == 'straight':
            self.create_straight_wire()
        elif self.wire_type == 'elbow':
            self.create_elbow_wire()
        elif self.wire_type == 'segment':
            self.create_segment_wire()

    def create_straight_wire(self):
        self.id = self.canvas.create_line(self.x1, self.y1, self.x2, self.y2,
width=self.width)

    def create_elbow_wire(self):
        w = self.x2 - self.x1
        h = self.y2 - self.y1

        if abs(w) >= abs(h):  # Horizontal
            self.id = self.canvas.create_line(self.x1, self.y1, self.x2,
self.y1,
                                              self.x2, self.y1, self.x2,
self.y2,
                                              fill=self.fill_color,
                                              width=self.border_width,
tags="wire")
        else:  # Vertical
            self.id = self.canvas.create_line(self.x1, self.y1, self.x1,
self.y2,
                                              self.x1, self.y2, self.x2,
self.y2,
                                              fill=self.fill_color,
                                              width=self.border_width,
tags="wire")

    def create_segment_wire(self):
        w = self.x2 - self.x1
        h = self.y2 - self.y1

        if abs(w) >= abs(h):  # Horizontal
            self.seg1 = self.x1, self.y1, self.x1 + w / 2, self.y1
            self.seg2 = self.x1 + w / 2, self.y1, self.x1 + w / 2, self.y2
            self.seg3 = self.x1 + w / 2, self.y2, self.x2, self.y2
        else:  # Vertical
            self.seg1 = self.x1, self.y1, self.x1, self.y1 + h / 2
            self.seg2 = self.x1, self.y1 + h / 2, self.x2, self.y1 + h / 2
            self.seg3 = self.x2, self.y1 + h / 2, self.x2, self.y2
        self.segment_list = [self.seg1, self.seg2, self.seg3]
        self.draw_segments()

    def draw_segments(self):
        for s in self.segment_list:
            self.id = self.canvas.create_line(s, fill=self.fill_color,
width=self.border_width, tags='wire')

    def update(self):
        self.update_position()
        self.update_bbox()
        self.update_selectors()
        self.update_selection()

    def update_position(self):
        if self.wire_type == 'straight':
            self.update_straight_position()
        elif self.wire_type == 'elbow':
            self.update_elbow_position()
        elif self.wire_type == 'segment':
            self.update_segment_position()

    def update_straight_position(self):
        """Update the position when the attached component is moved"""
        self.canvas.coords(self.id, self.x1, self.y1, self.x2, self.y2)

    def update_elbow_position(self):
        """Update the position when the attached component is moved"""
        w = self.x2 - self.x1
        h = self.y2 - self.y1
        if abs(w) >= abs(h):
            self.canvas.coords(self.id, self.x1, self.y1, self.x2, self.y1,
                                        self.x2, self.y1, self.x2, self.y2)
        else:
            self.canvas.coords(self.id, self.x1, self.y1, self.x1, self.y2,
                                        self.x1, self.y2, self.x2, self.y2)

    def update_segment_position(self):
        """Update the position when the attached component is moved"""
        w = self.x2 - self.x1
        h = self.y2 - self.y1
        if abs(w) >= abs(h):
            self.canvas.coords(self.id, self.x1, self.y1, self.x1 + w / 2,
self.y1,
                               self.x1 + w / 2, self.y1, self.x1 + w / 2,
self.y2,
                               self.x1 + w / 2, self.y2, self.x2, self.y2)
        else:
            self.canvas.coords(self.id, self.x1, self.y1, self.x1, self.y1 + h
/ 2,
                               self.x1, self.y1 + h / 2, self.x2, self.y1 + h /
2,
                               self.x2, self.y1 + h / 2, self.x2, self.y2)

    def hit_test(self, x, y):
        # 2-Point Line equation: y = m * (x - x1) + y1
        x1, y1 = self.x1, self.y1
        x2, y2 = self.x2, self.y2

        # Calculate the slope: m = (y2 - y1) / (x2 - x1)
        if (x2 - x1) == 0:
            m = 0
        else:
            m = (y2 - y1)/(x2 - x1)

        # Check to see if the point (x, y) is on the line and between the two
end points
        tol = 10
        if y - tol <= m*(x - x1) + y1 <= y + tol:
            if (min(x1, x2) <= x <= max(x1, x2)) and (min(y1, y2) <= y <=
max(y1, y2)):
                self.is_selected = True
        else:
            self.is_selected = False

    def reprJson(self):
        return dict(type=self.wire_type, x1=self.x1, y1=self.y1, x2=self.x2,
y2=self.y2, name=self.name)
```

UI_Lib/wire_button_frame.py

```
import customtkinter as ctk
from pathlib import Path
from PIL import Image

from Wire_Lib import AnalogWire
```

```
class WireButtonFrame(ctk.CTkFrame):
    def __init__(self, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas
        self.wire = None
        self.wire_count = 0

        # Add frame widgets here
        frame_name_label = ctk.CTkLabel(self, text="Wires", font=("Helvetica",
10), height=20)
        frame_name_label.grid(row=0, column=0, columnspan=3, sticky=ctk.W,
padx=2, pady=2)

        straight_wire_image = ctk.CTkImage(light_image=Image.open
                                (Path(__file__).parent /
"../icons/straight_line.png"),
                                dark_image=Image.open
                                (Path(__file__).parent /
"../icons/straight_line.png"),
                                size=(24, 24))

        straight_wire_button = ctk.CTkButton(self, text="",
image=straight_wire_image, width=30,
                                        command=self.create_straight_wire)
        straight_wire_button.grid(row=1, column=0, sticky=ctk.W, padx=2,
pady=2)

        segment_wire_image = ctk.CTkImage(light_image=Image.open
                                (Path(__file__).parent /
"../icons/segment_line.png"),
                                dark_image=Image.open
                                (Path(__file__).parent /
"../icons/segment_line.png"),
                                size=(24, 24))

        segment_wire_button = ctk.CTkButton(self, text="",
image=segment_wire_image, width=30,
                                        command=self.create_segment_wire)
        segment_wire_button.grid(row=1, column=1, sticky=ctk.W, padx=2, pady=2)

        elbow_wire_image = ctk.CTkImage(light_image=Image.open
                                (Path(__file__).parent /
"../icons/elbow_line.png"),
                                dark_image=Image.open
                                (Path(__file__).parent /
"../icons/elbow_line.png"),
                                size=(24, 24))

        elbow_wire_button = ctk.CTkButton(self, text="",
image=elbow_wire_image, width=30,
                                        command=self.create_elbow_wire)
        elbow_wire_button.grid(row=1, column=2, sticky=ctk.W, padx=2, pady=2)

    # Shape button handlers
    def create_straight_wire(self):
        wire = AnalogWire(self.canvas, 'straight', 0, 0, 0, 0)
        self.create_wire(wire)

    def create_segment_wire(self):
        wire = AnalogWire(self.canvas, 'segment', 0, 0, 0, 0)
        self.create_wire(wire)

    def create_elbow_wire(self):
        wire = AnalogWire(self.canvas, 'elbow', 0, 0, 0, 0)
        self.create_wire(wire)

    def create_wire(self, wire):
        self.assign_wire_name(wire)
        self.canvas.mouse.current_wire_obj = wire
        self.canvas.show_connectors()
        self.canvas.comp_list.append(wire)
        self.canvas.mouse.draw_wire_mouse_events()

    def assign_wire_name(self, wire):
        self.wire_count += 1
        wire_name = 'wire' + str(self.wire_count)
        wire.name = wire_name
        self.canvas.wire_dict[wire_name] = wire
```

UI_Lib/mouse.py

```
from Helper_Lib import Point
from Wire_Lib import AnalogWire
from Comp_Lib import Connection


class Mouse:
```

```python
    def __init__(self, canvas):
        self.canvas = canvas

        self.selected_comp = None
        self.current_wire_obj = None
        self.start = Point(0, 0)
        self.offset1 = Point(0, 0)
        self.offset2 = Point(0, 0)

        self.move_mouse_bind_events()

    def move_mouse_bind_events(self):
        self.canvas.bind("<Button-1>", self.move_left_down)
        self.canvas.bind("<B1-Motion>", self.move_left_drag)
        self.canvas.bind("<ButtonRelease-1>", self.move_left_up)

    def draw_wire_mouse_events(self):  # Added method to bind draw wire methods
        self.canvas.bind("<Button-1>", self.draw_left_down)
        self.canvas.bind("<B1-Motion>", self.draw_left_drag)
        self.canvas.bind("<ButtonRelease-1>", self.draw_left_up)

    def resize_wire_mouse_events(self):
        self.canvas.bind("<Button-1>", self.resize_left_down)
        self.canvas.bind("<B1-Motion>", self.resize_left_drag)
        self.canvas.bind("<ButtonRelease-1>", self.resize_left_up)

    def move_left_down(self, event):
        x, y = event.x, event.y
        self.comp_hit_test(x, y)
        if self.selected_comp:
            if isinstance(self.selected_comp, AnalogWire):
                x1, y1 = self.selected_comp.x1, self.selected_comp.y1
                x2, y2 = self.selected_comp.x2, self.selected_comp.y2
                self.offset1.set(x - x1, y - y1)
                self.offset2.set(x - x2, y - y2)
            else:
                x1, y1 = self.selected_comp.x1, self.selected_comp.y1
                self.offset1.x, self.offset1.y =
self.canvas.grid.snap_to_grid(self.offset1.x, self.offset1.y)
                self.offset1.set(x - x1, y - y1)

    def move_left_drag(self, event):
        if self.selected_comp:
            if isinstance(self.selected_comp, AnalogWire):
                x1 = event.x - self.offset1.x
                y1 = event.y - self.offset1.y
                x1, y1 = self.canvas.grid.snap_to_grid(x1, y1)
                x2 = event.x - self.offset2.x
                y2 = event.y - self.offset2.y
                x2, y2 = self.canvas.grid.snap_to_grid(x2, y2)
                self.selected_comp.x1, self.selected_comp.y1 = x1, y1
                self.selected_comp.x2, self.selected_comp.y2 = x2, y2
                self.canvas.redraw()
            else:
                x = event.x - self.offset1.x
                y = event.y - self.offset1.y
                x, y = self.canvas.grid.snap_to_grid(x, y)
                self.selected_comp.x1, self.selected_comp.y1 = x, y
                self.canvas.redraw()

    def move_left_up(self, _event):
        self.offset1.set(0, 0)
        self.offset2.set(0, 0)
        self.canvas.redraw()

    def draw_left_down(self, event):  # Added method for draw left down
        if self.current_wire_obj:
            # self.unselect_all()
            self.start.x = event.x
            self.start.y = event.y
            self.start.x, self.start.y =
self.canvas.grid.snap_to_grid(self.start.x, self.start.y)

            self.current_wire_obj.x1, self.current_wire_obj.y1 = self.start.x,
self.start.y
            self.current_wire_obj.x2, self.current_wire_obj.y2 = self.start.x,
self.start.y

            self.select_connector(self.current_wire_obj, "begin", self.start.x,
self.start.y)

    def draw_left_drag(self, event):  # Added method for draw left drag
        if self.current_wire_obj:
            wire = self.current_wire_obj
            x, y = event.x, event.y
            x, y = self.canvas.grid.snap_to_grid(x, y)
            wire.x1, wire.y1 = self.start.x, self.start.y
            wire.x2, wire.y2 = x, y
            self.current_wire_obj.update()

    def draw_left_up(self, event):  # Added method for draw left up
        self.select_connector(self.current_wire_obj, "end", event.x, event.y)
        self.canvas.hide_connectors()
        self.current_wire_obj = None
        self.move_mouse_bind_events()

    def resize_left_down(self, event):
        if self.selected_comp:
            x1, y1 = self.selected_comp.x1, self.selected_comp.y1
            x1, y1 = self.canvas.grid.snap_to_grid(x1, y1, "resize")
            x2, y2 = self.selected_comp.x2, self.selected_comp.y2
            x2, y2 = self.canvas.grid.snap_to_grid(x2, y2, "resize")
            self.offset1.x = event.x - x1
            self.offset1.y = event.y - y1
            self.offset2.x = event.x - x2
            self.offset2.y = event.y - y2
            self.selected_comp.update()

    def resize_left_drag(self, event):
        if self.selected_comp:
            offsets = [self.offset1.x, self.offset1.y, self.offset2.x,
self.offset2.y]
            self.selected_comp.resize(offsets, event)
            self.selected_comp.update()

    def resize_left_up(self, _event):
        self.offset1.x, self.offset1.y = 0, 0
        self.offset2.x, self.offset2.y = 0, 0
        self.move_mouse_bind_events()

    def comp_hit_test(self, x, y):
        for comp in self.canvas.comp_list:
            comp.hit_test(x, y)
            if comp.is_selected:
                if isinstance(self.selected_comp, AnalogWire):
                    result = comp.sel_hit_test(x, y)
                    if result is not None:
                        self.resize_wire_mouse_events()
                comp.update()
                self.selected_comp = comp
                return

        # No shape hit - unselect all
        self.selected_comp = None
        self.unselect_all()

    def unselect_all(self):
        for comp in self.canvas.comp_list:
            comp.is_selected = False
            comp.update()

    def select_connector(self, wire_obj, wire_end, x, y):
        for comp in self.canvas.comp_list:
            if not isinstance(comp, AnalogWire):
                conn = comp.check_connector_hit(x, y)
                if conn:
                    if wire_end == "begin":
                        wire_obj.x1, wire_obj.y1 = conn.x, conn.y
                    elif wire_end == "end":
                        wire_obj.x2, wire_obj.y2 = conn.x, conn.y
                    a_conn = Connection(conn.name, self.current_wire_obj.name,
wire_end)
                    wire_obj.create_wire_list_cnx(comp.comp_type, conn.name)
                    comp.wire_list.append(a_conn)
                    self.canvas.redraw()
```

Transistor Amplifier Components

Lets implement the AC Coupled Transistor Amplifier from PySpice Evaluation

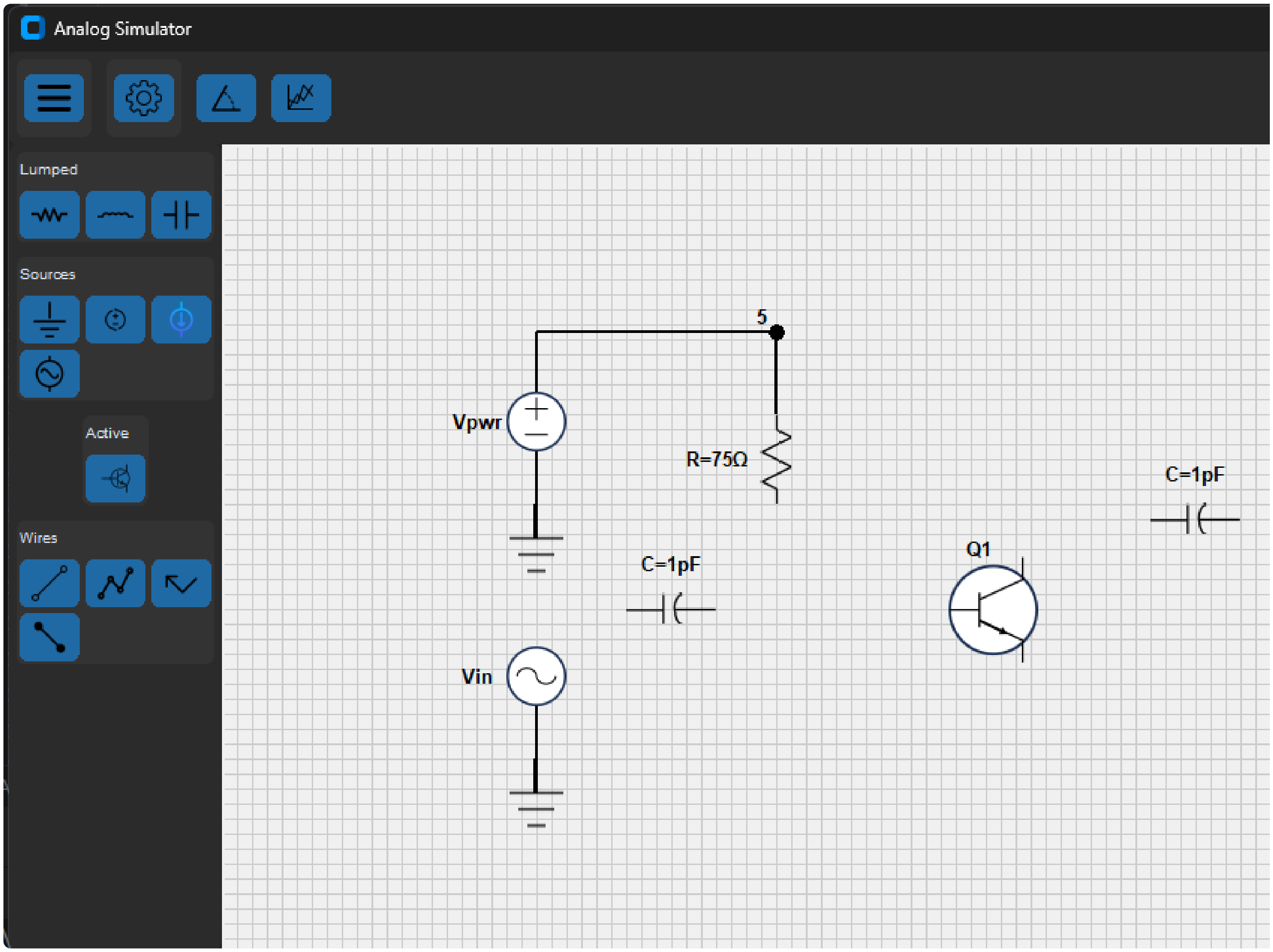
Components needed:

- DC Power Source
- AC Power Source
- Capacitor
- Resistor
- NPN Transistor
- Ground
- Nodes with Text - in, out, 0, 2, 3, 4, 5
- Matplotlib oscilloscope

PySpice Netlist

`circuit.R(1, 5, 2, 100@u_kΩ)`

- 1 = Resistor label as in R1
- 5, 2 = input and output nodes
- 100@u_kΩ = resistance with units



Comp_Lib/component.py

