Chapter 5 - Diagram Editor

The first advanced project is a Diagram Editor which will integrate the techniques and solutions that were developed in the Shape Editor and Line Editor applications.

Project Design, Features, and Specifications

Approach:

- Modify the LineEditor App to create the Diagram Editor App
- Create a custom Canvas Class
  - Create a custom Mouse Class
- Create a custom Top Frame Class
  - File Menu
  - Settings Menu
  - Help Menu
  - Shape Appearance Controls
  - Snap Settings Controls
- Create a custom Left Frame Class
  - Shape Buttons

Shape Features:

- Rectangle Class
- Oval Class
- Triangle Class
- Text Class
- Image Class
- Create new shapes by drawing shape with left mouse button
- Select and move shapes with left mouse button
- Unselect shapes with left mouse button, if selected
- Rotate shapes with 'r' key
- Resize shapes with left mouse button
- Show selectors when shape is selected that can be used to resize it
- Snap-to-grid for move and resize

Line Features:

- Straight Line Class
- Segment Line Class
- Elbow Line Class
- Draw new line with left mouse button
- Select line, show selectors
- Unselect line with left mouse button, if selected
- Move line with left mouse button
- Resize selected line ends with left mouse button
- Snap-to-Grid for move and resize
- Snap-to-Shape if begin or end point hits a shape connector
  - Move line end when connected shape is moved, resized, or rotates
  - If segmented or elbow line, change line direction using 'h' and 'v' keys

Deployment:

- Create diagram_editor.exe for windows using PyInstaller

Project Setup

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

External libraries:

- pip install customtkinter
- pip install ctkcolorpicker
- pip install pyInstaller - Create .exe file

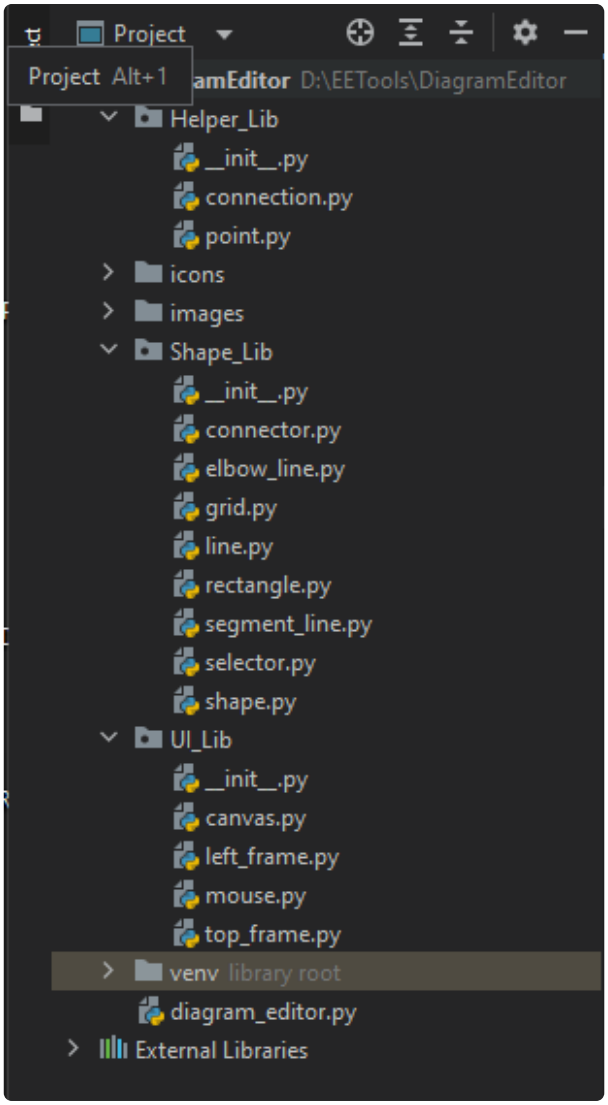
Initial Directory Structure

As an example of code reuse, we will start with the Line Editor Application and modify it to create the Diagram Editor Application.

Objectives:

- Copy the Line Editor files and directories into the Diagram Editor project
- Modify the main file
- Yes, we will use the icons and images directories in this project