```
import tkinter as tk
from PIL import Image, ImageTk
from pathlib import Path

from Helper_Lib import Point


class Component:
    def __init__(self, canvas, comp_type, x1, y1, value):
        """Base class for component classes"""
        self.canvas = canvas
        self.comp_type = comp_type
        self.x1 = x1
        self.y1 = y1
        self.value = value

        self.id = None
        self.sel_id = None

        self.is_selected = False
        self.is_drawing = False
        self.selector = None
        self.angle = 0
        self.comp_text = None
        self.text = None
        self.text_id = None

        self.filename = None
        self.a_image = None
        self.ph_image = None
        self.bbox = None

        self.out = None
        self.in1 = None
        self.ba = None
        self.em = None
        self.co = None

        self.conn_list = []
        self.wire_list = []

        # Define component parameters in a dictionary of dictionaries
        self.params = {
            'filename': {
                'resistor': "../images/lumped/resistor_60x30.png",
                'capacitor': "../images/lumped/capacitor_60x30.png",
                'inductor': "../images/lumped/inductor_60x30.png",
                'ground': "../images/sources/ground_50x40.png",
                'vsource': "../images/sources/voltage_source_40x40.png",
                'isource': "../images/sources/current_source_40x40.png",
                'ac_source': "../images/sources/ac_voltage_source_40x40.png",
                'npn_transistor':
"../images/transistors/npn_transistor_60x71.png"
            },
            'text': {
                'resistor': 'R=' + str(self.value) + '\u2126',
                'capacitor': 'C=' + str(self.value) + 'pF',
                'inductor': 'L=' + str(self.value) + 'nH',
                'ground': "",
                'vsource': "Vpwr",
                'isource': "I",
                'ac_source': "Vin",
                'npn_transistor': "Q1"
            }
        }

    def set_image_filename(self):
        self.filename = Path(__file__).parent / self.params['filename']
[self.comp_type]

    def create_text(self):
        self.comp_text = self.params['text'][self.comp_type]
        if self.comp_type == 'isource' or self.comp_type == 'vsource' or
self.comp_type == 'ac_source':
            text_loc = Point(self.x1-40, self.y1)  # Put text on left side of
symbol
        elif self.comp_type == 'npn_transistor':
            text_loc = Point(self.x1 - 10, self.y1 - 40)  # Put text above
symbol
        else:
            text_loc = Point(self.x1, self.y1-30)  # Put text above symbol
        self.text_id = self.canvas.create_text(text_loc.x, text_loc.y,
                                text=self.comp_text, fill="black",
                                font='Helvetica 10 bold',
                                angle=self.angle, tags="text")

    def create_image(self, filename):
        """Initial component image creation"""
        self.a_image = Image.open(filename)
        self.a_image = self.a_image.rotate(self.angle, expand=True)
        self.ph_image = ImageTk.PhotoImage(self.a_image)
        self.id = self.canvas.create_image(self.x1, self.y1, anchor=tk.CENTER,
image=self.ph_image, tags='gate')

    def update_position(self):
        """Update the position when the gate object is moved"""
        self.canvas.coords(self.id, self.x1, self.y1)  # Update position

    def update_image(self, filename):
        """Update the image for gate symbol rotation"""
        self.a_image = Image.open(filename)
        self.a_image = self.a_image.rotate(self.angle, expand=True)  # Update
image rotation
        self.ph_image = ImageTk.PhotoImage(self.a_image)
        self.canvas.itemconfig(self.id, image=self.ph_image)  # Update image

    def update_bbox(self):
        """Update the bounding box to get current gate coordinates"""
        self.bbox = self.canvas.bbox(self.id)
```

```python
    def update_text(self):
        if self.comp_type == 'isource' or self.comp_type == 'vsource' or self.comp_type == 'ac_source':
            if self.angle == 0 or self.angle == 180:
                self.set_east_text()
            elif self.angle == 90 or self.angle == 270:
                self.set_north_text()
        elif self.comp_type == 'npn_transistor':
            self.set_transistor_text()
        else:
            if self.angle == 0 or self.angle == 180:
                self.set_north_text()
            elif self.angle == 90 or self.angle == 270:
                self.set_east_text()

    def set_east_text(self):
        self.canvas.coords(self.text_id, self.x1 - 40, self.y1)

    def set_north_text(self):
        self.canvas.coords(self.text_id, self.x1, self.y1 - 30)

    def set_transistor_text(self):
        self.canvas.coords(self.text_id, self.x1 - 10, self.y1 - 40)

    def create_selector(self):
        """Create the red rectangle selector and check to see if the gate is selected"""
        x1, y1, x2, y2 = self.bbox[0] - 5, self.bbox[1] - 5, self.bbox[2] + 5, self.bbox[3] + 5
        self.sel_id = self.canvas.create_rectangle(x1, y1, x2, y2, fill=None, outline="red", width=2)
        self.set_selector_visibility()

    def update_selector(self):
        """Update the red rectangle selector coordinates and check to see if the gate is selected"""
        x1, y1, x2, y2 = self.bbox[0] - 5, self.bbox[1] - 5, self.bbox[2] + 5, self.bbox[3] + 5
        self.canvas.coords(self.sel_id, x1, y1, x2, y2)
        self.set_selector_visibility()

    def set_selector_visibility(self):
        """Set the selector visibility state"""
        if self.is_selected:
            self.canvas.itemconfig(self.sel_id, state='normal')
        else:
            self.canvas.itemconfig(self.sel_id, state='hidden')

    def set_connector_visibility(self):
        """Set the connector visibility state"""
        if self.is_drawing:
            for c in self.conn_list:
                self.canvas.itemconfig(c.id, state='normal')
        else:
            for c in self.conn_list:
                self.canvas.itemconfig(c.id, state='hidden')

    def check_connector_hit(self, x, y):
        """Hit test to see if a connector is at the provided x, y
coordinates"""
        for conn in self.conn_list:
            if conn.connector_hit_test(x, y):
                return conn
        return None

    def move_connected_wires(self):
        """Resize connected wires if the shape is moved"""
        for connection in self.wire_list:  # comp_conn, wire_name, wire_end
            for connector in self.conn_list:
                if connector.name == connection.comp_conn:
                    wire_obj = self.canvas.wire_dict[connection.wire_name]
                    if connection.wire_end == "begin":
                        wire_obj.x1 = connector.x
                        wire_obj.y1 = connector.y
                    elif connection.wire_end == "end":
                        wire_obj.x2 = connector.x
                        wire_obj.y2 = connector.y

    def rotate(self):
        """Set the rotation angle to the current angle + 90 deg, reset to 0 deg
if angle > 270 deg"""
        self.angle += 90
        if self.angle > 270:
            self.angle = 0

    def hit_test(self, x, y):
        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        if x1 <= x <= x2 and y1 <= y <= y2:
            self.is_selected = True
        else:
            self.is_selected = False
```

Comp_Lib/analog_component.py

```
from Comp_Lib.component import Component
from Comp_Lib.connector import Connector
from Helper_Lib import Point


class AnalogComponent(Component):
    """Universal model for analog components"""
    def __init__(self, canvas, comp_type, x1, y1, value=0):
        super().__init__(canvas, comp_type, x1, y1, value)

        self.conn_params = {
            'ports': {
                'resistor': 2,
                'capacitor': 2,
                'inductor': 2,
                'ground': 1,
                'vsource': 2,
                'isource': 2,
                'ac_source': 2,
                'npn_transistor': 3
            },
            'conn_loc': {
                'resistor': 'ew',
                'capacitor': 'ew',
                'inductor': 'ew',
                'ground': 'n',
                'vsource': 'ns',
                'isource': 'ns',
                'ac_source': 'ns',
                'npn_transistor': 'nsw'
            }
        }

        self.create()

    def create(self):
        self.set_image_filename()
        self.create_image(self.filename)
        self.update_bbox()
        self.create_text()
        self.create_selector()
        self.create_connectors()
        self.set_connector_visibility()

    def update(self):
        self.update_position()
        self.update_image(self.filename)
        self.update_bbox()
        self.update_text()
        self.update_selector()
        self.update_connectors()
        self.set_connector_visibility()

    def create_connectors(self):
        # Calculate position of connectors from current comp position and size
        center, e, w, n, s = self.get_geometry()

        num_ports = self.conn_params['ports'][self.comp_type]
        if num_ports == 1:
            self.out = Connector(self.canvas, "in1", center.x, center.y)
            self.conn_list = [self.out]
        elif num_ports == 2:
            self.out = Connector(self.canvas, "out", center.x, center.y)
            self.in1 = Connector(self.canvas, "in1", center.x, center.y)
            self.conn_list = [self.in1, self.out]
        elif num_ports == 3:
            self.ba = Connector(self.canvas, "base", center.x, center.y)
            self.em = Connector(self.canvas, "emitter", center.x, center.y)
            self.co = Connector(self.canvas, "collector", center.x, center.y)
            self.conn_list = [self.ba, self.em, self.co]

    def update_connectors(self):
        """Update the position of connectors here"""
        center, e, w, n, s = self.get_geometry()

        conn_loc = self.conn_params['conn_loc'][self.comp_type]
        if conn_loc == 'ew':  # 2-port with ew ports
            self.calc_ew_conn_rotation(n, s, e, w)
        elif conn_loc == 'ns':  # 2-port with ns ports
            self.calc_ns_conn_rotation(n, s, e, w)
        elif conn_loc == 'n':  # 1-port with n port
            self.calc_n_conn_rotation(n, w)
        elif conn_loc == 'nsw':  # 3-port with nsw ports
            self.calc_nsw_conn_rotation(n, s, e, w)
        for c in self.conn_list:
            c.update()

        self.move_connected_wires()

    def calc_ew_conn_rotation(self, n, s, e, w):
        if self.angle == 0 or self.angle == 180:
            self.out.x, self.out.y = w.x, w.y
            self.in1.x, self.in1.y = e.x, e.y
        elif self.angle == 90 or self.angle == 270:
            self.out.x, self.out.y = n.x, n.y
            self.in1.x, self.in1.y = s.x, s.y

    def calc_ns_conn_rotation(self, n, s, e, w):
        if self.angle == 0 or self.angle == 180:
            self.out.x, self.out.y = n.x, n.y
            self.in1.x, self.in1.y = s.x, s.y
        elif self.angle == 90 or self.angle == 270:
            self.out.x, self.out.y = w.x, w.y
            self.in1.x, self.in1.y = e.x, e.y

    def calc_n_conn_rotation(self, n, w):
        if self.angle == 0 or self.angle == 180:
            self.out.x, self.out.y = n.x, n.y
        elif self.angle == 90 or self.angle == 270:
            self.out.x, self.out.y = w.x, w.y

    def calc_nsw_conn_rotation(self, n, s, e, w):
        if self.angle == 0 or self.angle == 180:
            self.em.x, self.em.y = n.x, n.y
            self.co.x, self.co.y = s.x, s.y
            self.ba.x, self.ba.y = w.x, w.y
        elif self.angle == 90 or self.angle == 270:
            self.em.x, self.em.y = w.x, w.y
            self.co.x, self.co.y = e.x, e.y
            self.ba.x, self.ba.y = s.x, s.y

    def get_geometry(self):
        sign = lambda angle: 1 if angle == 0 or angle == 180 else -1

        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        w = x2 - x1
        h = y2 - y1
        center = Point(x1 + w / 2, y1 + h / 2)

        if self.comp_type == 'npn_transistor':
            e = Point(center.x + sign(self.angle) * w / 2, center.y)
            w = Point(center.x - sign(self.angle) * w / 2, center.y)
            n = Point(center.x + 20, center.y - sign(self.angle) * h / 2)
            s = Point(center.x + 20, center.y + sign(self.angle) * h / 2)
        else:
            e = Point(center.x - sign(self.angle) * w / 2, center.y)
            w = Point(center.x + sign(self.angle) * w / 2, center.y)
            n = Point(center.x, center.y - sign(self.angle) * h / 2)
            s = Point(center.x, center.y + sign(self.angle) * h / 2)

        return center, e, w, n, s

    def __repr__(self):
        return ("Type: " + self.comp_type + " x1: " + str(self.x1) + " y1: " +
str(self.y1) + " value: " +
                str(self.value) + " wire list: " +
str(self.wire_list.__repr__()))

    def reprJson(self):
        return dict(type=self.comp_type, x1=self.x1, y1=self.y1,
angle=self.angle, value=self.value,
                    wire_list=self.wire_list)
```

Wire_Lib/node.py

```
from Helper_Lib import Point
from Comp_Lib import Connector


class Node:
    def __init__(self, canvas, x1, y1):
        self.canvas = canvas
        self.x1 = x1
        self.y1 = y1

        self.text = '0'
        self.comp_type = 'node'
        self.id = None
        self.text_id = None
        self.sel_id = None
        self.wire_list = []
        self.bbox = None
        self.angle = 0
        self.is_selected = False
        self.is_drawing = False
        self.conn = None
        self.conn_list = []

        self.create()

    def create(self):
        self.create_node()
        self.update_bbox()
        self.create_text()
        self.create_selector()
        self.create_connectors()
        self.set_connector_visibility()

    def create_node(self):
        self.id = self.canvas.create_oval(self.x1 - 5, self.y1 - 5, self.x1 +
5, self.y1 + 5, fill="black")

    def update_bbox(self):
        """Update the bounding box to get current gate coordinates"""
        self.bbox = self.canvas.bbox(self.id)

    def create_text(self):
        text_loc = Point(self.x1-10, self.y1-10)  # Put text above symbol
        self.text_id = self.canvas.create_text(text_loc.x, text_loc.y,
                                text=self.text, fill="black",
                                font='Helvetica 10 bold',
                                angle=self.angle, tags="text")

    def create_selector(self):
        """Create the red rectangle selector and check to see if the gate is
selected"""
        x1, y1, x2, y2 = self.bbox[0] - 5, self.bbox[1] - 5, self.bbox[2] + 5,
self.bbox[3] + 5
        self.sel_id = self.canvas.create_rectangle(x1, y1, x2, y2, fill=None,
outline="red", width=2)
        self.set_selector_visibility()

    def set_selector_visibility(self):
        """Set the selector visibility state"""
        if self.is_selected:
            self.canvas.itemconfig(self.sel_id, state='normal')
        else:
            self.canvas.itemconfig(self.sel_id, state='hidden')
    def create_connectors(self):
        # Calculate position of connectors from current comp position and size
        center = self.get_geometry()

        self.conn = Connector(self.canvas, "conn", center.x, center.y)
        self.conn_list = [self.conn]

    def get_geometry(self):
        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        w = x2 - x1
        h = y2 - y1
        center = Point(x1 + w / 2, y1 + h / 2)

        return center

    def set_connector_visibility(self):
        """Set the connector visibility state"""
        if self.is_drawing:
            for c in self.conn_list:
                self.canvas.itemconfig(c.id, state='normal')
        else:
            for c in self.conn_list:
                self.canvas.itemconfig(c.id, state='hidden')

    def update(self):
        self.update_position()
        self.update_bbox()
        self.update_text()
        self.update_selector()
        self.update_connectors()
        self.set_connector_visibility()

    def update_position(self):
        self.canvas.coords(self.id, self.x1 - 5, self.y1 - 5, self.x1 + 5,
self.y1 + 5)  # Update position

    def update_text(self):
        self.canvas.itemconfig(self.text_id, text=self.text)
        self.canvas.coords(self.text_id, self.x1 - 10, self.y1 - 10)

    def update_selector(self):
        """Update the red rectangle selector coordinates and check to see if
the gate is selected"""
        x1, y1, x2, y2 = self.bbox[0] - 5, self.bbox[1] - 5, self.bbox[2] + 5,
self.bbox[3] + 5
        self.canvas.coords(self.sel_id, x1, y1, x2, y2)
        self.set_selector_visibility()

    def update_connectors(self):
        """Update the position of connectors here"""
        center = self.get_geometry()

        self.conn.x, self.conn.y = center.x, center.y

        for c in self.conn_list:
            c.update()

        self.move_connected_wires()

    def check_connector_hit(self, x, y):
        """Hit test to see if a connector is at the provided x, y
coordinates"""
        for conn in self.conn_list:
            if conn.connector_hit_test(x, y):
                return conn
        return None

    def move_connected_wires(self):
        """Resize connected wires if the shape is moved"""
        for connection in self.wire_list:  # comp_conn, wire_name, wire_end
            for connector in self.conn_list:
                if connector.name == connection.comp_conn:
                    wire_obj = self.canvas.wire_dict[connection.wire_name]
                    if connection.wire_end == "begin":
                        wire_obj.x1 = connector.x
                        wire_obj.y1 = connector.y
                    elif connection.wire_end == "end":
                        wire_obj.x2 = connector.x
                        wire_obj.y2 = connector.y

    def rotate(self):
        """Set the rotation angle to the current angle + 90 deg, reset to 0 deg
if angle > 270 deg"""
        self.angle += 90
        if self.angle > 270:
            self.angle = 0

    def hit_test(self, x, y):
        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        if x1 <= x <= x2 and y1 <= y <= y2:
            self.is_selected = True
        else:
            self.is_selected = False
```

UI_Lib/canvas.py

```
import customtkinter as ctk

from UI_Lib.mouse import Mouse
from UI_Lib.grid import Grid
from Wire_Lib import Node


class Canvas(ctk.CTkCanvas):
    def __init__(self, parent):
        super().__init__(parent)
        self.led_color = "red"
        self.led_size = "large"
        self.grid = Grid(self, 10)
        self.mouse = Mouse(self)
        self.comp_list = []
        self.wire_list = []
        self.conn_list = []
        self.wire_dict = {}

        self.mouse.move_mouse_bind_events()

    def redraw(self):
        self.delete('grid_line')
        self.grid.draw()
        self.tag_lower("grid_line")
        for c in self.comp_list:
            c.update()

    def show_connectors(self):
        for s in self.comp_list:
            s.is_drawing = True
        self.redraw()

    def hide_connectors(self):
        for s in self.comp_list:
            s.is_drawing = False
        self.redraw()

    def edit_shape(self, _event=None):
        if self.mouse.selected_comp:
            comp = self.mouse.selected_comp
            if isinstance(comp, Node):
                dialog = ctk.CTkInputDialog(text="Enter new text", title="Edit Node Text")
                comp.text = dialog.get_input()
                self.redraw()
```

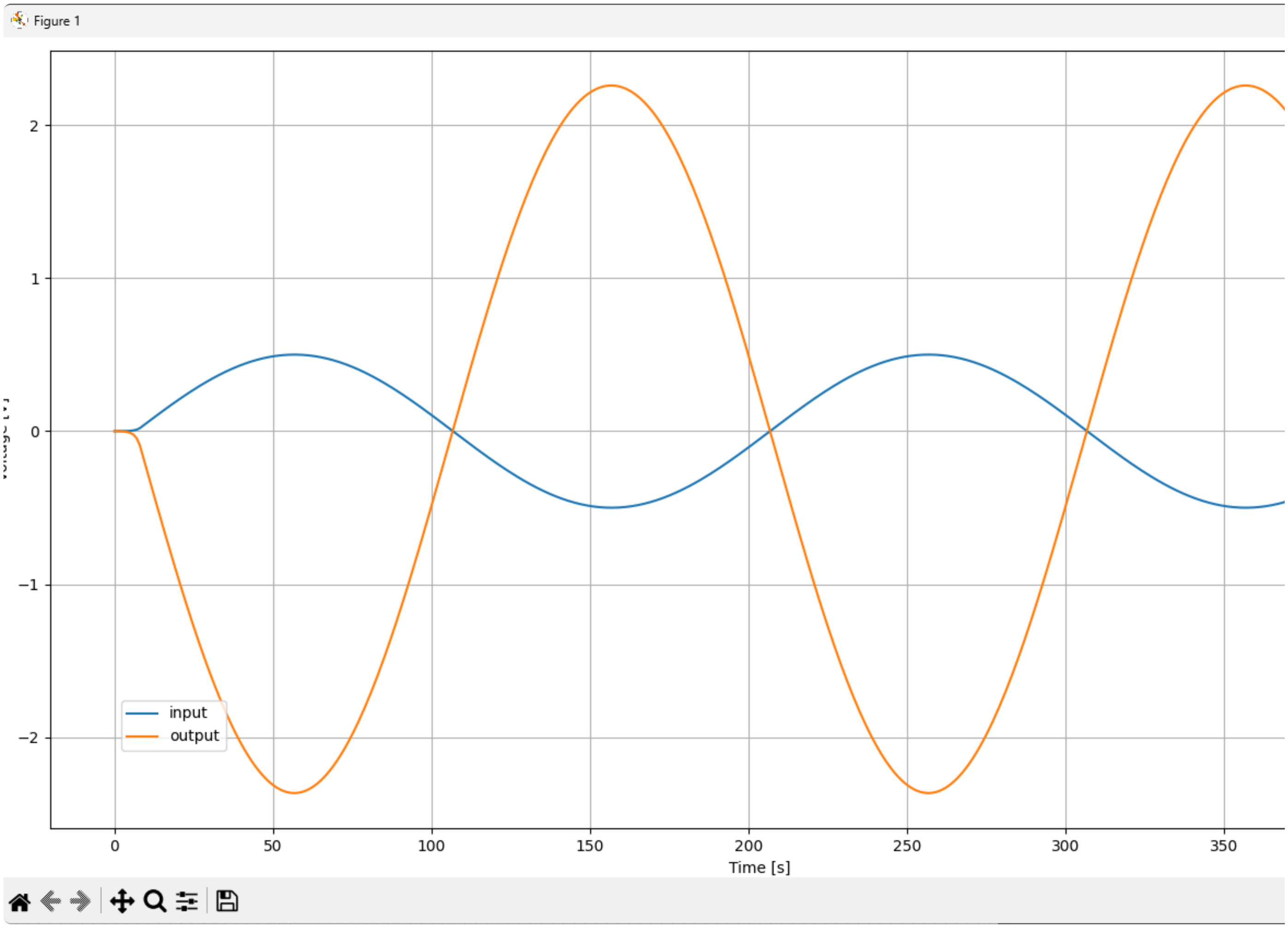
analog_simulator.py

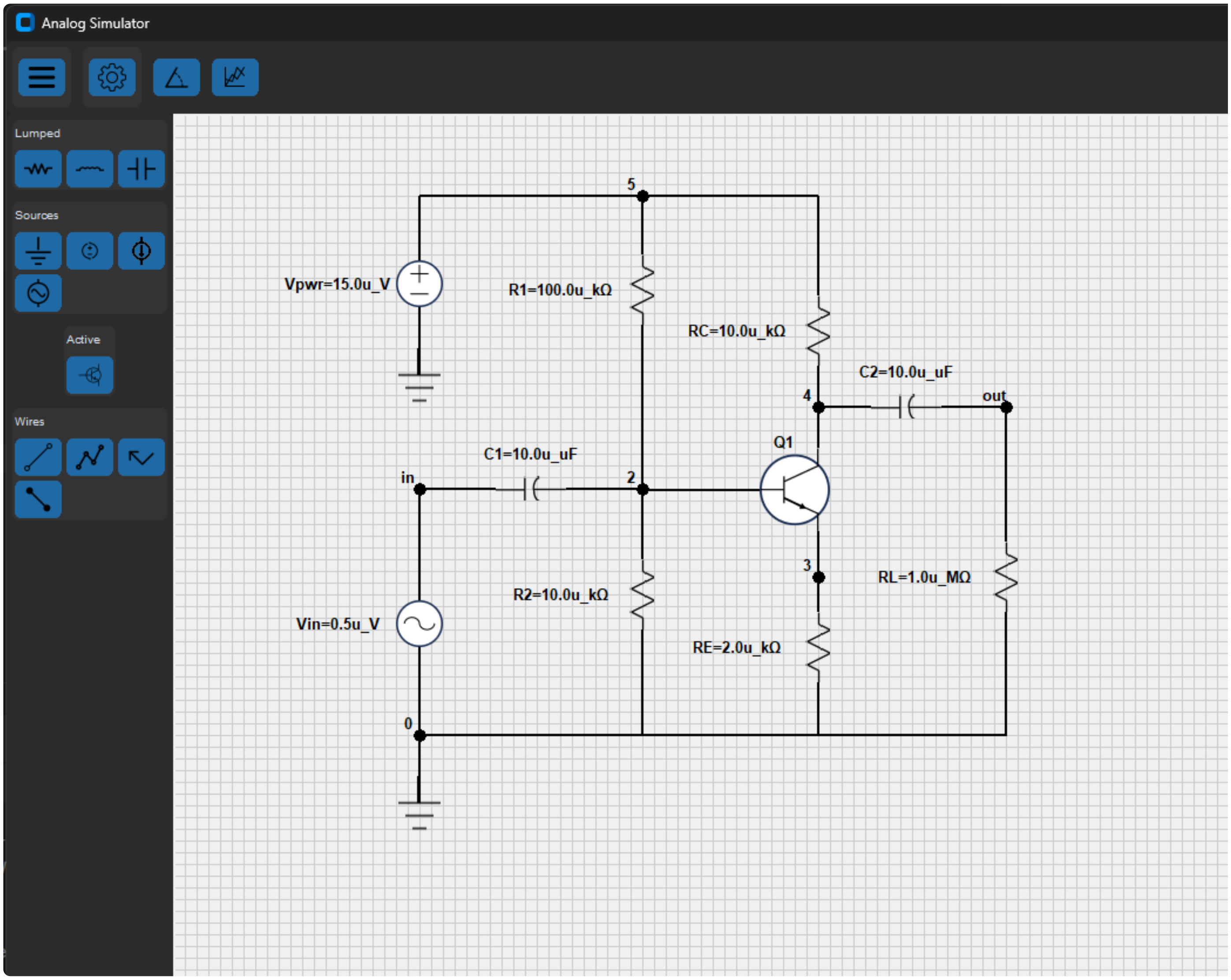
```
import customtkinter as ctk
from UI_Lib import LeftFrame, TopFrame, Canvas

ctk.set_appearance_mode("System")  # Modes: "System" (standard), "Dark", "Light"
ctk.set_default_color_theme("blue")  # Themes: "blue" (standard), "green", "dark-blue"


class App(ctk.CTk):
    """ctk.CTk is a CustomTkinter main window, similar to tk.Tk tkinter main window"""
    def __init__(self):
        super().__init__()
        self.geometry("1200x800x100x100")  # w, h, x, y
        self.title("Analog Simulator")

        self.canvas = Canvas(self)
        self.left_frame = LeftFrame(self, self.canvas)
        self.top_frame = TopFrame(self, self.canvas)

        self.top_frame.pack(side=ctk.TOP, fill=ctk.BOTH)
        self.left_frame.pack(side=ctk.LEFT, fill=ctk.BOTH)
        self.canvas.pack(side=ctk.LEFT, fill=ctk.BOTH, expand=True)

        # Add bindings here
        self.bind("<Configure>", self.on_window_resize)
        self.bind('<r>', self.rotate_comp)
        self.canvas.bind('<Button-3>', self.canvas.edit_shape)

    def on_window_resize(self, _event):
        self.canvas.redraw()
    def rotate_comp(self, _event=None):
        if self.canvas.mouse.selected_comp:
            self.canvas.mouse.selected_comp.rotate()
            self.canvas.redraw()


if __name__ == "__main__":
    """Instantiate the Microwave Simulator app and run the main loop"""
    app = App()
    app.mainloop()
```

Transistor Amplifier Circuit

Objective:

- ✓ Convert Canvas Component List to PySpice netlist
- ✓ Run Analysis





Circuits/`transistor-amplifier.json`