Initial Directory Structure

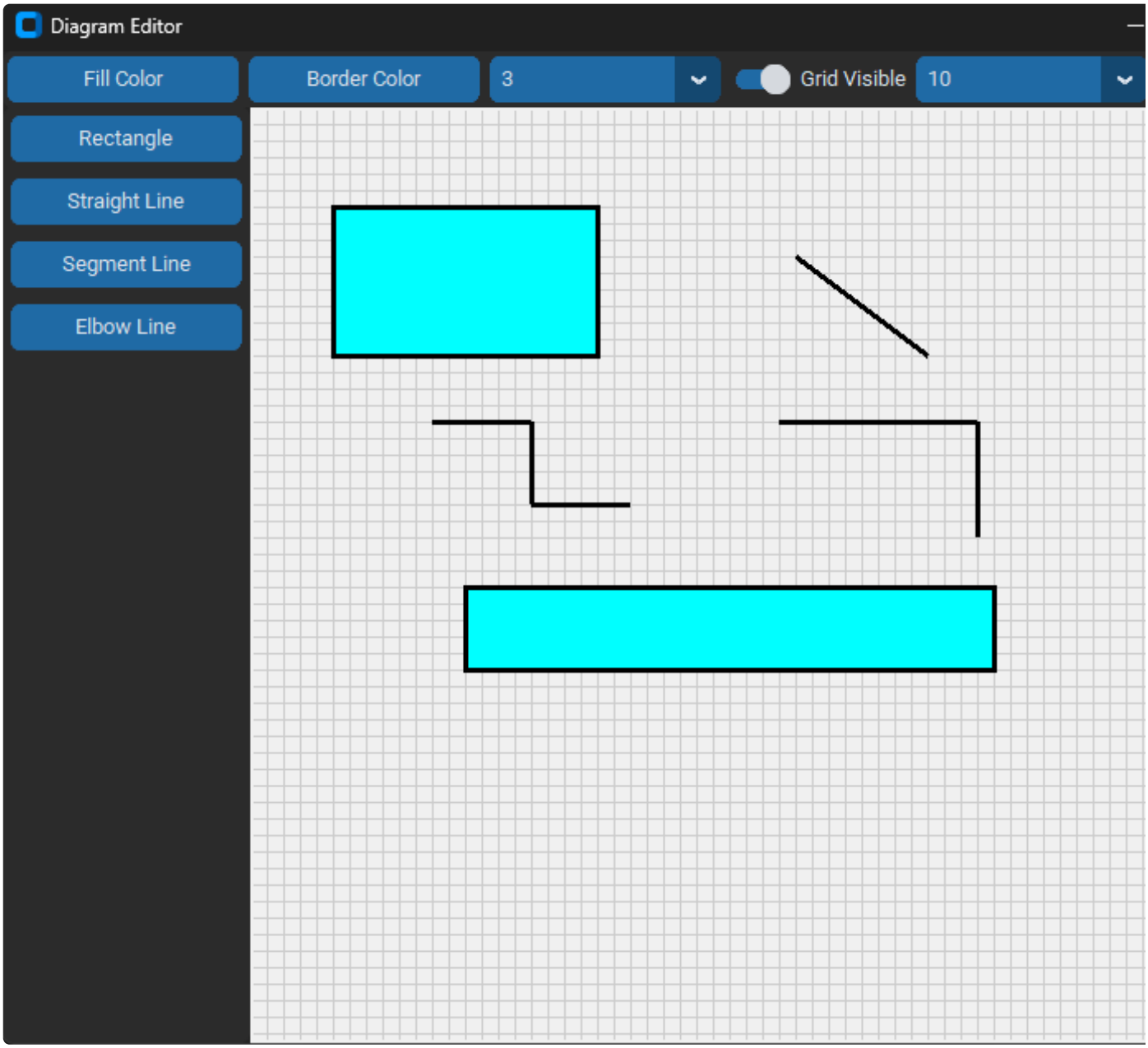


Main Diagram Editor App Class

diagram_editor.py

```
import customtkinter as ctk
from UI_Lib import *
```

```
class DiagramEditorApp(ctk.CTk):
    """ctk.CTk is a CustomTkinter main window, similar to tk.Tk tkinter main
window"""
    def __init__(self):
        super().__init__()
        self.geometry("800x600x100x100")  # w, h, x, y
        self.title("Diagram Editor")

        # Create Canvas widget
        self.canvas = Canvas()
        a_top_frame = TopFrame(self, self.canvas)
        a_left_frame = LeftFrame(self, self.canvas)

        a_top_frame.pack(side=ctk.TOP, fill=ctk.BOTH)
        a_left_frame.pack(side=ctk.LEFT, fill=ctk.BOTH)
        self.canvas.pack(side=ctk.RIGHT, fill=ctk.BOTH, expand=True)

        # Declare mouse bindings
        self.bind('<r>', self.canvas.rotate_shape)
        self.bind('<h>', self.canvas.set_horizontal_line_direction)
        self.bind('<v>', self.canvas.set_vertical_line_direction)
        self.bind("<Configure>", self.on_window_resize)

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


if __name__ == "__main__":
    # Instantiate the Line Editor application and run the main loop
    app = DiagramEditorApp()
    app.mainloop()
```



So far, so good. This should look like the Line Editor from Chapter 4.

Oval Class

Objectives:

- Create a new Oval Class using code from the Rectangle Class
- Modify the Left Frame Menu with an Oval Button
- Modify the Mouse Class to create an Oval in the Draw Left Down method

The Oval Class can be created from the Rectangle Class by creating a new oval.py file, copy the code from rectangle.py into it, and make 2 code modifications: change the class name to Oval and change the draw() method to call canvas draw oval method.

oval.py - changes only

```
. . .

class Oval(Shape):  # Change
    def __init__(self, canvas, x1, y1, x2, y2):
        super().__init__(canvas, x1, y1, x2, y2)

. . .

    def draw(self):
        self.points = [self.x1, self.y1, self.x2, self.y2]
        self.canvas.create_oval(self.points, fill=self.fill_color,
outline=self.border_color,  # Change
                                width=self.border_width)

. . .
```

Modify the Left Frame Class to add the Oval button and set the mouse current shape to "oval".

left_frame.py - changes only

```
. . .

        # Add left frame widgets here
        rect_button = ctk.CTkButton(self,
                                    text="Rectangle",
                                    command=self.create_rectangle)
        rect_button.pack(side=ctk.TOP, padx=5, pady=5)

        oval_button = ctk.CTkButton(self,  # Change
                                    text="Oval",
                                    command=self.create_oval)
        oval_button.pack(side=ctk.TOP, padx=5, pady=5) # Change

. . .
```

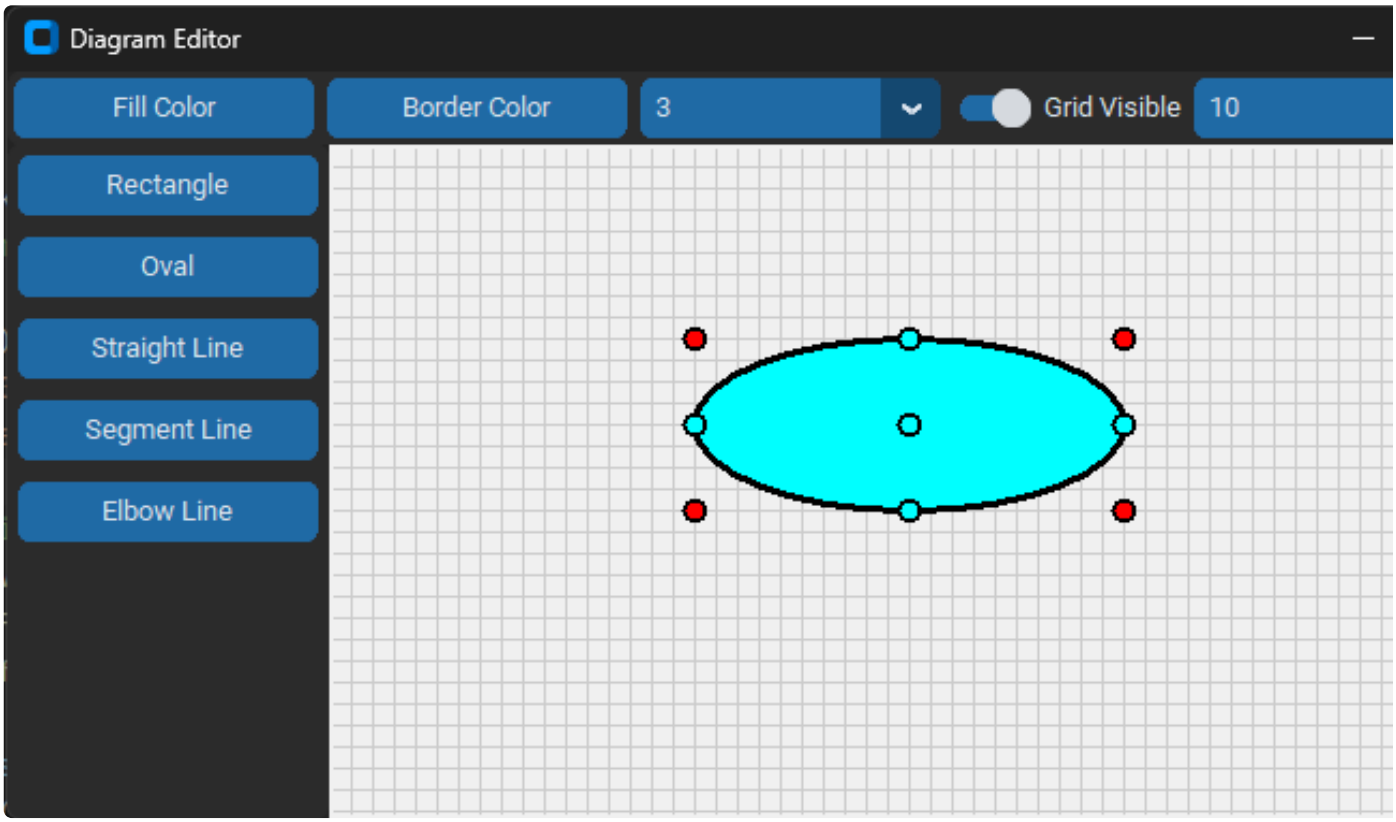
```
    def create_rectangle(self):
        self.canvas.mouse.mode = None
        self.canvas.draw_shapes()
        self.canvas.mouse.current_shape = "rectangle"
        self.canvas.mouse.draw_bind_mouse_events()

    def create_oval(self):  # Change
        self.canvas.mouse.mode = None
        self.canvas.draw_shapes()
        self.canvas.mouse.current_shape = "oval"
        self.canvas.mouse.draw_bind_mouse_events()

. . .
```

Modify the Mouse Class to import the Oval Class and create an oval in the Draw Left Down method.

```
from Helper_Lib.point import Point
from Helper_Lib.connection import Connection
from Shape_Lib.rectangle import Rectangle
from Shape_Lib.oval import Oval  # Change
from Shape_Lib.line import Line
from Shape_Lib.segment_line import SegmentLine
from Shape_Lib.elbow_line import ElbowLine

. . .

   def draw_left_down(self, event):
        self.unselect_all_shapes()
        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)

        if self.current_shape == "rectangle":
            self.current_shape_obj = Rectangle(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
        elif self.current_shape == "oval":
            self.current_shape_obj = Oval(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
        elif self.current_shape == "line":  # Change
            self.current_shape_obj = Line(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y) # Change
            self.select_connector(self.current_shape_obj, "begin",
```

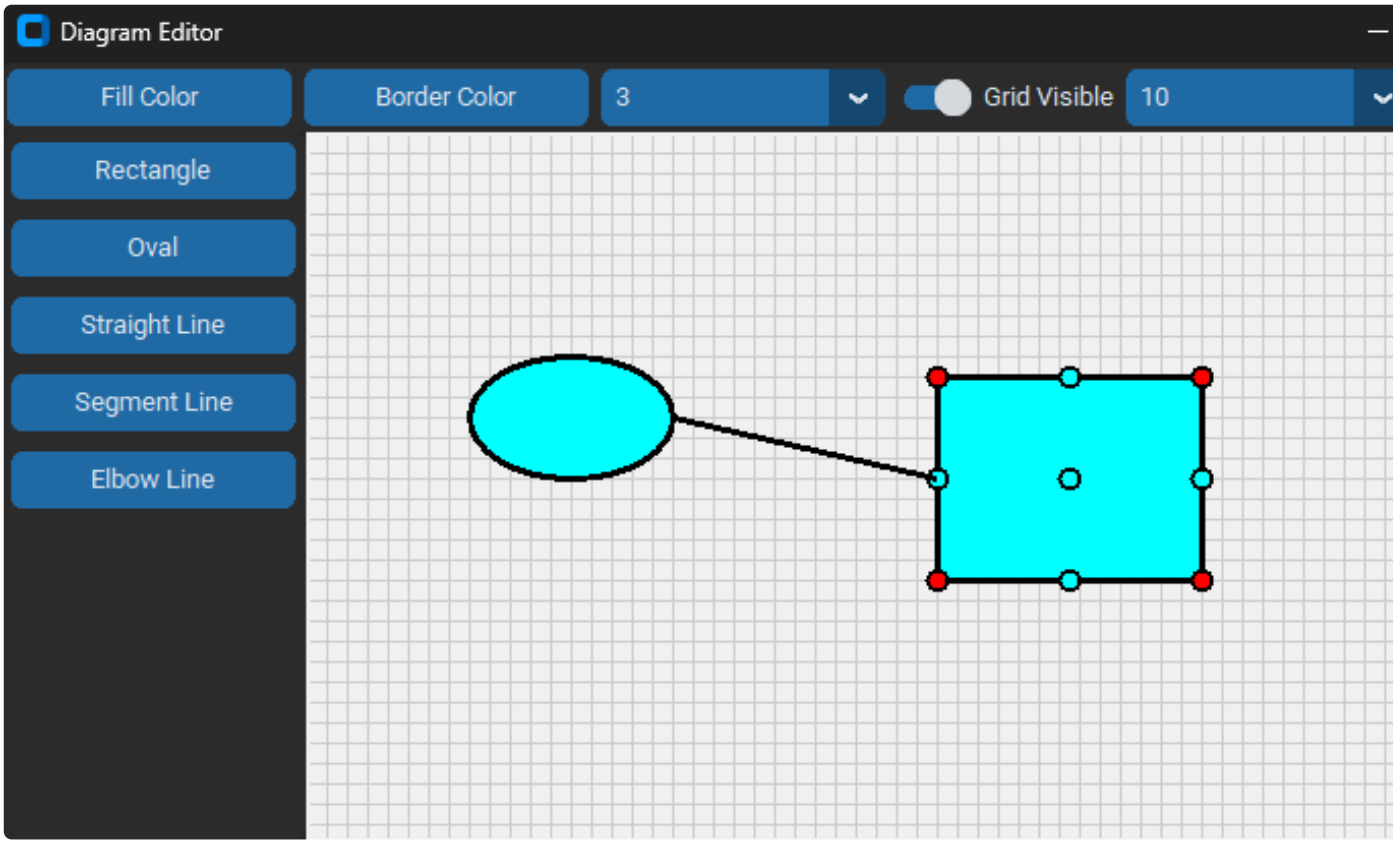
```
self.start.x, self.start.y) # Change
        elif self.current_shape == "segment":
            self.current_shape_obj = SegmentLine(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
            self.select_connector(self.current_shape_obj, "begin",
self.start.x, self.start.y)
        elif self.current_shape == "elbow":
            self.current_shape_obj = ElbowLine(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
            self.select_connector(self.current_shape_obj, "begin",
self.start.x, self.start.y)

. . .
```

Notice how the code structure can be easily modified to add new features.



Create a new diagram with one oval, one rectangle, and one line drawn from a connector on the oval to a connector on the rectangle. Move the oval and rectangle and confirm that

the line automatically resizes.