```
[
    {
        "comp_list": [
            {
                "type": "dc_source",
                "text": "Vpwr",
                "x1": 210,
                "y1": 145,
                "angle": 0,
                "value": "15",
                "units": "u_V",
                "wire_list": [
                    {
                        "type": "connection",
                        "comp_conn": "out",
                        "wire_name": "wire1",
                        "wire_end": "begin"
                    },
                    {
                        "type": "connection",
                        "comp_conn": "in1",
                        "wire_name": "wire2",
                        "wire_end": "begin"
                    }
                ]
            },
            {
                "type": "resistor",
                "text": "R1",
                "x1": 400,
                "y1": 150,
                "angle": 90,
                "value": "100",
                "units": "u_k\u03a9",
                "wire_list": [
                    {
                        "type": "connection",
                        "comp_conn": "in1",
                        "wire_name": "wire7",
                        "wire_end": "end"
                    },
                    {
                        "type": "connection",
                        "comp_conn": "out",
                        "wire_name": "wire8",
                        "wire_end": "begin"
                    }
                ]
            },
            {
                "type": "npn_transistor",
                "text": "Q1",
                "x1": 530,
                "y1": 320,
                "angle": 0,
                "bf": 80,
                "cjc": 5,
                "cjc_units": "u_pF",
                "rb": 100,
                "wire_list": [
                    {
                        "type": "connection",
                        "comp_conn": "base",
                        "wire_name": "wire10",
                        "wire_end": "end"
                    },
                    {
                        "type": "connection",
                        "comp_conn": "emitter",
                        "wire_name": "wire14",
                        "wire_end": "end"
                    },
                    {
                        "type": "connection",
                        "comp_conn": "collector",
                        "wire_name": "wire18",
                        "wire_end": "begin"
                    }
                ]
            },
            {
                "type": "resistor",
                "text": "RC",
                "x1": 550,
                "y1": 185,
                "angle": 90,
                "value": "10",
                "units": "u_k\u03a9",
                "wire_list": [
                    {
                        "type": "connection",
                        "comp_conn": "in1",
                        "wire_name": "wire12",
                        "wire_end": "end"
                    },
                    {
                        "type": "connection",
                        "comp_conn": "out",
                        "wire_name": "wire13",
                        "wire_end": "begin"
                    }
                ]
            },
            {
                "type": "node",
                "text": "5",
                "x1": 400,
                "y1": 70,
                "angle": 0,
                "wire_list": [
                    {
                        "type": "connection",
                        "comp_conn": "conn",
                        "wire_name": "wire1",
                        "wire_end": "end"
                    },
                    {
                        "type": "connection",
                        "comp_conn": "conn",
                        "wire_name": "wire7",
                        "wire_end": "begin"
                    },
                    {
                        "type": "connection",
                        "comp_conn": "conn",
                        "wire_name": "wire12",
                        "wire_end": "begin"
                    }
                ]
            },
            {
                "type": "node",
                "text": "4",
                "x1": 550,
                "y1": 250,
                "angle": 0,
                "wire_list": [
                    {
                        "type": "connection",
                        "comp_conn": "conn",
                        "wire_name": "wire13",
                        "wire_end": "end"
                    },
                    {
                        "type": "connection",
                        "comp_conn": "conn",
                        "wire_name": "wire14",
                        "wire_end": "begin"
                    },
                    {
                        "type": "connection",
                        "comp_conn": "conn",
                        "wire_name": "wire15",
                    "wire_end": "begin"
                }
            ]
        },
        {
            "type": "node",
            "text": "3",
            "x1": 550,
            "y1": 395,
            "angle": 0,
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "conn",
                    "wire_name": "wire18",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "comp_conn": "conn",
                    "wire_name": "wire19",
                    "wire_end": "begin"
                }
            ]
        },
        {
            "type": "node",
            "text": "2",
            "x1": 400,
            "y1": 320,
            "angle": 0,
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "conn",
                    "wire_name": "wire8",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "comp_conn": "conn",
                    "wire_name": "wire9",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "comp_conn": "conn",
                    "wire_name": "wire10",
                    "wire_end": "begin"
                },
                {
                    "type": "connection",
                    "comp_conn": "conn",
                    "wire_name": "wire11",
                    "wire_end": "begin"
                }
            ]
        },
        {
            "type": "capacitor",
            "text": "C1",
            "x1": 305,
            "y1": 320,
            "angle": 0,
            "value": "10",
            "units": "u_uF",
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "in1",
                    "wire_name": "wire3",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "comp_conn": "out",
                    "wire_name": "wire9",
                    "wire_end": "begin"
                }
            ]
        },
        {
            "type": "ac_source",
            "text": "Vin",
            "x1": 210,
            "y1": 435,
            "angle": 0,
            "value": "0.5",
            "units": "u_V",
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "out",
                    "wire_name": "wire4",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "comp_conn": "in1",
                    "wire_name": "wire5",
                    "wire_end": "begin"
                }
            ]
        },
        {
            "type": "node",
            "text": "in",
            "x1": 210,
            "y1": 320,
            "angle": 0,
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "conn",
                    "wire_name": "wire3",
                    "wire_end": "begin"
                },
                {
                    "type": "connection",
                    "comp_conn": "conn",
                    "wire_name": "wire4",
                    "wire_end": "begin"
                }
            ]
        },
        {
            "type": "ground",
            "text": null,
            "x1": 210,
            "y1": 225,
            "angle": 0,
            "value": 0,
            "units": null,
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "in1",
                    "wire_name": "wire2",
                    "wire_end": "end"
                }
            ]
        },
        {
            "type": "ground",
            "text": null,
            "x1": 210,
            "y1": 590,
            "angle": 0,
            "value": 0,
            "units": null,
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "in1",
                    "wire_name": "wire6",
                    "wire_end": "end"
                }
            ]
        },
        {
            "type": "node",
            "text": "0",
            "x1": 210,
            "y1": 530,
            "angle": 0,
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "conn",
                    "wire_name": "wire5",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "comp_conn": "conn",
                    "wire_name": "wire6",
                    "wire_end": "begin"
                },
                {
                    "type": "connection",
                    "comp_conn": "conn",
                    "wire_name": "wire20",
                    "wire_end": "begin"
                },
                {
                    "type": "connection",
                    "comp_conn": "conn",
                    "wire_name": "wire21",
                    "wire_end": "begin"
                },
                {
                    "type": "connection",
                    "comp_conn": "conn",
                    "wire_name": "wire22",
                    "wire_end": "begin"
                }
            ]
        },
        {
            "type": "resistor",
            "text": "RE",
            "x1": 550,
            "y1": 455,
            "angle": 90,
            "value": "2",
            "units": "u_k\u03a9",
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "in1",
                    "wire_name": "wire19",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "comp_conn": "out",
                    "wire_name": "wire21",
                    "wire_end": "end"
                }
            ]
        },
        {
            "type": "resistor",
            "text": "R2",
            "x1": 400,
            "y1": 410,
            "angle": 90,
            "value": "10",
            "units": "u_k\u03a9",
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "in1",
                    "wire_name": "wire11",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "comp_conn": "out",
                    "wire_name": "wire22",
                    "wire_end": "end"
                }
            ]
        },
        {
            "type": "capacitor",
            "text": "C2",
            "x1": 625,
            "y1": 250,
            "angle": 0,
            "value": "10",
            "units": "u_uF",
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "in1",
                    "wire_name": "wire15",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "comp_conn": "out",
                    "wire_name": "wire16",
                    "wire_end": "begin"
                }
            ]
        },
        {
            "type": "node",
            "text": "out",
            "x1": 710,
            "y1": 250,
            "angle": 0,
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "conn",
                    "wire_name": "wire16",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "comp_conn": "conn",
                    "wire_name": "wire17",
                    "wire_end": "begin"
                }
            ]
        },
        {
            "type": "resistor",
            "text": "RL",
            "x1": 710,
            "y1": 395,
            "angle": 90,
            "value": "1",
            "units": "u_M\u03a9",
            "wire_list": [
                {
                    "type": "connection",
                    "comp_conn": "in1",
                    "wire_name": "wire17",
                    "wire_end": "end"
                },
                {
                    "type": "connection",
                    "comp_conn": "out",
                    "wire_name": "wire20",
                    "wire_end": "end"
                }
            ]
        },
        {
            "type": "elbow",
            "wire_dir": "V",
            "node_num": "5",
            "x1": 210.0,
            "y1": 125.0,
            "x2": 400.0,
            "y2": 70.0,
            "name": "wire1"
        },
        {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "0",
                "x1": 210.0,
                "y1": 165.0,
                "x2": 210.0,
                "y2": 200.0,
                "name": "wire2"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "in",
                "x1": 210.0,
                "y1": 320.0,
                "x2": 275.0,
                "y2": 320.0,
                "name": "wire3"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "in",
                "x1": 210.0,
                "y1": 320.0,
                "x2": 210.0,
                "y2": 415.0,
                "name": "wire4"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "0",
                "x1": 210.0,
                "y1": 455.0,
                "x2": 210.0,
                "y2": 530.0,
                "name": "wire5"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "0",
                "x1": 210.0,
                "y1": 530.0,
                "x2": 210.0,
                "y2": 565.0,
                "name": "wire6"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "5",
                "x1": 400.0,
                "y1": 70.0,
                "x2": 400.0,
                "y2": 120.0,
                "name": "wire7"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "2",
                "x1": 400.0,
                "y1": 180.0,
                "x2": 400.0,
                "y2": 320.0,
                "name": "wire8"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "2",
                "x1": 335.0,
                "y1": 320.0,
                "x2": 400.0,
                "y2": 320.0,
                "name": "wire9"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "2",
                "x1": 400.0,
                "y1": 320.0,
                "x2": 500.0,
                "y2": 320.5,
                "name": "wire10"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "2",
                "x1": 400.0,
                "y1": 320.0,
                "x2": 400.0,
                "y2": 380.0,
                "name": "wire11"
            },
            {
                "type": "elbow",
                "wire_dir": "H",
                "node_num": "5",
                "x1": 400.0,
                "y1": 70.0,
                "x2": 550.0,
                "y2": 155.0,
                "name": "wire12"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "4",
                "x1": 550.0,
                "y1": 215.0,
                "x2": 550.0,
                "y2": 250.0,
                "name": "wire13"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "4",
                "x1": 550.0,
                "y1": 250.0,
                "x2": 550.0,
                "y2": 285.0,
                "name": "wire14"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "4",
                "x1": 550.0,
                "y1": 250.0,
                "x2": 595.0,
                "y2": 250.0,
                "name": "wire15"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "out",
                "x1": 655.0,
                "y1": 250.0,
                "x2": 710.0,
                "y2": 250.0,
                "name": "wire16"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "out",
                "x1": 710.0,
                "y1": 250.0,
                "x2": 710.0,
                "y2": 365.0,
                "name": "wire17"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "3",
                "x1": 550.0,
                "y1": 356.0,
                "x2": 550.0,
                "y2": 395.0,
                "name": "wire18"
            },
            {
                "type": "straight",
                "wire_dir": "H",
                "node_num": "3",
                "x1": 550.0,
                "y1": 395.0,
                "x2": 550.0,
                "y2": 425.0,
                "name": "wire19"
            },
            {
                "type": "elbow",
                "wire_dir": "H",
                "node_num": "0",
                "x1": 210.0,
                "y1": 530.0,
                "x2": 710.0,
                "y2": 425.0,
                "name": "wire20"
            },
            {
                "type": "elbow",
                "wire_dir": "H",
                "node_num": "0",
                "x1": 210.0,
                "y1": 530.0,
                "x2": 550.0,
                "y2": 485.0,
                "name": "wire21"
            },
            {
                "type": "elbow",
                "wire_dir": "H",
                "node_num": "0",
                "x1": 210.0,
                "y1": 530.0,
                "x2": 400.0,
                "y2": 440.0,
                "name": "wire22"
            }
        ]
    }
]
```

Final source code

Comp_Lib/`__init__.py`

```python
# Import lumped components
from Comp_Lib.analog_component import AnalogComponent

# Import connector related classes
from Comp_Lib.connector import Connector
from Comp_Lib.connection import Connection
```

Comp_Lib/component.py

```python
import tkinter as tk
from PIL import Image, ImageTk
from pathlib import Path

from Helper_Lib import Point


class Component:
    def __init__(self, canvas, comp_type, x1, y1, value):
        """Base class for component classes"""
        self.canvas = canvas
        self.comp_type = comp_type
        self.x1 = x1
        self.y1 = y1
        self.value = value
        self.units = None

        if self.comp_type == 'npn_transistor':
            self.bf = None
            self.cjc = None
            self.cjc_units = None
            self.rb = None

        self.id = None
        self.sel_id = None

        self.is_selected = False
        self.is_drawing = False
        self.selector = None
        self.angle = 0
        self.text = None
        self.comp_text = None
        self.text_id = None

        self.filename = None
        self.a_image = None
        self.ph_image = None
        self.bbox = None

        self.out = None
        self.in1 = None
        self.ba = None
        self.em = None
        self.co = None

        self.conn_list = []
        self.wire_list = []

        # Define component parameters in a dictionary of dictionaries
        self.params = {
            'filename': {
                'resistor': "../images/lumped/resistor_60x30.png",
                'capacitor': "../images/lumped/capacitor_60x30.png",
                'inductor': "../images/lumped/inductor_60x30.png",
                'ground': "../images/sources/ground_50x40.png",
                'dc_source': "../images/sources/voltage_source_40x40.png",
                'isource': "../images/sources/current_source_40x40.png",
                'ac_source': "../images/sources/ac_voltage_source_40x40.png",
                'npn_transistor':
"../images/transistors/npn_transistor_60x71.png"
            },
            'text': {
                'resistor': 'R1',
                'capacitor': 'C1',
                'inductor': 'L1',
                'ground': "",
                'dc_source': "Vpwr",
                'isource': "I",
                'ac_source': "Vin",
                'npn_transistor': "Q1"
            }
        }

    def set_image_filename(self):
        self.filename = Path(__file__).parent / self.params['filename']
[self.comp_type]

    def create_text(self):
        self.comp_text = self.params['text'][self.comp_type]
        if self.comp_type == 'isource' or self.comp_type == 'dc_source' or
self.comp_type == 'ac_source':
            text_loc = Point(self.x1-40, self.y1)  # Put text on left side of
symbol
        elif self.comp_type == 'npn_transistor':
            text_loc = Point(self.x1 - 10, self.y1 - 40)  # Put text above
symbol
        else:
            text_loc = Point(self.x1, self.y1-30)  # Put text above symbol
        self.text_id = self.canvas.create_text(text_loc.x, text_loc.y,
                                text=self.comp_text, fill="black",
                                font='Helvetica 10 bold',
                                angle=self.angle, tags="text")

    def create_image(self, filename):
        """Initial component image creation"""
        self.a_image = Image.open(filename)
        self.a_image = self.a_image.rotate(self.angle, expand=True)
        self.ph_image = ImageTk.PhotoImage(self.a_image)
        self.id = self.canvas.create_image(self.x1, self.y1, anchor=tk.CENTER,
image=self.ph_image, tags='gate')

    def update_position(self):
        """Update the position when the gate object is moved"""
        self.canvas.coords(self.id, self.x1, self.y1)  # Update position

    def update_image(self, filename):
        """Update the image for gate symbol rotation"""
        self.a_image = Image.open(filename)
        self.a_image = self.a_image.rotate(self.angle, expand=True)  # Update
image rotation
        self.ph_image = ImageTk.PhotoImage(self.a_image)
        self.canvas.itemconfig(self.id, image=self.ph_image)  # Update image

    def update_bbox(self):
        """Update the bounding box to get current gate coordinates"""
        self.bbox = self.canvas.bbox(self.id)

    def update_text(self):
        self.canvas.itemconfig(self.text_id, text=self.comp_text)
        if self.comp_type == 'isource' or self.comp_type == 'dc_source' or
self.comp_type == 'ac_source':
            if self.angle == 0 or self.angle == 180:
                self.set_east_text()
            elif self.angle == 90 or self.angle == 270:
                self.set_north_text()
        elif self.comp_type == 'npn_transistor':
            self.set_transistor_text()
        else:
            if self.angle == 0 or self.angle == 180:
                self.set_north_text()
            elif self.angle == 90 or self.angle == 270:
                self.set_east_text()

    def set_east_text(self):
        self.canvas.coords(self.text_id, self.x1 - 70, self.y1)
    def set_north_text(self):
        self.canvas.coords(self.text_id, self.x1, self.y1 - 30)

    def set_transistor_text(self):
        self.canvas.itemconfig(self.text_id, text=self.text)
        self.canvas.coords(self.text_id, self.x1 - 10, self.y1 - 40)

    def create_selector(self):
        """Create the red rectangle selector and check to see if the gate is selected"""
        x1, y1, x2, y2 = self.bbox[0] - 5, self.bbox[1] - 5, self.bbox[2] + 5, self.bbox[3] + 5
        self.sel_id = self.canvas.create_rectangle(x1, y1, x2, y2, fill=None, outline="red", width=2)
        self.set_selector_visibility()

    def update_selector(self):
        """Update the red rectangle selector coordinates and check to see if the gate is selected"""
        x1, y1, x2, y2 = self.bbox[0] - 5, self.bbox[1] - 5, self.bbox[2] + 5, self.bbox[3] + 5
        self.canvas.coords(self.sel_id, x1, y1, x2, y2)
        self.set_selector_visibility()

    def set_selector_visibility(self):
        """Set the selector visibility state"""
        if self.is_selected:
            self.canvas.itemconfig(self.sel_id, state='normal')
        else:
            self.canvas.itemconfig(self.sel_id, state='hidden')

    def set_connector_visibility(self):
        """Set the connector visibility state"""
        if self.is_drawing:
            for c in self.conn_list:
                self.canvas.itemconfig(c.id, state='normal')
        else:
            for c in self.conn_list:
                self.canvas.itemconfig(c.id, state='hidden')

    def check_connector_hit(self, x, y):
        """Hit test to see if a connector is at the provided x, y coordinates"""
        for conn in self.conn_list:
            if conn.connector_hit_test(x, y):
                return conn
        return None

    def move_connected_wires(self):
        """Resize connected wires if the shape is moved"""
        for connection in self.wire_list:  # comp_conn, wire_name, wire_end
            for connector in self.conn_list:
                if connector.name == connection.comp_conn:
                    wire_obj = self.canvas.wire_dict[connection.wire_name]
                    if connection.wire_end == "begin":
                        wire_obj.x1 = connector.x
                        wire_obj.y1 = connector.y
                    elif connection.wire_end == "end":
                        wire_obj.x2 = connector.x
                        wire_obj.y2 = connector.y

    def rotate(self):
        """Set the rotation angle to the current angle + 90 deg, reset to 0 deg if angle > 270 deg"""
        self.angle += 90
        if self.angle > 270:
            self.angle = 0

    def hit_test(self, x, y):
        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        if x1 <= x <= x2 and y1 <= y <= y2:
            self.is_selected = True
        else:
            self.is_selected = False
```

Comp_Lib/analog_component.py

```
from Comp_Lib.component import Component
from Comp_Lib.connector import Connector
from Helper_Lib import Point


class AnalogComponent(Component):
    """Universal model for analog components"""
    def __init__(self, canvas, comp_type, x1, y1, value=0):
        super().__init__(canvas, comp_type, x1, y1, value)

        self.conn_params = {
            'ports': {
                'resistor': 2,
                'capacitor': 2,
                'inductor': 2,
                'ground': 1,
                'dc_source': 2,
                'isource': 2,
                'ac_source': 2,
                'npn_transistor': 3
            },
            'conn_loc': {
                'resistor': 'ew',
                'capacitor': 'ew',
                'inductor': 'ew',
                'ground': 'n',
                'dc_source': 'ns',
                'isource': 'ns',
                'ac_source': 'ns',
                'npn_transistor': 'nsw'
            }
        }

        self.create()

    def create(self):
        self.set_image_filename()
        self.create_image(self.filename)
        self.update_bbox()
        self.create_text()
        self.create_selector()
        self.create_connectors()
        self.set_connector_visibility()

    def update(self):
        self.update_position()
        self.update_image(self.filename)
        self.update_bbox()
        self.update_text()
        self.update_selector()
        self.update_connectors()
        self.set_connector_visibility()

    def create_connectors(self):
        # Calculate position of connectors from current comp position and size
        center, e, w, n, s = self.get_geometry()
        num_ports = self.conn_params['ports'][self.comp_type]
        if num_ports == 1:
            self.out = Connector(self.canvas, "in1", center.x, center.y)
            self.conn_list = [self.out]
        elif num_ports == 2:
            self.out = Connector(self.canvas, "out", center.x, center.y)
            self.in1 = Connector(self.canvas, "in1", center.x, center.y)
            self.conn_list = [self.in1, self.out]
        elif num_ports == 3:
            self.ba = Connector(self.canvas, "base", center.x, center.y)
            self.em = Connector(self.canvas, "emitter", center.x, center.y)
            self.co = Connector(self.canvas, "collector", center.x, center.y)
            self.conn_list = [self.ba, self.em, self.co]

    def update_connectors(self):
        """Update the position of connectors here"""
        center, e, w, n, s = self.get_geometry()

        conn_loc = self.conn_params['conn_loc'][self.comp_type]
        if conn_loc == 'ew':  # 2-port with ew ports
            self.calc_ew_conn_rotation(n, s, e, w)
        elif conn_loc == 'ns':  # 2-port with ns ports
            self.calc_ns_conn_rotation(n, s, e, w)
        elif conn_loc == 'n':  # 1-port with n port
            self.calc_n_conn_rotation(n, w)
        elif conn_loc == 'nsw':  # 3-port with nsw ports
            self.calc_nsw_conn_rotation(n, s, e, w)

        for c in self.conn_list:
            c.update()

        self.move_connected_wires()

    def calc_ew_conn_rotation(self, n, s, e, w):
        if self.angle == 0 or self.angle == 180:
            self.out.x, self.out.y = w.x, w.y
            self.in1.x, self.in1.y = e.x, e.y
        elif self.angle == 90 or self.angle == 270:
            self.out.x, self.out.y = n.x, n.y
            self.in1.x, self.in1.y = s.x, s.y

    def calc_ns_conn_rotation(self, n, s, e, w):
        if self.angle == 0 or self.angle == 180:
            self.out.x, self.out.y = n.x, n.y
            self.in1.x, self.in1.y = s.x, s.y
        elif self.angle == 90 or self.angle == 270:
            self.out.x, self.out.y = w.x, w.y
            self.in1.x, self.in1.y = e.x, e.y

    def calc_n_conn_rotation(self, n, w):
        if self.angle == 0 or self.angle == 180:
            self.out.x, self.out.y = n.x, n.y
        elif self.angle == 90 or self.angle == 270:
            self.out.x, self.out.y = w.x, w.y

    def calc_nsw_conn_rotation(self, n, s, e, w):
        if self.angle == 0 or self.angle == 180:
            self.em.x, self.em.y = n.x, n.y
            self.co.x, self.co.y = s.x, s.y
            self.ba.x, self.ba.y = w.x, w.y
        elif self.angle == 90 or self.angle == 270:
            self.em.x, self.em.y = w.x, w.y
            self.co.x, self.co.y = e.x, e.y
            self.ba.x, self.ba.y = s.x, s.y

    def get_geometry(self):
        sign = lambda angle: 1 if angle == 0 or angle == 180 else -1

        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        w = x2 - x1
        h = y2 - y1
        center = Point(x1 + w / 2, y1 + h / 2)

        if self.comp_type == 'npn_transistor':
            e = Point(center.x + sign(self.angle) * w / 2, center.y)
            w = Point(center.x - sign(self.angle) * w / 2, center.y)
            n = Point(center.x + 20, center.y - sign(self.angle) * h / 2)
            s = Point(center.x + 20, center.y + sign(self.angle) * h / 2)
        else:
            e = Point(center.x - sign(self.angle) * w / 2, center.y)
            w = Point(center.x + sign(self.angle) * w / 2, center.y)
            n = Point(center.x, center.y - sign(self.angle) * h / 2)
            s = Point(center.x, center.y + sign(self.angle) * h / 2)

        return center, e, w, n, s

    def __repr__(self):
        if self.comp_type == "npn_transistor":
            return ("Type: " + self.comp_type + " Comp Text: " + self.comp_text
+ " x1: " + str(self.x1) +
                    " y1: " + str(self.y1) + " bf: " + str(self.bf) + " cjc: "
+ str(self.cjc) + " cjc_units: " +
                    self.cjc_units + " rb: " + str(self.rb) + " wire list: " +
str(self.wire_list.__repr__()))
        else:
            return ("Type: " + self.comp_type + " Comp Text: " + self.comp_text
+ " x1: " + str(self.x1) + " y1: "
                    + str(self.y1) + " value: " + str(self.value) + " wire
list: " + str(self.wire_list.__repr__()))

    def reprJson(self):
        if self.comp_type == "npn_transistor":
            return dict(type=self.comp_type, text=self.text, x1=self.x1,
y1=self.y1, angle=self.angle,
                        bf=self.bf, cjc=self.cjc, cjc_units=self.cjc_units,
rb=self.rb, wire_list=self.wire_list)
        else:
            return dict(type=self.comp_type, text=self.text, x1=self.x1,
y1=self.y1, angle=self.angle,
                        value=self.value, units=self.units,
wire_list=self.wire_list)
```

Comp_Lib/connection.py

```
class Connection:
    def __init__(self, comp_conn, wire_name, wire_end):
        self.type = "connection"
        self.comp_conn = comp_conn   # in1, out
        self.wire_name = wire_name        # wire string name
        self.wire_end = wire_end         # "begin" or "end"

    def __repr__(self):
        return "Connector Name: " + self.comp_conn + \
               " Wire Name: " + self.wire_name + \
               " Wire End: " + self.wire_end

    def reprJson(self):
        return dict(type=self.type, comp_conn=self.comp_conn,
wire_name=self.wire_name, wire_end=self.wire_end)
```

Comp_Lib/connector.py

```
class Connector:
    def __init__(self, canvas, name, x, y):
        """Connector class"""
        self.canvas = canvas
        self.name = name
        self.x = x
        self.y = y

        self.id = None

        self.radius = 5
        self.x1, self.y1, self.x2, self.y2 = (self.x - self.radius, self.y -
self.radius,
                                              self.x + self.radius, self.y +
self.radius)

        self.create_connector()

    def create_connector(self):
        # Create the connector here
        points = [self.x - self.radius, self.y - self.radius, self.x +
self.radius, self.y + self.radius]
        self.id = self.canvas.create_oval(points, fill="white",
outline="black", width=2, tags='connector')

    def update(self):
        """Update the connector here"""
        self.x1, self.y1, self.x2, self.y2 = (self.x - self.radius, self.y -
self.radius,
                                              self.x + self.radius, self.y +
self.radius)
        points = [self.x - self.radius, self.y - self.radius, self.x +
self.radius, self.y + self.radius]
        self.canvas.coords(self.id, points)

    def set_pos(self, x, y):
        """Set the connector position here"""
        self.x = x
        self.y = y

    def connector_hit_test(self, x, y):
        """Connector hit test"""
        if self.x1 <= x <= self.x2 and self.y1 <= y <= self.y2:
            return True
        else:
            return False
```

```python
    def __repr__(self):
        return ("Connector: " + self.name + " (" + str(self.x1) + ", " +
str(self.y1) + ")" +
                " (" + str(self.x2) + ", " + str(self.y2) + ")")
```

Helper_Lib/`__init__.py`

```python
# Import helper classes
from .point import Point
```

Helper_Lib/point.py

```python
class Point:
    def __init__(self, x, y):
        self.x = x
        self.y = y

    def set(self, x, y):
        self.x = x
        self.y = y

    def __repr__(self):
        return "(" + str(self.x) + ", " + str(self.y) + ")"


if __name__ == "__main__":
    # Test block
    pt1 = Point(10, 10)
    pt2 = Point(100, 100)
    print("pt1: ", pt1)
    print("pt2: ", pt2)
```

UI_Lib/`__init__.py`

```python
# Import custom frame and canvas classes
from UI_Lib.left_frame import LeftFrame
from UI_Lib.top_frame import TopFrame
from UI_Lib.canvas import Canvas

# Import custom button frame classes
from UI_Lib.lump_button_frame import LumpButtonFrame
from UI_Lib.sources_button_frame import SourcesButtonFrame
from UI_Lib.wire_button_frame import WireButtonFrame

# Import custom classes
from UI_Lib.mouse import Mouse
from UI_Lib.grid import Grid
```

UI_Lib/active_button_frame.py

```
import customtkinter as ctk
from tktooltip import ToolTip
from pathlib import Path
from PIL import Image

from Comp_Lib import AnalogComponent


class ActiveButtonFrame(ctk.CTkFrame):
    def __init__(self, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas
        self.button_id = None

        self.button_list = [("npn_transistor", "../icons/npn-transistor.png")]

        self.init_frame_widgets()

    def init_frame_widgets(self):
        frame_name_label = ctk.CTkLabel(self, text="Active", font=("Helvetica",
10), height=20)
        frame_name_label.grid(row=0, column=0, columnspan=3, sticky=ctk.W,
padx=2, pady=2)

        row_num, col_num = 1, 0
        for button in self.button_list:
            a_image = ctk.CTkImage(light_image=Image.open
                                    (Path(__file__).parent / button[1]),
                                    dark_image=Image.open
                                    (Path(__file__).parent / button[1]),
                                    size=(24, 24))
            self.button_id = ctk.CTkButton(self, text="", image=a_image,
width=30,
                                           command=lambda
a_name=button[0]:self.create_events(a_name))
            self.button_id.grid(row=row_num, column=col_num, sticky=ctk.W,
padx=2, pady=2)
            ToolTip(self.button_id, msg=button[0])
            row_num, col_num = self.update_grid_numbers(row_num, col_num)

    def create_events(self, name):
        comp = None
        if name == "npn_transistor":
            comp = AnalogComponent(self.canvas, 'npn_transistor', 150, 150)
        self.canvas.comp_list.append(comp)
        self.canvas.redraw()
        self.canvas.mouse.move_mouse_bind_events()

    @staticmethod
    def update_grid_numbers(row, column):
        column += 1
        if column > 2:
            column = 0
            row += 1
        return row, column
```

UI_Lib/canvas.py

```
import customtkinter as ctk

from UI_Lib.mouse import Mouse
from UI_Lib.grid import Grid
from Wire_Lib import Node


class CompDialog(ctk.CTkToplevel):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.geometry("400x300")
        self.title("Component Dialog")
        self.text = ""
        self.value = 0
        self.units = ""

        self.text_entry = None
        self.value_entry = None
        self.units_entry = None

        self.initDialog()

    def initDialog(self):
        text_frame = ctk.CTkFrame(self)
        text_frame.pack(fill=ctk.X)

        text_lbl = ctk.CTkLabel(text_frame, text="Text", width=6)
        text_lbl.pack(side=ctk.LEFT, padx=5, pady=10)

        self.text_entry = ctk.CTkEntry(text_frame, placeholder_text="CTkEntry")
        self.text_entry.pack(fill=ctk.X, padx=5, expand=True)

        value_frame = ctk.CTkFrame(self)
        value_frame.pack(fill=ctk.X)

        value_lbl = ctk.CTkLabel(value_frame, text="Value", width=6)
        value_lbl.pack(side=ctk.LEFT, padx=5, pady=10)

        self.value_entry = ctk.CTkEntry(value_frame,
placeholder_text="CTkEntry")
        self.value_entry.pack(fill=ctk.X, padx=5, expand=True)

        units_frame = ctk.CTkFrame(self)
        units_frame.pack(fill=ctk.X)

        units_lbl = ctk.CTkLabel(units_frame, text="Units (Ω,nF,pF, etc.)",
width=6)
        units_lbl.pack(side=ctk.LEFT, padx=5, pady=10)

        self.units_entry = ctk.CTkEntry(units_frame,
placeholder_text="CTkEntry")
        self.units_entry.pack(fill=ctk.X, padx=5, expand=True)

        frame9 = ctk.CTkFrame(self)
        frame9.pack(fill=ctk.X)

        btn = ctk.CTkButton(frame9, text="Cancel", command=self.cancel,
width=60)
```

```
        btn.pack(side=ctk.RIGHT, padx=5, pady=5)

        btn = ctk.CTkButton(frame9, text="OK", command=self.onSubmit, width=60)
        btn.pack(side=ctk.RIGHT, padx=5, pady=5)

    def onSubmit(self):
        self.text = self.text_entry.get()
        self.value = self.value_entry.get()
        self.units = self.units_entry.get()
        self.destroy()

    def cancel(self):
        self.destroy()


class TransistorDialog(ctk.CTkToplevel):
    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.geometry("400x300")
        self.title("Transistor Dialog")

        self.text = ""
        self.output1 = ""
        self.output2 = ""
        self.output3 = ""
        self.output4 = ""

        self.text_entry = None
        self.entry1 = None
        self.entry2 = None
        self.entry3 = None
        self.entry4 = None

        self.initDialog()

    def initDialog(self):
        text_frame = ctk.CTkFrame(self)
        text_frame.pack(fill=ctk.X)

        text_lbl = ctk.CTkLabel(text_frame, text="Text", width=6)
        text_lbl.pack(side=ctk.LEFT, padx=5, pady=10)

        self.text_entry = ctk.CTkEntry(text_frame, placeholder_text="CTkEntry")
        self.text_entry.pack(fill=ctk.X, padx=5, expand=True)

        frame1 = ctk.CTkFrame(self)
        frame1.pack(fill=ctk.X)

        lbl1 = ctk.CTkLabel(frame1, text="bf", width=6)
        lbl1.pack(side=ctk.LEFT, padx=5, pady=10)

        self.entry1 = ctk.CTkEntry(frame1, placeholder_text="CTkEntry")
        self.entry1.pack(fill=ctk.X, padx=5, expand=True)

        frame2 = ctk.CTkFrame(self)
        frame2.pack(fill=ctk.X)

        lbl2 = ctk.CTkLabel(frame2, text="cjc", width=6)
        lbl2.pack(side=ctk.LEFT, padx=5, pady=10)

        self.entry2 = ctk.CTkEntry(frame2, placeholder_text="CTkEntry")
        self.entry2.pack(fill=ctk.X, padx=5, expand=True)

        frame3 = ctk.CTkFrame(self)
        frame3.pack(fill=ctk.X)

        lbl3 = ctk.CTkLabel(frame3, text="cjc_units", width=6)
        lbl3.pack(side=ctk.LEFT, padx=5, pady=10)

        self.entry3 = ctk.CTkEntry(frame3, placeholder_text="CTkEntry")
        self.entry3.pack(fill=ctk.X, padx=5, expand=True)

        frame4 = ctk.CTkFrame(self)
        frame4.pack(fill=ctk.X)

        lbl4 = ctk.CTkLabel(frame4, text="rb", width=6)
        lbl4.pack(side=ctk.LEFT, padx=5, pady=10)

        self.entry4 = ctk.CTkEntry(frame4, placeholder_text="CTkEntry")
        self.entry4.pack(fill=ctk.X, padx=5, expand=True)

        frame9 = ctk.CTkFrame(self)
        frame9.pack(fill=ctk.X)

        btn = ctk.CTkButton(frame9, text="Cancel", command=self.cancel,
width=60)
        btn.pack(side=ctk.RIGHT, padx=5, pady=5)

        # Command tells the form what to do when the button is clicked
        btn = ctk.CTkButton(frame9, text="OK", command=self.onSubmit, width=60)
        btn.pack(side=ctk.RIGHT, padx=5, pady=5)
```

```
    def onSubmit(self):
        self.text = self.text_entry.get()
        self.output1 = self.entry1.get()
        self.output2 = self.entry2.get()
        self.output3 = self.entry3.get()
        self.output4 = self.entry4.get()
        self.destroy()

    def cancel(self):
        self.destroy()


class Canvas(ctk.CTkCanvas):
    def __init__(self, parent):
        super().__init__(parent)
        self.led_color = "red"
        self.led_size = "large"
        self.grid = Grid(self, 10)
        self.wire_dir = "H"
        self.mouse = Mouse(self)
        self.comp_list = []
        self.wire_list = []
        self.wire_dict = {}

        self.mouse.move_mouse_bind_events()

    def redraw(self):
        self.delete('grid_line')
        self.grid.draw()
        self.tag_lower("grid_line")
        for c in self.comp_list:
            c.update()

    def show_connectors(self):
        for s in self.comp_list:
            s.is_drawing = True
        self.redraw()

    def hide_connectors(self):
        for s in self.comp_list:
            s.is_drawing = False
        self.redraw()

    def edit_shape(self, _event=None):
        if self.mouse.selected_comp:
            comp = self.mouse.selected_comp
            if isinstance(comp, Node):
                dialog = ctk.CTkInputDialog(text="Enter new text", title="Edit
Node Text")
                comp.text = dialog.get_input()
                self.redraw()
            elif (comp.comp_type == 'resistor' or comp.comp_type == 'inductor'
or comp.comp_type == 'capacitor' or
                  comp.comp_type == 'dc_source' or comp.comp_type ==
'ac_source' or
                  comp.comp_type == 'isource'):
                dialog = CompDialog(self)
                dialog.wm_attributes('-topmost', True)
                dialog.wait_window()
                comp.text = dialog.text
                comp.value = dialog.value
                comp.units = dialog.units
                comp.comp_text = comp.text + "=" + str(comp.value) + comp.units
                self.redraw()
            elif comp.comp_type == 'npn_transistor':
                dialog = TransistorDialog(self)
                dialog.wm_attributes('-topmost', True)
                dialog.wait_window()
                comp.text = dialog.text
                comp.bf = dialog.output1
                comp.cjc = dialog.output2
                comp.cjc_units = dialog.output3
                comp.rb = dialog.output4
                self.redraw()

    def set_horiz_dir(self, _event):
        self.wire_dir = "H"

    def set_vert_dir(self, _event):
        self.wire_dir = "V"
```

UI_Lib/file_menu_frame.py

```
import customtkinter as ctk
from tkinter import filedialog as fd
from pathlib import Path
import json
from PIL import Image
```

```python
from Comp_Lib import Connection, AnalogComponent
from Wire_Lib import AnalogWire, Node


class Encoder(json.JSONEncoder):
    def default(self, o):
        if hasattr(o, "reprJson"):
            return o.reprJson()
        else:
            return super().default(o)


class Decoder(json.JSONDecoder):
    def __init__(self):
        json.JSONDecoder.__init__(self, object_hook=Decoder.from_dict)