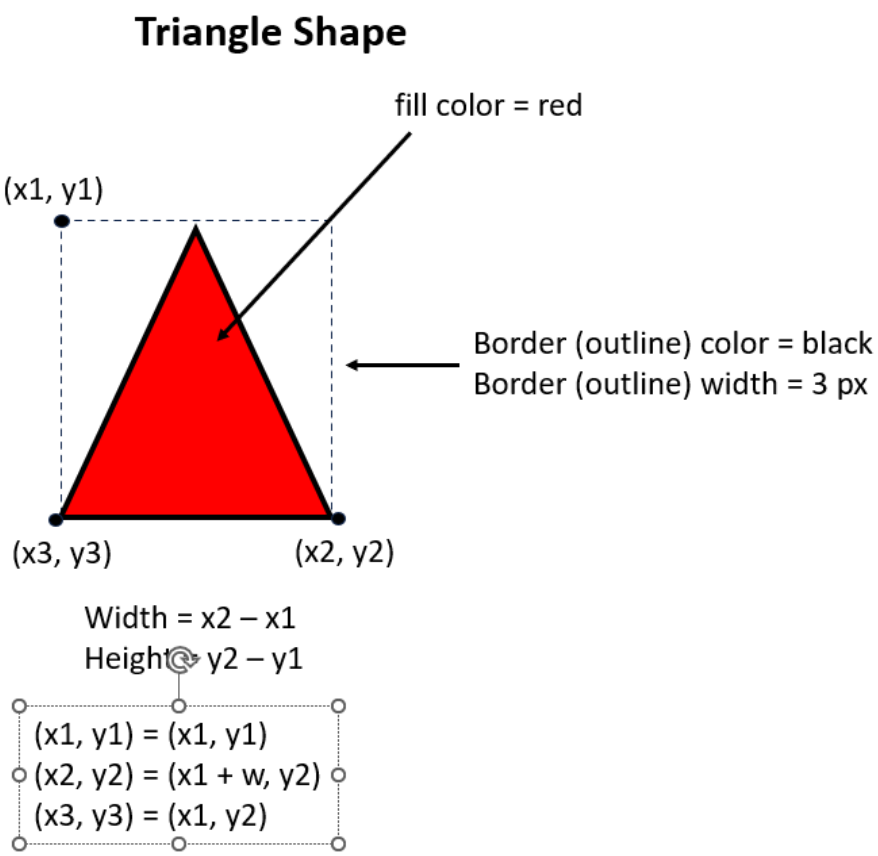
Triangle Class

Objectives:

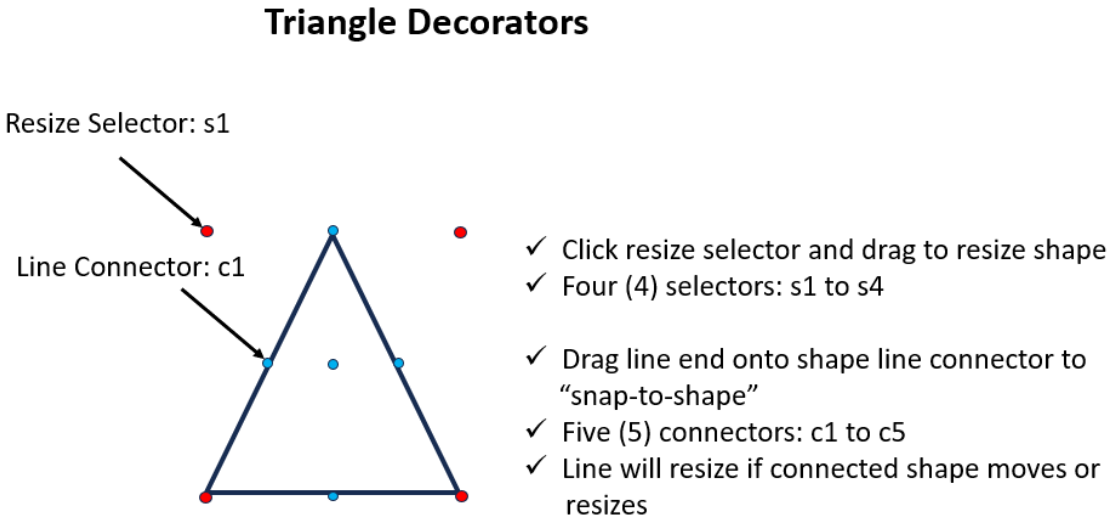
- Create a new Triangle Class starting with code from the Rectangle Class
- Modify the Left Frame Menu with a Triangle Button
- Modify the Mouse Class to create a Triangle in the Draw Left Down method

The Triangle Class can be created from the Rectangle Class by creating a new triangle.py file, copy the code from rectangle.py into it. The modifications for a Triangle Class are more extensive than the Oval Class. We will draw the Triangle using a `create_polygon` canvas method that needs 3 points.

The Triangle will be drawn as an isosceles triangle as shown in the figure below. It will be drawn in a bounding box with x1, y1 in the upper-left corner, x2, y2 in the lower-right corner, and x3, y3 in the lower-left corner. Note that x3, y3 = x1, y2 is easily calculated.