    @staticmethod
    def from_dict(_d):
        return _d


class FileMenuFrame(ctk.CTkFrame):
    def __init__(self, window, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas
        self.obj_type_dict = {'resistor': AnalogComponent,
                              'inductor': AnalogComponent,
                              'capacitor': AnalogComponent,
                              'ground': AnalogComponent,
                              'dc_source': AnalogComponent,
                              'isource': AnalogComponent,
                              'ac_source': AnalogComponent,
                              'npn_transistor': AnalogComponent,
                              'straight': AnalogWire,
                              'segment': AnalogWire,
                              'elbow': AnalogWire}

        self.menu_on = False

        self.menu_frame = ctk.CTkFrame(window, height=100, bg_color="white")

        new_btn = ctk.CTkButton(self.menu_frame, text="New", width=150,
command=self.new_diagram)
        new_btn.pack(pady=5)
        open_btn = ctk.CTkButton(self.menu_frame, text="Open", width=150,
command=self.load_diagram)
        open_btn.pack(pady=5)

        save_btn = ctk.CTkButton(self.menu_frame, text="Save", width=150,
command=self.save_diagram)
        save_btn.pack(pady=5)

        exit_btn = ctk.CTkButton(self.menu_frame, text="Exit", width=150,
command=window.destroy)
        exit_btn.pack(pady=5)

        my_image = ctk.CTkImage(light_image=Image.open
                                (Path(__file__).parent /
"../icons/hamburger_menu.png"),
                                dark_image=Image.open
                                (Path(__file__).parent /
"../icons/hamburger_menu.png"),
                                size=(24, 24))

        button = ctk.CTkButton(self, text="", image=my_image, width=30,
command=self.show_menu)
        button.pack(side=ctk.LEFT, padx=5, pady=10)

    def new_diagram(self):
        self.canvas.delete("all")
        self.canvas.comp_list.clear()
        self.canvas.wire_list.clear()

    def load_diagram(self):
        try:
            filetypes = (('json files', '*.json'), ('All files', '*.*'))
            f = fd.askopenfilename(filetypes=filetypes, initialdir="./")
            with open(f) as file:
                d = json.load(file)
                self.convert_json_data(d)
        except FileNotFoundError:
            with open('untitled.canvas', 'w') as _file:
                pass

    def convert_json_data(self, data):
        """Convert json data to a circuit object"""
        # Get circuit list from json data
        json_comp_list = data[0]['comp_list']
        for json_comp in json_comp_list:
            if (json_comp['type'] == 'resistor' or json_comp['type'] ==
'inductor' or json_comp['type'] == 'capacitor'
                    or json_comp['type'] == 'dc_source' or json_comp['type'] ==
'isource'
                    or json_comp['type'] == 'ac_source' or json_comp['type'] ==
'ground'):
                comp = AnalogComponent(self.canvas, json_comp['type'],
int(json_comp['x1']), int(json_comp['y1']),
                                      float(json_comp['value']))
                comp.text = json_comp['text']
                comp.units = json_comp['units']
                if comp.value:
                    comp.comp_text = comp.text + "=" + str(comp.value) +
comp.units
                comp.angle = json_comp['angle']
                conn_dict = json_comp['wire_list'][0]
                comp.wire_list.append(Connection(conn_dict['comp_conn'],
conn_dict['wire_name'],
                                                conn_dict['wire_end']))
                if not json_comp['type'] == 'ground':
                    conn_dict = json_comp['wire_list'][1]
                    comp.wire_list.append(Connection(conn_dict['comp_conn'],
conn_dict['wire_name'],
                                                    conn_dict['wire_end']))
                self.canvas.comp_list.append(comp)
            elif json_comp['type'] == 'npn_transistor':
                comp = AnalogComponent(self.canvas, json_comp['type'],
int(json_comp['x1']), int(json_comp['y1']), 0)
                comp.text = json_comp['text']
                comp.bf = json_comp['bf']
                comp.cjc = json_comp['cjc']
                comp.cjc_units = json_comp['cjc_units']
                comp.rb = json_comp['rb']
                comp.angle = json_comp['angle']
                conn_dict = json_comp['wire_list'][0]
                comp.wire_list.append(Connection(conn_dict['comp_conn'],
conn_dict['wire_name'],
                                                conn_dict['wire_end']))
                conn_dict = json_comp['wire_list'][1]
                comp.wire_list.append(Connection(conn_dict['comp_conn'],
conn_dict['wire_name'],
                                                conn_dict['wire_end']))
                conn_dict = json_comp['wire_list'][2]
                comp.wire_list.append(Connection(conn_dict['comp_conn'],
conn_dict['wire_name'],
                                                 conn_dict['wire_end']))
                self.canvas.comp_list.append(comp)
            elif json_comp['type'] == 'straight' or json_comp['type'] ==
'segment' or json_comp['type'] == 'elbow':
                self.canvas.wire_dir = json_comp['wire_dir']
                wire = AnalogWire(self.canvas, json_comp['type'],
int(json_comp['x1']), int(json_comp['y1']),
                                  int(json_comp['x2']), int(json_comp['y2']))
                wire.name = json_comp['name']
                wire.node_num = json_comp['node_num']
                self.canvas.comp_list.append(wire)
                self.canvas.wire_dict[wire.name] = wire
            elif json_comp['type'] == 'node':
                node = Node(self.canvas, int(json_comp['x1']),
int(json_comp['y1']))
                node.text = json_comp['text']
                for connection in json_comp['wire_list']:
                    node.wire_list.append(Connection(connection['comp_conn'],
connection['wire_name'],
                                                     connection['wire_end']))
                self.canvas.comp_list.append(node)

        # for comp in self.canvas.comp_list:
        #     print(comp)
        self.canvas.redraw()

    def save_diagram(self):
        comp_dict = {'comp_list': self.canvas.comp_list}
        circuit = [comp_dict]

        filetypes = (('json files', '*.json'), ('All files', '*.*'))
        f = fd.asksaveasfilename(filetypes=filetypes, initialdir="./")
        with open(f, 'w') as file:
            file.write(json.dumps(circuit, cls=Encoder, indent=4))

    def show_menu(self):
        if not self.menu_on:
            self.menu_frame.place(x=5, y=60)
            self.menu_frame.tkraise()
            self.menu_on = True
        else:
            self.menu_frame.place_forget()
            self.menu_on = False
```

UI_Lib/grid.py

```
class Grid:
    def __init__(self, canvas, grid_size):
        self.canvas = canvas
        self.grid_size = grid_size
        self.grid_visible = True

        self.grid_snap = 5
        self.draw()

    def draw(self):
        if self.grid_visible:
            w = self.canvas.winfo_width()  # Get current width of canvas
            h = self.canvas.winfo_height()  # Get current height of canvas

            # Creates all vertical lines at intervals of 100
            for i in range(0, w, self.grid_size):
                self.canvas.create_line([(i, 0), (i, h)],  fill='#cccccc',
tags='grid_line')

            # Creates all horizontal lines at intervals of 100
            for i in range(0, h, self.grid_size):
                self.canvas.create_line([(0, i), (w, i)],  fill='#cccccc',
tags='grid_line')

    def snap_to_grid(self, x, y):
        if self.grid_visible:
            x = round(x / self.grid_snap) * self.grid_snap
            y = round(y / self.grid_snap) * self.grid_snap
        return x, y
```

UI_Lib/help_frame.py

```
import customtkinter as ctk
from tkinter import messagebox
from PIL import Image


class HelpFrame(ctk.CTkFrame):
    def __init__(self, window, parent, canvas):
        super().__init__(parent)
        self.window = window
        self.parent = parent
        self.canvas = canvas
        self.menu_on = False

        self.menu_frame = ctk.CTkFrame(window, height=100, bg_color="white")

        about_image = ctk.CTkImage(light_image=Image.open

("D:/EETools/SimpleDiagramEditor/icons/about.png"),
                                dark_image=Image.open

("D:/EETools/SimpleDiagramEditor/icons/about.png"),
                                size=(24, 24))

        about_button = ctk.CTkButton(self.menu_frame, text="About",
image=about_image, width=30,
                                     command=self.show_about_dialog)
        about_button.pack(side=ctk.TOP,padx=5, pady=5)

        my_image = ctk.CTkImage(light_image=Image.open

("D:/EETools/SimpleDiagramEditor/icons/help.png"),
                                dark_image=Image.open

("D:/EETools/SimpleDiagramEditor/icons/help.png"),
                                size=(24, 24))

        button = ctk.CTkButton(self, text="", image=my_image, width=30,
command=self.show_menu)
        button.pack(side=ctk.LEFT, padx=5, pady=10)

    def show_menu(self):
        if not self.menu_on:
            menu_pos_x = self.canvas.winfo_width()
            self.menu_frame.place(x=menu_pos_x + 50, y=60)
            self.menu_frame.tkraise()
            self.menu_on = True
        else:
            self.menu_frame.place_forget()
            self.menu_on = False

    @staticmethod
    def show_about_dialog():
        messagebox.showinfo("About Digital Simulator", "RF/Microwave Simulator
v0.1\n" +
                            "Author: Rick A. Crist\n" + "2023")
```

UI_Lib/left_frame.py

```
import customtkinter as ctk
from UI_Lib.lump_button_frame import LumpButtonFrame
from UI_Lib.sources_button_frame import SourcesButtonFrame
from UI_Lib.active_button_frame import ActiveButtonFrame
from UI_Lib.wire_button_frame import WireButtonFrame


class LeftFrame(ctk.CTkFrame):
    def __init__(self, parent, canvas):
        super().__init__(parent)
        self.canvas = canvas

        self.comp_button_frame = LumpButtonFrame(self, self.canvas)
        self.comp_button_frame.pack(side=ctk.TOP, padx=5, pady=5)

        self.sources_button_frame = SourcesButtonFrame(self, self.canvas)
        self.sources_button_frame.pack(side=ctk.TOP, padx=5, pady=5)

        self.active_button_frame = ActiveButtonFrame(self, self.canvas)
        self.active_button_frame.pack(side=ctk.TOP, padx=5, pady=5)

        self.wire_button_frame = WireButtonFrame(self, self.canvas)
        self.wire_button_frame.pack(side=ctk.TOP, padx=5, pady=5)
```

UI_Lib/lump_button_frame.py

```
import customtkinter as ctk
from tktooltip import ToolTip
from pathlib import Path
from PIL import Image

from Comp_Lib import AnalogComponent


class LumpButtonFrame(ctk.CTkFrame):
    def __init__(self, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas
        self.button_id = None
        self.button_list = [("resistor", "../icons/resistor.png"),
                            ("inductor", "../icons/inductor.png"),
                            ("capacitor", "../icons/capacitor.png")]

        self.init_frame_widgets()

    def init_frame_widgets(self):
        frame_name_label = ctk.CTkLabel(self, text="Lumped", font=("Helvetica",
10), height=20)
        frame_name_label.grid(row=0, column=0, columnspan=3, sticky=ctk.W,
padx=2, pady=2)

        row_num, col_num = 1, 0
        for button in self.button_list:
            a_image = ctk.CTkImage(light_image=Image.open
                                    (Path(__file__).parent / button[1]),
                                    dark_image=Image.open
                                    (Path(__file__).parent / button[1]),
                                    size=(24, 24))
            self.button_id = ctk.CTkButton(self, text="", image=a_image,
width=30,
                                           command=lambda
a_name=button[0]:self.create_events(a_name))
            self.button_id.grid(row=row_num, column=col_num, sticky=ctk.W,
padx=2, pady=2)
            ToolTip(self.button_id, msg=button[0])
            row_num, col_num = self.update_grid_numbers(row_num, col_num)

    def create_events(self, name):
        comp = None
        if name == "resistor":
            comp = AnalogComponent(self.canvas, 'resistor', 100, 100, 75)
        elif name == "inductor":
            comp = AnalogComponent(self.canvas, 'inductor', 100, 100, 5)
        elif name == "capacitor":
            comp = AnalogComponent(self.canvas, 'capacitor',100, 100, 1)
        self.canvas.comp_list.append(comp)
        self.canvas.redraw()
        self.canvas.mouse.move_mouse_bind_events()

    @staticmethod
    def update_grid_numbers(row, column):
        column += 1
        if column > 2:
            column = 0
            row += 1
        return row, column
```

UI_Lib/mouse.py

```
from Helper_Lib import Point
from Wire_Lib import AnalogWire, Node
from Comp_Lib import Connection


class Mouse:
    def __init__(self, canvas):
        self.canvas = canvas

        self.selected_comp = None
        self.current_wire_obj = None
        self.start = Point(0, 0)
        self.offset1 = Point(0, 0)
        self.offset2 = Point(0, 0)

        self.move_mouse_bind_events()

    def move_mouse_bind_events(self):
        self.canvas.bind("<Button-1>", self.move_left_down)
        self.canvas.bind("<B1-Motion>", self.move_left_drag)
        self.canvas.bind("<ButtonRelease-1>", self.move_left_up)

    def draw_wire_mouse_events(self):  # Added method to bind draw wire methods
        self.canvas.bind("<Button-1>", self.draw_left_down)
        self.canvas.bind("<B1-Motion>", self.draw_left_drag)
        self.canvas.bind("<ButtonRelease-1>", self.draw_left_up)

    def resize_wire_mouse_events(self):
        self.canvas.bind("<Button-1>", self.resize_left_down)
        self.canvas.bind("<B1-Motion>", self.resize_left_drag)
        self.canvas.bind("<ButtonRelease-1>", self.resize_left_up)

    def move_left_down(self, event):
        x, y = event.x, event.y
        self.comp_hit_test(x, y)
        if self.selected_comp:
            if isinstance(self.selected_comp, AnalogWire):
                x1, y1 = self.selected_comp.x1, self.selected_comp.y1
                x2, y2 = self.selected_comp.x2, self.selected_comp.y2
                self.offset1.set(x - x1, y - y1)
                self.offset2.set(x - x2, y - y2)
            else:
                x1, y1 = self.selected_comp.x1, self.selected_comp.y1
                self.offset1.x, self.offset1.y =
self.canvas.grid.snap_to_grid(self.offset1.x, self.offset1.y)
                self.offset1.set(x - x1, y - y1)

    def move_left_drag(self, event):
        if self.selected_comp:
            if isinstance(self.selected_comp, AnalogWire):
                x1 = event.x - self.offset1.x
                y1 = event.y - self.offset1.y
                x1, y1 = self.canvas.grid.snap_to_grid(x1, y1)
                x2 = event.x - self.offset2.x
                y2 = event.y - self.offset2.y
                x2, y2 = self.canvas.grid.snap_to_grid(x2, y2)
                self.selected_comp.x1, self.selected_comp.y1 = x1, y1
                self.selected_comp.x2, self.selected_comp.y2 = x2, y2
                self.canvas.redraw()
            else:
                x = event.x - self.offset1.x
                y = event.y - self.offset1.y
                x, y = self.canvas.grid.snap_to_grid(x, y)
                self.selected_comp.x1, self.selected_comp.y1 = x, y
                self.canvas.redraw()

    def move_left_up(self, _event):
        self.offset1.set(0, 0)
        self.offset2.set(0, 0)
        self.canvas.redraw()

    def draw_left_down(self, event):  # Added method for draw left down
        if self.current_wire_obj:
            # self.unselect_all()
            self.start.x = event.x
            self.start.y = event.y
            self.start.x, self.start.y =
self.canvas.grid.snap_to_grid(self.start.x, self.start.y)

            self.current_wire_obj.x1, self.current_wire_obj.y1 = self.start.x,
self.start.y
            self.current_wire_obj.x2, self.current_wire_obj.y2 = self.start.x,
self.start.y
```

```
            self.select_connector(self.current_wire_obj, "begin", self.start.x,
self.start.y)

    def draw_left_drag(self, event):  # Added method for draw left drag
        if self.current_wire_obj:
            wire = self.current_wire_obj
            x, y = event.x, event.y
            x, y = self.canvas.grid.snap_to_grid(x, y)
            wire.x1, wire.y1 = self.start.x, self.start.y
            wire.x2, wire.y2 = x, y
            self.current_wire_obj.update()

    def draw_left_up(self, event):  # Added method for draw left up
        self.select_connector(self.current_wire_obj, "end", event.x, event.y)
        self.canvas.hide_connectors()
        self.current_wire_obj = None
        self.move_mouse_bind_events()

    def resize_left_down(self, event):
        if self.selected_comp:
            x1, y1 = self.selected_comp.x1, self.selected_comp.y1
            x1, y1 = self.canvas.grid.snap_to_grid(x1, y1, "resize")
            x2, y2 = self.selected_comp.x2, self.selected_comp.y2
            x2, y2 = self.canvas.grid.snap_to_grid(x2, y2, "resize")
            self.offset1.x = event.x - x1
            self.offset1.y = event.y - y1
            self.offset2.x = event.x - x2
            self.offset2.y = event.y - y2
            self.selected_comp.update()

    def resize_left_drag(self, event):
        if self.selected_comp:
            offsets = [self.offset1.x, self.offset1.y, self.offset2.x,
self.offset2.y]
            self.selected_comp.resize(offsets, event)
            self.selected_comp.update()

    def resize_left_up(self, _event):
        self.offset1.x, self.offset1.y = 0, 0
        self.offset2.x, self.offset2.y = 0, 0
        self.move_mouse_bind_events()

    def comp_hit_test(self, x, y):
        for comp in self.canvas.comp_list:
            comp.hit_test(x, y)
            if comp.is_selected:
                if isinstance(self.selected_comp, AnalogWire):
                    result = comp.sel_hit_test(x, y)
                    if result is not None:
                        self.resize_wire_mouse_events()
                comp.update()
                self.selected_comp = comp
                return

        # No shape hit - unselect all
        self.selected_comp = None
        self.unselect_all()

    def unselect_all(self):
        for comp in self.canvas.comp_list:
            comp.is_selected = False
            comp.update()

    def select_connector(self, wire_obj, wire_end, x, y):
        for comp in self.canvas.comp_list:
            if not isinstance(comp, AnalogWire):
                conn = comp.check_connector_hit(x, y)
                if conn:
                    if wire_end == "begin":
                        wire_obj.x1, wire_obj.y1 = conn.x, conn.y
                    elif wire_end == "end":
                        wire_obj.x2, wire_obj.y2 = conn.x, conn.y
                    a_conn = Connection(conn.name, self.current_wire_obj.name,
wire_end)
                    # wire_obj.create_wire_list_cnx(comp.comp_type, conn.name)
                    if isinstance(comp, Node):
                        wire_obj.set_node_num(comp.text)
                    comp.wire_list.append(a_conn)
                    self.canvas.redraw()
```

UI_Lib/settings_frame.py

```
import customtkinter as ctk
from PIL import Image


class SettingsFrame(ctk.CTkFrame):
    def __init__(self, window, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas

        self.menu_on = False

        self.menu_frame = ctk.CTkFrame(window, height=100, bg_color="white")

        def grid_switch_event():
            if canvas.grid.grid_visible:
                self.canvas.grid.grid_visible = False
            else:
                self.canvas.grid.grid_visible = True
            self.canvas.redraw()

        switch_var = ctk.StringVar(value="on")
        switch = ctk.CTkSwitch(self.menu_frame, text="Grid",
command=grid_switch_event,
                                         variable=switch_var, onvalue="on",
offvalue="off")
        switch.pack(padx=5, pady=5)

        grid_size_label = ctk.CTkLabel(self.menu_frame, text="Grid Size", font=
("Helvetica", 10), height=20)
        grid_size_label.pack(padx=5, pady=5, anchor="w")

        def optionmenu_callback(choice):
            self.canvas.grid.grid_size = int(choice)
            self.canvas.redraw()

        optionmenu = ctk.CTkOptionMenu(self.menu_frame, values=["5", "10",
"20", "30", "40", "50"],
                                                 command=optionmenu_callback)
        optionmenu.pack(padx=5, pady=5)
        optionmenu.set("10")

        grid_snap_label = ctk.CTkLabel(self.menu_frame, text="Snap Size", font=
("Helvetica", 10), height=20)
        grid_snap_label.pack(padx=5, pady=5, anchor="w")

        def snap_option_callback(choice):
            if choice == "Grid Size":
                self.canvas.grid.grid_snap = canvas.grid.grid_size
            else:
                self.canvas.grid.grid_snap = int(choice)
            canvas.redraw()
        snap_option = ctk.CTkOptionMenu(self.menu_frame, values=["Grid Size",
"5", "10", "20", "30", "40", "50"],
                                                 command=snap_option_callback)
        snap_option.pack(padx=5, pady=5)
        snap_option.set("Grid Size")

        self.appearance_mode_label = ctk.CTkLabel(self.menu_frame,
text="Appearance Mode:", anchor="w")
        self.appearance_mode_label.pack(padx=5, pady=5)
        self.appearance_mode_optionemenu = ctk.CTkOptionMenu(self.menu_frame,
                                                                       values=
["Light", "Dark", "System"],

command=self.change_appearance_mode_event)
        self.appearance_mode_optionemenu.pack(padx=5, pady=5)
        self.appearance_mode_optionemenu.set("Dark")

        my_image = ctk.CTkImage(light_image=Image.open

("D:/EETools/DiagramEditor/icons/settings.png"),
                                dark_image=Image.open

("D:/EETools/DiagramEditor/icons/settings.png"),
                                size=(24, 24))

        button = ctk.CTkButton(self, text="", image=my_image, width=30,
command=self.show_menu)
        button.pack(side=ctk.LEFT, padx=5, pady=10)

    def show_menu(self):
        if not self.menu_on:
            self.menu_frame.place(x=15, y=60)
            self.menu_frame.tkraise()
            self.menu_on = True
        else:
            self.menu_frame.place_forget()
            self.menu_on = False

    @staticmethod
    def change_appearance_mode_event(new_appearance_mode: str):
        ctk.set_appearance_mode(new_appearance_mode)
```

UI_Lib/sources_button_frame.py

```
import customtkinter as ctk
from PIL import Image


class SettingsFrame(ctk.CTkFrame):
    def __init__(self, window, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas

        self.menu_on = False

        self.menu_frame = ctk.CTkFrame(window, height=100, bg_color="white")

        def grid_switch_event():
            if canvas.grid.grid_visible:
                self.canvas.grid.grid_visible = False
            else:
                self.canvas.grid.grid_visible = True
            self.canvas.redraw()

        switch_var = ctk.StringVar(value="on")
        switch = ctk.CTkSwitch(self.menu_frame, text="Grid",
command=grid_switch_event,
                                         variable=switch_var, onvalue="on",
offvalue="off")
        switch.pack(padx=5, pady=5)

        grid_size_label = ctk.CTkLabel(self.menu_frame, text="Grid Size", font=
("Helvetica", 10), height=20)
        grid_size_label.pack(padx=5, pady=5, anchor="w")

        def optionmenu_callback(choice):
            self.canvas.grid.grid_size = int(choice)
            self.canvas.redraw()

        optionmenu = ctk.CTkOptionMenu(self.menu_frame, values=["5", "10",
"20", "30", "40", "50"],
                                                 command=optionmenu_callback)
        optionmenu.pack(padx=5, pady=5)
        optionmenu.set("10")

        grid_snap_label = ctk.CTkLabel(self.menu_frame, text="Snap Size", font=
("Helvetica", 10), height=20)
        grid_snap_label.pack(padx=5, pady=5, anchor="w")
        def snap_option_callback(choice):
            if choice == "Grid Size":
                self.canvas.grid.grid_snap = canvas.grid.grid_size
            else:
                self.canvas.grid.grid_snap = int(choice)
            canvas.redraw()

        snap_option = ctk.CTkOptionMenu(self.menu_frame, values=["Grid Size",
"5", "10", "20", "30", "40", "50"],
                                         command=snap_option_callback)
        snap_option.pack(padx=5, pady=5)
        snap_option.set("Grid Size")

        self.appearance_mode_label = ctk.CTkLabel(self.menu_frame,
text="Appearance Mode:", anchor="w")
        self.appearance_mode_label.pack(padx=5, pady=5)
        self.appearance_mode_optionemenu = ctk.CTkOptionMenu(self.menu_frame,
                                                                       values=
["Light", "Dark", "System"],

command=self.change_appearance_mode_event)
        self.appearance_mode_optionemenu.pack(padx=5, pady=5)
        self.appearance_mode_optionemenu.set("Dark")

        my_image = ctk.CTkImage(light_image=Image.open

("D:/EETools/DiagramEditor/icons/settings.png"),
                                dark_image=Image.open

("D:/EETools/DiagramEditor/icons/settings.png"),
                                size=(24, 24))

        button = ctk.CTkButton(self, text="", image=my_image, width=30,
command=self.show_menu)
        button.pack(side=ctk.LEFT, padx=5, pady=10)

    def show_menu(self):
        if not self.menu_on:
            self.menu_frame.place(x=15, y=60)
            self.menu_frame.tkraise()
            self.menu_on = True
        else:
            self.menu_frame.place_forget()
            self.menu_on = False

    @staticmethod
    def change_appearance_mode_event(new_appearance_mode: str):
        ctk.set_appearance_mode(new_appearance_mode)
```

UI_Lib/top_frame.py

```
import customtkinter as ctk
from tktooltip import ToolTip
from pathlib import Path
from PIL import Image

import matplotlib.pyplot as plt
import PySpice.Logging.Logging as Logging
logger = Logging.setup_logging()
from PySpice.Doc.ExampleTools import find_libraries
from PySpice.Spice.Library import SpiceLibrary
from PySpice.Spice.Netlist import Circuit

from UI_Lib.file_menu_frame import FileMenuFrame
from UI_Lib.settings_frame import SettingsFrame
from UI_Lib.help_frame import HelpFrame
from Wire_Lib import AnalogWire


class TopFrame(ctk.CTkFrame):
    def __init__(self, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas

        # Add Top Frame widget here
        file_frame = FileMenuFrame(self.parent, self, self.canvas)
        file_frame.pack(side=ctk.LEFT, padx=5, pady=5)

        settings_frame = SettingsFrame(self.parent, self, self.canvas)
        settings_frame.pack(side=ctk.LEFT, padx=5, pady=5)

        help_frame = HelpFrame(self.parent, self, self.canvas)
        help_frame.pack(side=ctk.RIGHT, padx=5, pady=5)

        a_image = ctk.CTkImage(light_image=Image.open(Path(__file__).parent /
"../icons/angle.png"),
                               dark_image=Image.open(Path(__file__).parent /
"../icons/angle.png"),
                               size=(24, 24))
        self.button_id = ctk.CTkButton(self, text="", image=a_image, width=30,
command=self.rotate_comp)
        self.button_id.pack(side=ctk.LEFT, padx=5, pady=5)
        ToolTip(self.button_id, msg="Rotate selected component")

        a_image = ctk.CTkImage(light_image=Image.open(Path(__file__).parent /
"../icons/analyze.png"),
                               dark_image=Image.open(Path(__file__).parent /
"../icons/analyze.png"),
                               size=(24, 24))
        self.button_id = ctk.CTkButton(self, text="", image=a_image, width=30,
command=self.analyze_circuit)
        self.button_id.pack(side=ctk.LEFT, padx=5, pady=5)
        ToolTip(self.button_id, msg="Analyze circuit")

    def rotate_comp(self):
        self.parent.rotate_comp(_event=None)

    def analyze_circuit(self):
        libraries_path = find_libraries()
        _spice_library = SpiceLibrary(libraries_path)

        units = {
            'u_V': 1,
            'u_kΩ': 1000,
            'u_kHz': 1000,
            'u_uF': 1e-6,
            'u_MΩ': 1e6,
            'u_pF': 1e-12
        }

        freq_amplitude = 1
        freq_units = units['u_kHz']

        source = None

        circuit = Circuit('Transistor')

        for comp in self.canvas.comp_list:
            if not isinstance(comp, AnalogWire):
                if comp.comp_type == 'dc_source':
                    node_list = self.get_nodes(comp)
                    circuit.V('power', node_list[0], circuit.gnd, comp.value *
units[comp.units])
                elif comp.comp_type == 'ac_source':
                    node_list = self.get_nodes(comp)
                    source = circuit.SinusoidalVoltageSource('in',
node_list[0], circuit.gnd,
                             amplitude=comp.value*units[comp.units],
frequency=freq_amplitude*freq_units)
                elif comp.comp_type == 'resistor':
                    comp_num = self.get_comp_num(comp.text)
                    node_list = self.get_nodes(comp)
                    circuit.R(comp_num, node_list[0], node_list[1],
comp.value*units[comp.units])
                elif comp.comp_type == 'capacitor':
                    comp_num = self.get_comp_num(comp.text)
                    node_list = self.get_nodes(comp)
                    circuit.C(comp_num, node_list[0], node_list[1], comp.value
* units[comp.units])
                elif comp.comp_type == 'inductor':
                    comp_num = self.get_comp_num(comp.text)
                    node_list = self.get_nodes(comp)
                    circuit.L(comp_num, node_list[0], node_list[1], comp.value
* units[comp.units])
                elif comp.comp_type == 'npn_transistor':
                    comp_num = self.get_comp_num(comp.text)
                    node_list = self.get_nodes(comp)
                    circuit.BJT(comp_num, node_list[1], node_list[0],
node_list[2], model='bjt')
                    circuit.model('bjt', 'npn', bf=comp.bf,
cjc=comp.cjc*units[comp.cjc_units], rb=comp.rb)

        print(circuit)
        # Create circuit plots
        figure, ax = plt.subplots(figsize=(20, 10))

        # .ac dec 5 10m 1G

        simulator = circuit.simulator(temperature=25, nominal_temperature=25)
        analysis = simulator.transient(step_time=source.period / 200,
end_time=source.period * 2)

        ax.set_title('')
        ax.set_xlabel('Time [s]')
        ax.set_ylabel('Voltage [V]')
        ax.grid()
        ax.plot(analysis['in'])
        ax.plot(analysis.out)
        ax.legend(('input', 'output'), loc=(.05, .1))
        plt.tight_layout()
        plt.show()

    @staticmethod
    def get_comp_num(comp_text):
        comp_num = comp_text[1:]
        return comp_num

    def get_nodes(self, in_comp):
        node_list = []
        for conn in in_comp.wire_list:
            for comp in self.canvas.comp_list:
                if isinstance(comp, AnalogWire) and comp.name ==
conn.wire_name:
                    node_list.append(comp.node_num)
        return node_list
```

UI_Lib/wire_button_frame.py

```
import customtkinter as ctk
from tktooltip import ToolTip
from pathlib import Path
from PIL import Image

from Wire_Lib import AnalogWire, Node


class WireButtonFrame(ctk.CTkFrame):
    def __init__(self, parent, canvas):
        super().__init__(parent)
        self.parent = parent
        self.canvas = canvas
        self.wire = None
        self.wire_count = 0

        # Add frame widgets here
        frame_name_label = ctk.CTkLabel(self, text="Wires", font=("Helvetica",
10), height=20)
        frame_name_label.grid(row=0, column=0, columnspan=3, sticky=ctk.W,
padx=2, pady=2)

        straight_wire_image = ctk.CTkImage(light_image=Image.open
                                (Path(__file__).parent /
"../icons/straight_line.png"),
                                dark_image=Image.open
                                (Path(__file__).parent /
"../icons/straight_line.png"),
                                size=(24, 24))

        straight_wire_button = ctk.CTkButton(self, text="",
image=straight_wire_image, width=30,
                                        command=self.create_straight_wire)
        straight_wire_button.grid(row=1, column=0, sticky=ctk.W, padx=2,
pady=2)
        ToolTip(straight_wire_button, msg='Straight wire')

        segment_wire_image = ctk.CTkImage(light_image=Image.open
                                (Path(__file__).parent /
"../icons/segment_line.png"),
                                dark_image=Image.open
                                (Path(__file__).parent /
"../icons/segment_line.png"),
                                size=(24, 24))

        segment_wire_button = ctk.CTkButton(self, text="",
image=segment_wire_image, width=30,
                                        command=self.create_segment_wire)
        segment_wire_button.grid(row=1, column=1, sticky=ctk.W, padx=2, pady=2)
        ToolTip(segment_wire_button, msg='Segment wire')

        elbow_wire_image = ctk.CTkImage(light_image=Image.open
                                (Path(__file__).parent /
"../icons/elbow_line.png"),
                                dark_image=Image.open
                                (Path(__file__).parent /
"../icons/elbow_line.png"),
                                size=(24, 24))

        elbow_wire_button = ctk.CTkButton(self, text="",
image=elbow_wire_image, width=30,
                                        command=self.create_elbow_wire)
        elbow_wire_button.grid(row=1, column=2, sticky=ctk.W, padx=2, pady=2)
        ToolTip(elbow_wire_button, msg='Elbow wire')

        node_image = ctk.CTkImage(light_image=Image.open
                                (Path(__file__).parent / "../icons/node.png"),
                                dark_image=Image.open
                                (Path(__file__).parent / "../icons/node.png"),
                                size=(24, 24))
        node_button = ctk.CTkButton(self, text="", image=node_image, width=30,
                                        command=self.create_node)
        node_button.grid(row=2, column=0, sticky=ctk.W, padx=2, pady=2)
        ToolTip(node_button, msg='Node')

    # Shape button handlers
    def create_straight_wire(self):
        wire = AnalogWire(self.canvas, 'straight', 0, 0, 0, 0)
        self.create_wire(wire)

    def create_segment_wire(self):
        wire = AnalogWire(self.canvas, 'segment', 0, 0, 0, 0)
        self.create_wire(wire)

    def create_elbow_wire(self):
        wire = AnalogWire(self.canvas, 'elbow', 0, 0, 0, 0)
        self.create_wire(wire)

    def create_node(self):
        node = Node(self.canvas, 100, 100)
        self.canvas.comp_list.append(node)
        self.canvas.redraw()
        self.canvas.mouse.move_mouse_bind_events()

    def create_wire(self, wire):
        self.assign_wire_name(wire)
        self.canvas.mouse.current_wire_obj = wire
        self.canvas.show_connectors()
        self.canvas.comp_list.append(wire)
        self.canvas.mouse.draw_wire_mouse_events()

    def assign_wire_name(self, wire):
        self.wire_count += 1
        wire_name = 'wire' + str(self.wire_count)
        wire.name = wire_name
        self.canvas.wire_dict[wire_name] = wire
```

`Wire_Lib`/`__init__.py`