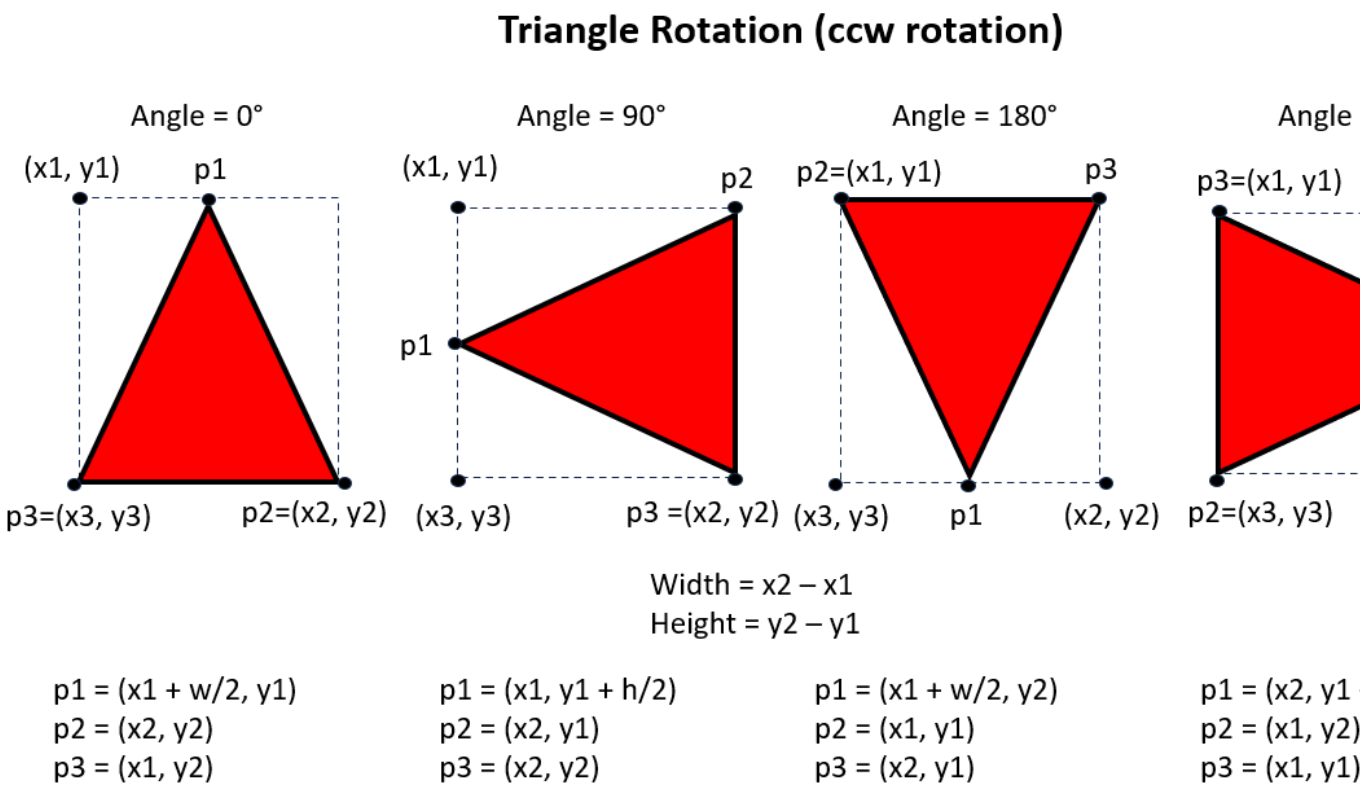


The Triangle Selectors and Connectors are shown in the figure below. It will have 4 selectors in the corners of the bounding box and 5 connectors with one at the top, 2 at the center of the triangle sides, one in the center of the bottom side, and one at the triangle center.



Triangle resize and move do not require any changes. However, Triangle rotation requires changes because the triangle is not symmetric like a rectangle and oval. We will need to

draw the Triangle for each angle in 90 degree increments. The points for the rotated Triangle are shown in the figure below.



Triangle Class

Create a file named triangle.py in the Shape_Lib directory. Copy the code from rectangle.py to triangle.py.

Imports do not change

```
from Shape_Lib.shape import Shape
from Shape_Lib.selector import Selector
from Shape_Lib.connector import Connector
from Helper_Lib.point import Point
```

Modify the class name to Triangle and create a `self.angle` variable in the initializer. No change to the number of selectors (4) or number of connectors (5).

```
class Triangle(Shape):  # Change
    def __init__(self, canvas, x1, y1, x2, y2):
```

```
        super().__init__(canvas, x1, y1, x2, y2)
        self.selector = None
        self.x3, self.y3 = x1, y2  # Change

        self.angle = 0  # Change

        self.fill_color = "cyan"
        self.border_color = "black"
        self.border_width = 3

        # Create 4 selectors
        self.s1, self.s2, self.s3, self.s4 = None, None, None, None
        self.create_selectors()

        # Create 5 connectors
        self.c1, self.c2, self.c3, self.c4, self.c5 = None, None, None, None,
None
        self.create_connectors()
```

In the draw() method, use the canvas.create_polygon method to draw the triangle from the points list. When the triangle is rotated, the points list will change depending on the rotation angle.

```
def draw(self):
    self.rotation_points()  # Change
    self.canvas.create_polygon(self.points, fill=self.fill_color,
outline=self.border_color,  # Change
                               width=self.border_width)

    if self.is_selected:
        self.draw_selectors()
        self.draw_connectors()

    if self.canvas.mouse.mode == "line_draw":
        self.draw_connectors()
```

No change is required for the `create_selectors()`, `draw_selectors()`, or `create_connectors()` methods.

Due to the non-symmetric shape of a triangle, the `draw_connectors()` method needs to change the position of the connectors based on the current angle.

```
    def draw_connectors(self):
        # Recalculate position of connectors from current shape position and
size
        w, h = self.x2 - self.x1, self.y2 - self.y1
        center = Point(self.x1 + w / 2, self.y1 + h / 2)

        # Define 5 connectors: shape center, 4 side centers - number of
connectors is unique to each shape
        if self.angle == 0 or self.angle == 180:  # Change
            self.c1.x, self.c1.y = center.x, center.y
            self.c2.x, self.c2.y = center.x, center.y - h / 2
            self.c3.x, self.c3.y = center.x + w / 4, center.y
            self.c4.x, self.c4.y = center.x, center.y + h / 2
            self.c5.x, self.c5.y = center.x - w / 4, center.y
        elif self.angle == 90 or self.angle == 270:  # Change
            self.c1.x, self.c1.y = center.x, center.y           # center
            self.c2.x, self.c2.y = center.x - w/2, center.y     # left peak
            self.c3.x, self.c3.y = center.x, center.y - h/4     # top center
            self.c4.x, self.c4.y = center.x + w/2, center.y     # right
            self.c5.x, self.c5.y = center.x, center.y + h/4     # bottom center

        # Draw the connectors
        for c in self.conn_list:
            c.draw()

                self.move_connected_lines()
```

The rotate() method calculates the angle of the shape each time the user presses the 'r' key on the keyboard. The `rotation_points()` method that recalculates the points list such that the triangle is drawn in the correct orientation. Note that the draw() method calls the `rotation_points()` method to get the points list.

```
    def rotate(self):
        # Calculate rotation angle
        self.angle += 90
        if self.angle > 270:
            self.angle = 0

    def rotation_points(self):
        w = self.x2 - self.x1
        h = self.y2 - self.y1
        if self.angle == 0:
            self.points = [self.x1 + w / 2, self.y1, self.x2, self.y2, self.x1,
```

```
self.y2]
        elif self.angle == 90:
            self.points = [self.x1, self.y1 + h/2, self.x2, self.y1, self.x2,
self.y2]
        elif self.angle == 180:
            self.points = [self.x1 + w/2, self.y2, self.x1, self.y1, self.x2,
self.y1]
        elif self.angle == 270:
            self.points = [self.x2, self.y1 + h/2, self.x1, self.y2, self.x1,
self.y1]
```

Interestingly, no change is required to the resize() method.