```
# import wire related classes
from Wire_Lib.analog_wire import AnalogWire
from Wire_Lib.node import Node
```

```
# import wire selector related classes
from Wire_Lib.wire_selector import WireSelector
```

Wire_Lib/wire.py

```
from Wire_Lib.wire_selector import WireSelector


class Wire:
    """Base class for wire classes"""
    def __init__(self, canvas, wire_type, x1, y1, x2, y2):
        self.canvas = canvas
        self.wire_type = wire_type
        self.x1 = x1
        self.y1 = y1
        self.x2 = x2
        self.y2 = y2

        self.fill_color = "black"
        self.border_width = 2
        self.wire_dir = self.canvas.wire_dir

        self.name = None
        self.id = None
        self.is_selected = False
        self.selector = None
        self.width = 2
        self.bbox = None
        self.node_num = '0'

        # Connections for wire list
        self.in_cnx = None
        self.out_cnx = None
        self.cnx = []

        self.sel1, self.sel2 = None, None

    def update_bbox(self):
        self.bbox = self.canvas.bbox(self.id)

    def create_selectors(self):
        self.sel1 = WireSelector(self.canvas, "begin", self.x1, self.y1)
        self.sel2 = WireSelector(self.canvas, "end", self.x2, self.y2)

    def update_selectors(self):
        self.sel1.x, self.sel1.y = self.x1, self.y1
        self.sel2.x, self.sel2.y = self.x2, self.y2
        self.sel1.update()
        self.sel2.update()

    def update_selection(self):
        if self.is_selected:
            self.canvas.itemconfigure(self.id, fill="red")
            self.canvas.itemconfigure(self.sel1.id, state='normal')
            self.canvas.itemconfigure(self.sel2.id, state='normal')
        else:
            self.canvas.itemconfigure(self.id, fill="black")
            self.canvas.itemconfigure(self.sel1.id, state='hidden')
            self.canvas.itemconfigure(self.sel2.id, state='hidden')

    def hit_test(self, x, y):
        x1, y1 = self.bbox[0], self.bbox[1]
        x2, y2 = self.bbox[2], self.bbox[3]
        if x1 <= x <= x2 and y1 <= y <= y2:
            self.is_selected = True
        else:
            self.is_selected = False

    def sel_hit_test(self, x, y):
        if self.sel1.selector_hit_test(x, y):
            self.selector = self.sel1.name
            return self.sel1
        elif self.sel2.selector_hit_test(x, y):
            self.selector = self.sel2.name
            return self.sel2
        else:
            return None

    def resize(self, offsets, event):
        offset_x1, offset_y1, offset_x2, offset_y2 = offsets
        if self.selector == "end":
            x2 = event.x - offset_x2
            y2 = event.y - offset_y2
            self.x2, self.y2 = x2, y2
            self.x2, self.y2 = self.canvas.grid.snap_to_grid(self.x2, self.y2)
        elif self.selector == "begin":
            x1 = event.x - offset_x1
            y1 = event.y - offset_y1
            self.x1, self.y1 = x1, y1
            self.x1, self.y1 = self.canvas.grid.snap_to_grid(self.x1, self.y1)

    def set_node_num(self, node_num):
        self.node_num = node_num
```

Wire_Lib/analog_wire.py

```
from Wire_Lib.wire import Wire


class AnalogWire(Wire):
    def __init__(self, canvas, wire_type, x1, y1, x2, y2):
        super().__init__(canvas, wire_type, x1, y1, x2, y2)

        self.seg1, self.seg2, self.seg3 = None, None, None
        self.segment_list = []

        self.create_wire()
        self.update_bbox()
        self.create_selectors()
        self.update_selection()

    def create_wire(self):
        if self.wire_type == 'straight':
            self.create_straight_wire()
        elif self.wire_type == 'elbow':
            self.create_elbow_wire()
        elif self.wire_type == 'segment':
            self.create_segment_wire()

    def create_straight_wire(self):
        self.id = self.canvas.create_line(self.x1, self.y1, self.x2, self.y2,
width=self.width)

    def create_elbow_wire(self):
        if self.wire_dir == "H":  # Horizontal
            self.id = self.canvas.create_line(self.x1, self.y1, self.x2,
self.y1,
                                              self.x2, self.y1, self.x2,
self.y2,
                                              fill=self.fill_color,
                                              width=self.border_width,
tags="wire")
        elif self.wire_dir == "V":  # Vertical
            self.id = self.canvas.create_line(self.x1, self.y1, self.x1,
self.y2,
                                              self.x1, self.y2, self.x2,
self.y2,
                                              fill=self.fill_color,
                                              width=self.border_width,
tags="wire")

    def create_segment_wire(self):
        w = self.x2 - self.x1
        h = self.y2 - self.y1

        if self.wire_dir == "H":  # Horizontal
            self.seg1 = self.x1, self.y1, self.x1 + w / 2, self.y1
            self.seg2 = self.x1 + w / 2, self.y1, self.x1 + w / 2, self.y2
            self.seg3 = self.x1 + w / 2, self.y2, self.x2, self.y2
        elif self.wire_dir == "V":  # Vertical
            self.seg1 = self.x1, self.y1, self.x1, self.y1 + h / 2
            self.seg2 = self.x1, self.y1 + h / 2, self.x2, self.y1 + h / 2
            self.seg3 = self.x2, self.y1 + h / 2, self.x2, self.y2
        self.segment_list = [self.seg1, self.seg2, self.seg3]
        self.draw_segments()

    def draw_segments(self):
        for s in self.segment_list:
            self.id = self.canvas.create_line(s, fill=self.fill_color,
width=self.border_width, tags='wire')

    def update(self):
        self.update_position()
        self.update_bbox()
        self.update_selectors()
        self.update_selection()

    def update_position(self):
        if self.wire_type == 'straight':
            self.update_straight_position()
        elif self.wire_type == 'elbow':
            self.update_elbow_position()
        elif self.wire_type == 'segment':
            self.update_segment_position()

    def update_straight_position(self):
        """Update the position when the attached component is moved"""
        self.canvas.coords(self.id, self.x1, self.y1, self.x2, self.y2)

    def update_elbow_position(self):
        """Update the position when the attached component is moved"""
        if self.wire_dir == "H":
            self.canvas.coords(self.id, self.x1, self.y1, self.x2, self.y1,
                                        self.x2, self.y1, self.x2, self.y2)
        elif self.wire_dir == "V":
            self.canvas.coords(self.id, self.x1, self.y1, self.x1, self.y2,
                                        self.x1, self.y2, self.x2, self.y2)

    def update_segment_position(self):
        """Update the position when the attached component is moved"""
        w = self.x2 - self.x1
        h = self.y2 - self.y1
        if self.wire_dir == "H":
            self.canvas.coords(self.id, self.x1, self.y1, self.x1 + w / 2,
self.y1,
                               self.x1 + w / 2, self.y1, self.x1 + w / 2,
self.y2,
                               self.x1 + w / 2, self.y2, self.x2, self.y2)
        elif self.wire_dir == "V":
            self.canvas.coords(self.id, self.x1, self.y1, self.x1, self.y1 + h
/ 2,
                               self.x1, self.y1 + h / 2, self.x2, self.y1 + h /
2,
                               self.x2, self.y1 + h / 2, self.x2, self.y2)

    def hit_test(self, x, y):
        # 2-Point Line equation: y = m * (x - x1) + y1
        x1, y1 = self.x1, self.y1
        x2, y2 = self.x2, self.y2

        # Calculate the slope: m = (y2 - y1) / (x2 - x1)
        if (x2 - x1) == 0:
            m = 0
        else:
            m = (y2 - y1)/(x2 - x1)

        # Check to see if the point (x, y) is on the line and between the two
end points
        tol = 10
        if y - tol <= m*(x - x1) + y1 <= y + tol:
            if (min(x1, x2) <= x <= max(x1, x2)) and (min(y1, y2) <= y <=
max(y1, y2)):
                self.is_selected = True
        else:
            self.is_selected = False

    def __repr__(self):
        return ("Type: " + self.wire_type + " node_num: " + self.node_num +
                " x1: " + str(self.x1) + " y1: " + str(self.y1) +
                " x2: " + str(self.x2) + " y2: " + str(self.y2))

    def reprJson(self):
        return dict(type=self.wire_type, wire_dir=self.wire_dir,
node_num=self.node_num, x1=self.x1, y1=self.y1,
                    x2=self.x2, y2=self.y2, name=self.name)
```

Wire_Lib/node.py

```
from Helper_Lib import Point
from Comp_Lib import Connector


class Node:
    def __init__(self, canvas, x1, y1):
        self.canvas = canvas
        self.x1 = x1
        self.y1 = y1

        self.text = '0'
        self.comp_type = 'node'
        self.id = None
        self.text_id = None
        self.sel_id = None
        self.wire_list = []
        self.bbox = None
        self.angle = 0
        self.is_selected = False
        self.is_drawing = False
        self.conn = None
        self.conn_list = []

        self.create()

    def create(self):
        self.create_node()
        self.update_bbox()
        self.create_text()
        self.create_selector()
        self.create_connectors()
        self.set_connector_visibility()

    def create_node(self):
        self.id = self.canvas.create_oval(self.x1 - 5, self.y1 - 5, self.x1 +
5, self.y1 + 5, fill="black")

    def update_bbox(self):
        """Update the bounding box to get current gate coordinates"""
        self.bbox = self.canvas.bbox(self.id)

    def create_text(self):
        text_loc = Point(self.x1-10, self.y1-10)  # Put text above symbol
        self.text_id = self.canvas.create_text(text_loc.x, text_loc.y,
                                text=self.text, fill="black",
                                font='Helvetica 10 bold',
                                angle=self.angle, tags="text")

    def create_selector(self):
        """Create the red rectangle selector and check to see if the gate is
selected"""
        x1, y1, x2, y2 = self.bbox[0] - 5, self.bbox[1] - 5, self.bbox[2] + 5,
self.bbox[3] + 5
        self.sel_id = self.canvas.create_rectangle(x1, y1, x2, y2, fill=None,
outline="red", width=2)
        self.set_selector_visibility()

    def set_selector_visibility(self):
        """Set the selector visibility state"""
        if self.is_selected:
            self.canvas.itemconfig(self.sel_id, state='normal')
        else:
            self.canvas.itemconfig(self.sel_id, state='hidden')

    def create_connectors(self):
        # Calculate position of connectors from current comp position and size
        center = self.get_geometry()

        self.conn = Connector(self.canvas, "conn", center.x, center.y)
        self.conn_list = [self.conn]

    def get_geometry(self):
        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        w = x2 - x1
```

```python
        h = y2 - y1
        center = Point(x1 + w / 2, y1 + h / 2)

        return center

    def set_connector_visibility(self):
        """Set the connector visibility state"""
        if self.is_drawing:
            for c in self.conn_list:
                self.canvas.itemconfig(c.id, state='normal')
        else:
            for c in self.conn_list:
                self.canvas.itemconfig(c.id, state='hidden')

    def update(self):
        self.update_position()
        self.update_bbox()
        self.update_text()
        self.update_selector()
        self.update_connectors()
        self.set_connector_visibility()

    def update_position(self):
        self.canvas.coords(self.id, self.x1 - 5, self.y1 - 5, self.x1 + 5, self.y1 + 5)  # Update position

    def update_text(self):
        self.canvas.itemconfig(self.text_id, text=self.text)
        self.canvas.coords(self.text_id, self.x1 - 10, self.y1 - 10)

    def update_selector(self):
        """Update the red rectangle selector coordinates and check to see if the gate is selected"""
        x1, y1, x2, y2 = self.bbox[0] - 5, self.bbox[1] - 5, self.bbox[2] + 5, self.bbox[3] + 5
        self.canvas.coords(self.sel_id, x1, y1, x2, y2)
        self.set_selector_visibility()

    def update_connectors(self):
        """Update the position of connectors here"""
        center = self.get_geometry()

        self.conn.x, self.conn.y = center.x, center.y

        for c in self.conn_list:
            c.update()
        self.move_connected_wires()

    def check_connector_hit(self, x, y):
        """Hit test to see if a connector is at the provided x, y
coordinates"""
        for conn in self.conn_list:
            if conn.connector_hit_test(x, y):
                return conn
        return None

    def move_connected_wires(self):
        """Resize connected wires if the shape is moved"""
        for connection in self.wire_list:  # comp_conn, wire_name, wire_end
            for connector in self.conn_list:
                if connector.name == connection.comp_conn:
                    wire_obj = self.canvas.wire_dict[connection.wire_name]
                    if connection.wire_end == "begin":
                        wire_obj.x1 = connector.x
                        wire_obj.y1 = connector.y
                    elif connection.wire_end == "end":
                        wire_obj.x2 = connector.x
                        wire_obj.y2 = connector.y

    def rotate(self):
        """Set the rotation angle to the current angle + 90 deg, reset to 0 deg
if angle > 270 deg"""
        self.angle += 90
        if self.angle > 270:
            self.angle = 0

    def hit_test(self, x, y):
        x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]
        if x1 <= x <= x2 and y1 <= y <= y2:
            self.is_selected = True
        else:
            self.is_selected = False

    def __repr__(self):
        return ("Type: " + self.comp_type + " Text: " + self.text + " x1: " +
str(self.x1) + " y1: " +
                str(self.y1) + " wire list: " + str(self.wire_list.__repr__()))

    def reprJson(self):
        return dict(type=self.comp_type, text=self.text, x1=self.x1,
y1=self.y1, angle=self.angle,
                    wire_list=self.wire_list)
```

Wire_Lib/wire_selector.py

```
class WireSelector:
    def __init__(self, canvas, name, x, y):
        self.canvas = canvas
        self.name = name
        self.x = x
        self.y = y
        self.is_selected = False

        self.id = self.canvas.create_oval(self.x - 5, self.y - 5, self.x + 5, self.y + 5,
                                          state='normal', fill="white")

    def update(self):
        self.update_position()
        self.update_selection()

    def update_position(self):
        """Update the selector position"""
        sel_points = [self.x - 5, self.y - 5, self.x + 5, self.y + 5]
        self.canvas.coords(self.id, sel_points)

    def update_selection(self):
        if self.is_selected:
            self.canvas.itemconfigure(self.id, fill="yellow")
        else:
            self.canvas.itemconfigure(self.id, fill="white")

    def selector_hit_test(self, event_x, event_y):
        if self.x-5 <= event_x <= self.x+5 and self.y-5 <= event_y <= self.y+5:
            self.is_selected = True
            self.update_selection()
            return True
        else:
            self.is_selected = False
            self.update_selection()
            return False
```

analog_simulator.py

```
import customtkinter as ctk
from UI_Lib import LeftFrame, TopFrame, Canvas

ctk.set_appearance_mode("System")  # Modes: "System" (standard), "Dark",
"Light"
ctk.set_default_color_theme("blue")  # Themes: "blue" (standard), "green",
"dark-blue"


class App(ctk.CTk):
    """ctk.CTk is a CustomTkinter main window, similar to tk.Tk tkinter main
window"""
    def __init__(self):
        super().__init__()
        self.geometry("1200x800x100x100")  # w, h, x, y
        self.title("Analog Simulator")

        self.canvas = Canvas(self)
        self.left_frame = LeftFrame(self, self.canvas)
        self.top_frame = TopFrame(self, self.canvas)

        self.top_frame.pack(side=ctk.TOP, fill=ctk.BOTH)
        self.left_frame.pack(side=ctk.LEFT, fill=ctk.BOTH)
        self.canvas.pack(side=ctk.LEFT, fill=ctk.BOTH, expand=True)

        # Add bindings here
        self.bind("<Configure>", self.on_window_resize)
        self.bind('<r>', self.rotate_comp)
        self.bind('<h>', self.canvas.set_horiz_dir)
        self.bind('<v>',self.canvas.set_vert_dir)
        self.canvas.bind('<Button-3>', self.canvas.edit_shape)

    def on_window_resize(self, _event):
        self.canvas.redraw()

    def rotate_comp(self, _event=None):
        if self.canvas.mouse.selected_comp:
            self.canvas.mouse.selected_comp.rotate()
            self.canvas.redraw()


if __name__ == "__main__":
    """Instantiate the Microwave Simulator app and run the main loop"""
    app = App()
    app.mainloop()
```

Conclusion

This concludes all EE Tools design projects. I hope you enjoyed the journey as much as I did. Ever since I designed my first microwave circuit on Touchstone, I wanted to know how to write electrical engineering software design tools. As you can see, it involves learning a software language such as Python, taking incremental development steps such as the projects in this book:

- Python Tutorial
- Beginner Project - Scientific Calculator
- Intermediate Projects
  - Shape Editor
  - Line Editor
- Advanced Projects
  - Diagram Editor
  - Digital Circuit Simulator
  - Microwave Circuit Simulator
  - Analog Circuit Simulator

There is much room for improvement and the reader is encouraged to modify the code to add features, make it more robust, and explore areas of interest. Good luck in your engineering journey.

R. A. Crist