Left Frame Class - changes only

```
. . .
        # Add left frame widgets here
        rect_button = ctk.CTkButton(self,
                                    text="Rectangle",
                                    command=self.create_rectangle)
        rect_button.pack(side=ctk.TOP, padx=5, pady=5)

        oval_button = ctk.CTkButton(self,
                                    text="Oval",
                                    command=self.create_oval)
        oval_button.pack(side=ctk.TOP, padx=5, pady=5)

        tri_button = ctk.CTkButton(self,  # Change
                                    text="Triangle",
                                    command=self.create_triangle)
        tri_button.pack(side=ctk.TOP, padx=5, pady=5)  # Change
. . .
    def create_rectangle(self):
        self.canvas.mouse.mode = None
        self.canvas.draw_shapes()
        self.canvas.mouse.current_shape = "rectangle"
        self.canvas.mouse.draw_bind_mouse_events()

    def create_oval(self):
        self.canvas.mouse.mode = None
        self.canvas.draw_shapes()
        self.canvas.mouse.current_shape = "oval"
        self.canvas.mouse.draw_bind_mouse_events()
```

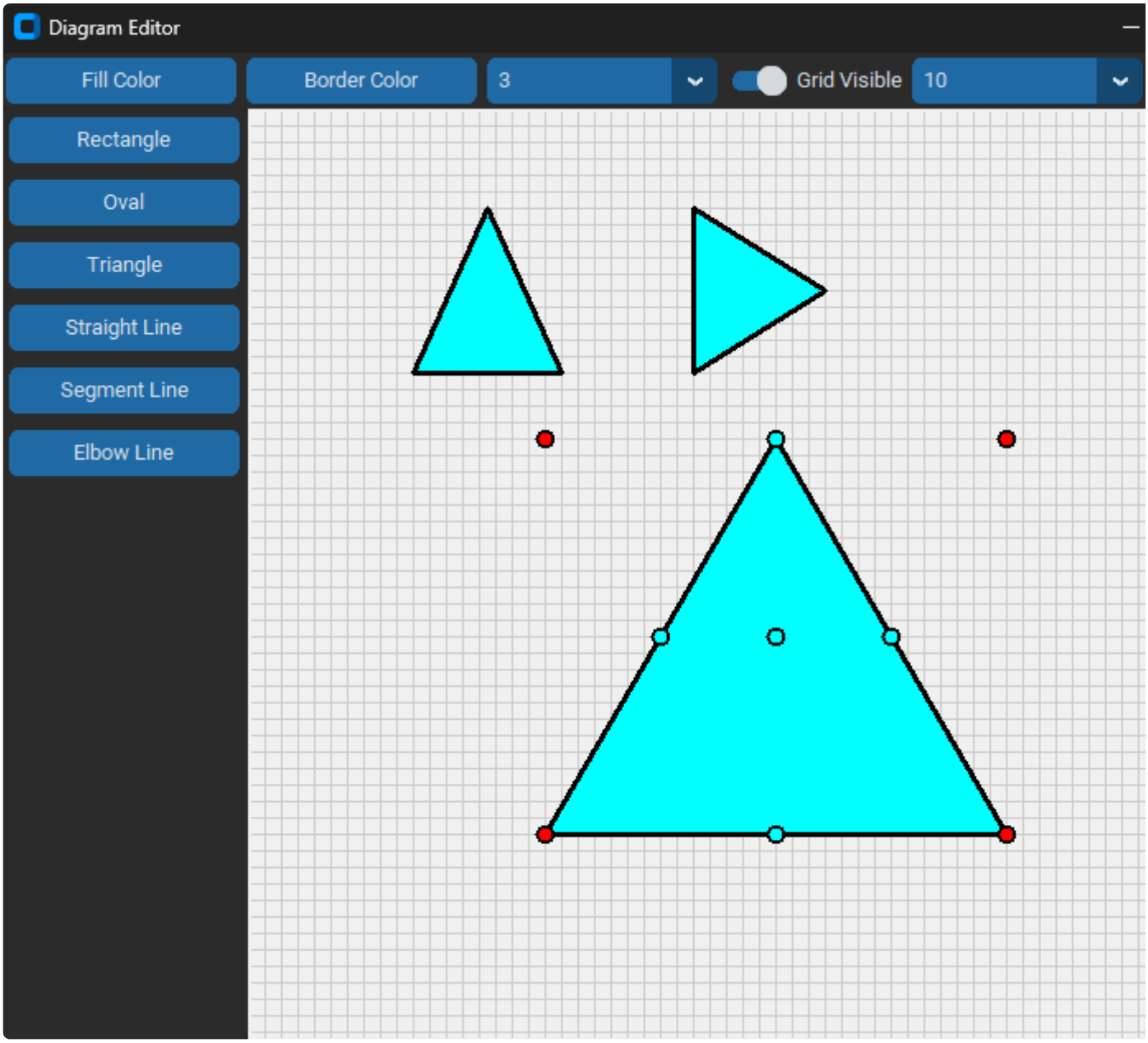
```
    def create_triangle(self):  # Change
        self.canvas.mouse.mode = None
        self.canvas.draw_shapes()
        self.canvas.mouse.current_shape = "triangle"
        self.canvas.mouse.draw_bind_mouse_events()
```

Modify the Mouse Class to import the Triangle Class and create a Triangle in the Draw Left Down method.
mouse.py

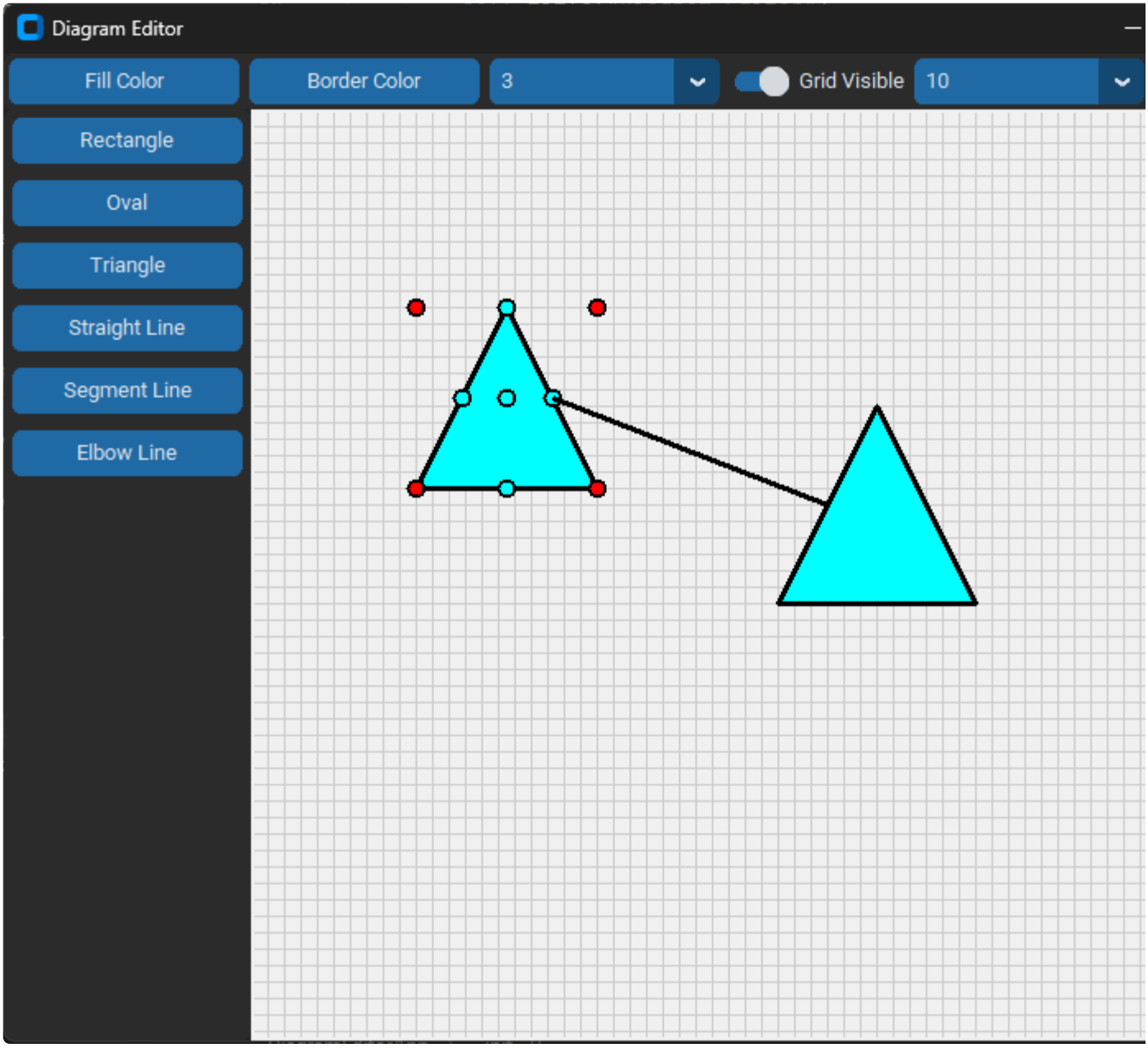
```
from Helper_Lib.point import Point
from Helper_Lib.connection import Connection
from Shape_Lib.rectangle import Rectangle
from Shape_Lib.oval import Oval
from Shape_Lib.triangle import Triangle  # Change
from Shape_Lib.line import Line
from Shape_Lib.segment_line import SegmentLine
from Shape_Lib.elbow_line import ElbowLine
. . .
    def draw_left_down(self, event):
        self.unselect_all_shapes()
        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)

        if self.current_shape == "rectangle":
            self.current_shape_obj = Rectangle(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
        elif self.current_shape == "oval":
            self.current_shape_obj = Oval(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
        elif self.current_shape == "triangle":  # Change
            self.current_shape_obj = Triangle(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)  # Change
        elif self.current_shape == "line":
            self.current_shape_obj = Line(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
            self.select_connector(self.current_shape_obj, "begin",
self.start.x, self.start.y)
        elif self.current_shape == "segment":
            self.current_shape_obj = SegmentLine(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
            self.select_connector(self.current_shape_obj, "begin",
```

```
self.start.x, self.start.y)
        elif self.current_shape == "elbow":
            self.current_shape_obj = ElbowLine(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
            self.select_connector(self.current_shape_obj, "begin",
self.start.x, self.start.y)

        if self.current_shape_obj is not None:
            self.canvas.shape_list.append(self.current_shape_obj)
            self.canvas.draw_shapes()
```

Run the program and verify that you can draw a Triangle, select and move it, rotate it, and resize it.



Create a diagram with 2 triangle and 1 line drawn between a connector on the first triangle to a connector on the second triangle. Move the triangles to confirm that the connected line resizes.



Text Class

Objectives:

- Create a new Text Class starting with code from the Rectangle Class
- Modify the Left Frame Menu with a Text Button
- Modify the Mouse Class to create a Text object in the Draw Left Down method
- Modify the text in our text object using a right click to show a dialog box
- Allow the user to enter text in the dialog box and update the text in the text object

The Text Class can be created from the Rectangle Class by creating a new text.py file, copy the code from rectangle.py into it, and make code modifications.

text.py
No change to imports

```
from Shape_Lib.shape import Shape
from Shape_Lib.selector import Selector
from Shape_Lib.connector import Connector
from Helper_Lib.point import Point
```

Text Class Initializer - Set the class name to Text. The draw text method only uses a single point x1, y1 so we initialize x2, y2 = 0, 0. The text displayed on the screen is stored in a variable named `self.text`. We add angle, `shape_id`, and `bbox` variables. We need to draw the shape in the initializer to initialize the `bbox` variable. We still need 4 selectors and 5 connectors however a text object cannot be resized, so the selectors indicate that the text is selected.

```
class Text(Shape):  # Change
    def __init__(self, canvas, x1, y1, x2=0, y2=0):  # Change
        super().__init__(canvas, x1, y1, x2, y2)
        self.selector = None
        self.text = 'Hello World'  # Change
        self.angle = 0  # Change

        self.shape_id = None  # Change
        self.bbox = None  # Change

        self.fill_color = "black"
        self.border_color = "black"
        self.border_width = 3

        self.draw()  # Change

        # Create 4 selectors
        self.s1, self.s2, self.s3, self.s4 = None, None, None, None
        self.create_selectors()

        # Create 5 connectors
        self.c1, self.c2, self.c3, self.c4, self.c5 = None, None, None, None,
```

```
None
        self.create_connectors()
```

Text Draw Method uses the `canvas.create_text()` method to draw the text on the canvas at point x1, y1, with text = `self.text`, fill color, font, angle, and tags set to "texts". We capture the shape ID and use it to find the bounding box coordinates from `self.canvas.bbox(self.shape_id)`.

```
def draw(self):
    self.points = [self.x1, self.y1, self.x2, self.y2]
    self.shape_id = self.canvas.create_text(self.x1, self.y1,     # Change
                            text=self.text, fill=self.fill_color,
                            font='Helvetica 15 bold',
                            angle=self.angle, tags="texts")
    self.bbox = self.canvas.bbox(self.shape_id)  # Change

    if self.is_selected:
        self.draw_selectors()
        self.draw_connectors()

    if self.canvas.mouse.mode == "line_draw":
        self.draw_connectors()
```

The create selectors, draw selectors, create connectors, and draw connectors method are modified to use the `bbox` coordinates to determine the location of each decorator: selector or connector.

```
def create_selectors(self):
    # Calculate position of selectors from current shape position
    x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]  #
Change

    # Create 4 selector objects: 4 corner of shape
    self.s1 = Selector(self.canvas, "s1", x1, y1)
    self.s2 = Selector(self.canvas, "s2", x2, y1)
    self.s3 = Selector(self.canvas, "s3", x2, y2)
    self.s4 = Selector(self.canvas, "s4", x1, y2)

    # Update the selector list
    self.sel_list = [self.s1, self.s2, self.s3, self.s4]

def draw_selectors(self):
```

```
    # Recalculate position of selectors from current shape position
    x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]  #
Change

    # Define 5 selectors: 4 shape corner - number of selectors is unique to
each shape
    self.s1.x, self.s1.y = x1, y1
    self.s2.x, self.s2.y = x2, y1
    self.s3.x, self.s3.y = x2, y2
    self.s4.x, self.s4.y = x1, y2

    # Draw the selectors
    for s in self.sel_list:
        s.draw()

def create_connectors(self):
    # Calculate position of connectors from current shape position and size
    x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]  #
Change
    w, h = x2 - x1, y2 - y1
    center = Point(x1 + w / 2, y1 + h / 2)

    # Define 5 connectors: shape center, 4 side centers - number of connectors
is unique to each shape
    self.c1 = Connector(self.canvas, "c1", center.x, center.y)
    self.c2 = Connector(self.canvas, "c2", center.x, center.y - h / 2)
    self.c3 = Connector(self.canvas, "c3", center.x + w / 2, center.y)
    self.c4 = Connector(self.canvas, "c4", center.x, center.y + h / 2)
    self.c5 = Connector(self.canvas, "c5", center.x - w / 2, center.y)

    # Update the connector list
    self.conn_list = [self.c1, self.c2, self.c3, self.c4, self.c5]

def draw_connectors(self):
    # Recalculate position of connectors from current shape position and size
    x1, y1, x2, y2 = self.bbox[0], self.bbox[1], self.bbox[2], self.bbox[3]  #
Change
    w, h = x2 - x1, y2 - y1
    center = Point(x1 + w / 2, y1 + h / 2)

    # Define 5 connectors: shape center, 4 side centers - number of connectors
is unique to each shape
    self.c1.x, self.c1.y = center.x, center.y
    self.c2.x, self.c2.y = center.x, center.y - h / 2
    self.c3.x, self.c3.y = center.x + w / 2, center.y
    self.c4.x, self.c4.y = center.x, center.y + h / 2
```

```
        self.c5.x, self.c5.y = center.x - w / 2, center.y

        # Draw the connectors
        for c in self.conn_list:
            c.draw()

        self.move_connected_lines()
```

Finally, the rotate() method calculates the current angle similar to the Triangle rotate() method. Text objects cannot be resized so we simply "pass" in the resize() method.

```
    def rotate(self):
        # Calculate rotation angle
        self.angle += 90
        if self.angle > 270:
            self.angle = 0

    def resize(self, offsets, event):
        pass
```

Left Frame Class - changes only

```
. . .
        tri_button = ctk.CTkButton(self,
                                   text="Triangle",
                                   command=self.create_triangle)
        tri_button.pack(side=ctk.TOP, padx=5, pady=5)

        text_button = ctk.CTkButton(self,  # Change
                                   text="Text",
                                   command=self.create_text)
        text_button.pack(side=ctk.TOP, padx=5, pady=5)
. . .
    def create_triangle(self):
        self.canvas.mouse.mode = None
        self.canvas.draw_shapes()
        self.canvas.mouse.current_shape = "triangle"
        self.canvas.mouse.draw_bind_mouse_events()

    def create_text(self):  # Change
        self.canvas.mouse.mode = None
        self.canvas.draw_shapes()
```

```
        self.canvas.mouse.current_shape = "text"
        self.canvas.mouse.draw_bind_mouse_events()
. . .
```

Modify the Mouse Class to import the Text Class, create a Text in the Draw Left Down method, and use the `bbox` coordinates in the select_shape method.
mouse.py

```
from Helper_Lib.point import Point
from Helper_Lib.connection import Connection
from Shape_Lib.rectangle import Rectangle
from Shape_Lib.oval import Oval
from Shape_Lib.triangle import Triangle
from Shape_Lib.text import Text  # Change
from Shape_Lib.line import Line
from Shape_Lib.segment_line import SegmentLine
from Shape_Lib.elbow_line import ElbowLine
. . .
    def draw_left_down(self, event):
        self.unselect_all_shapes()
        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)

        if self.current_shape == "rectangle":
            self.current_shape_obj = Rectangle(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
        elif self.current_shape == "oval":
            self.current_shape_obj = Oval(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
        elif self.current_shape == "triangle":
            self.current_shape_obj = Triangle(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
        elif self.current_shape == "text":  # Change
            self.current_shape_obj = Text(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)  # Change
        elif self.current_shape == "line":
            self.current_shape_obj = Line(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
            self.select_connector(self.current_shape_obj, "begin",
self.start.x, self.start.y)
. . .
    def select_shape(self, x, y):
```

```
        self.unselect_all_shapes()
        for shape in self.canvas.shape_list:
            if isinstance(shape, Text):
                if shape.bbox[0] <= x <= shape.bbox[2] and shape.bbox[1] <= y
<= shape.bbox[3]:
                    shape.is_selected = True
                    self.selected_obj = shape
                    self.canvas.draw_shapes()
            elif shape.x1 <= x <= shape.x2 and shape.y1 <= y <= shape.y2:
                shape.is_selected = True
                self.selected_obj = shape
                self.canvas.draw_shapes()
. . .
```

To change the displayed text we will display a text dialog box when the user selects the text and right clicks on the text. We will add a new mouse binding the `diagram_editor.py` file. The binding to the right mouse button `<Button-3>` calls a new event handler in the Canvas Class called `edit_shape()`.

```
. . .
        # Declare mouse bindings
        self.bind('<r>', self.canvas.rotate_shape)
        self.bind('<h>', self.canvas.set_horizontal_line_direction)
        self.bind('<v>', self.canvas.set_vertical_line_direction)
        self.bind("<Configure>", self.on_window_resize)
        self.canvas.bind('<Button-3>', self.canvas.edit_shape)  # Change
. . .
```

Canvas Class - changes only. We will add the `edit_shape()` event handler after the draw_shapes() method. The `edit_shape()` event handler checks to see if a Text object has been selected, if selected it opens a CTk Input Dialog box which allows the user to enter new text which is updated in the Text object using the `dialog.get_input()` method.

```
. . .
    def draw_shapes(self):
        self.delete('all')
        self.grid.draw()
        for shape in self.shape_list:
            shape.draw()

    def edit_shape(self, _event):  # Change
```

```
        print("edit shape called...")
            if isinstance(self.mouse.selected_obj, Text):
                dialog = ctk.CTkInputDialog(text="Enter new text", title="Edit
Text")
                self.mouse.selected_obj.text = dialog.get_input()
                self.draw_shapes()
. . .
```

Run the program and add 2 Text objects to the canvas. Select one of them and right click to show the text dialog box, input some new text, and close the dialog. Verify that the text has been updated.

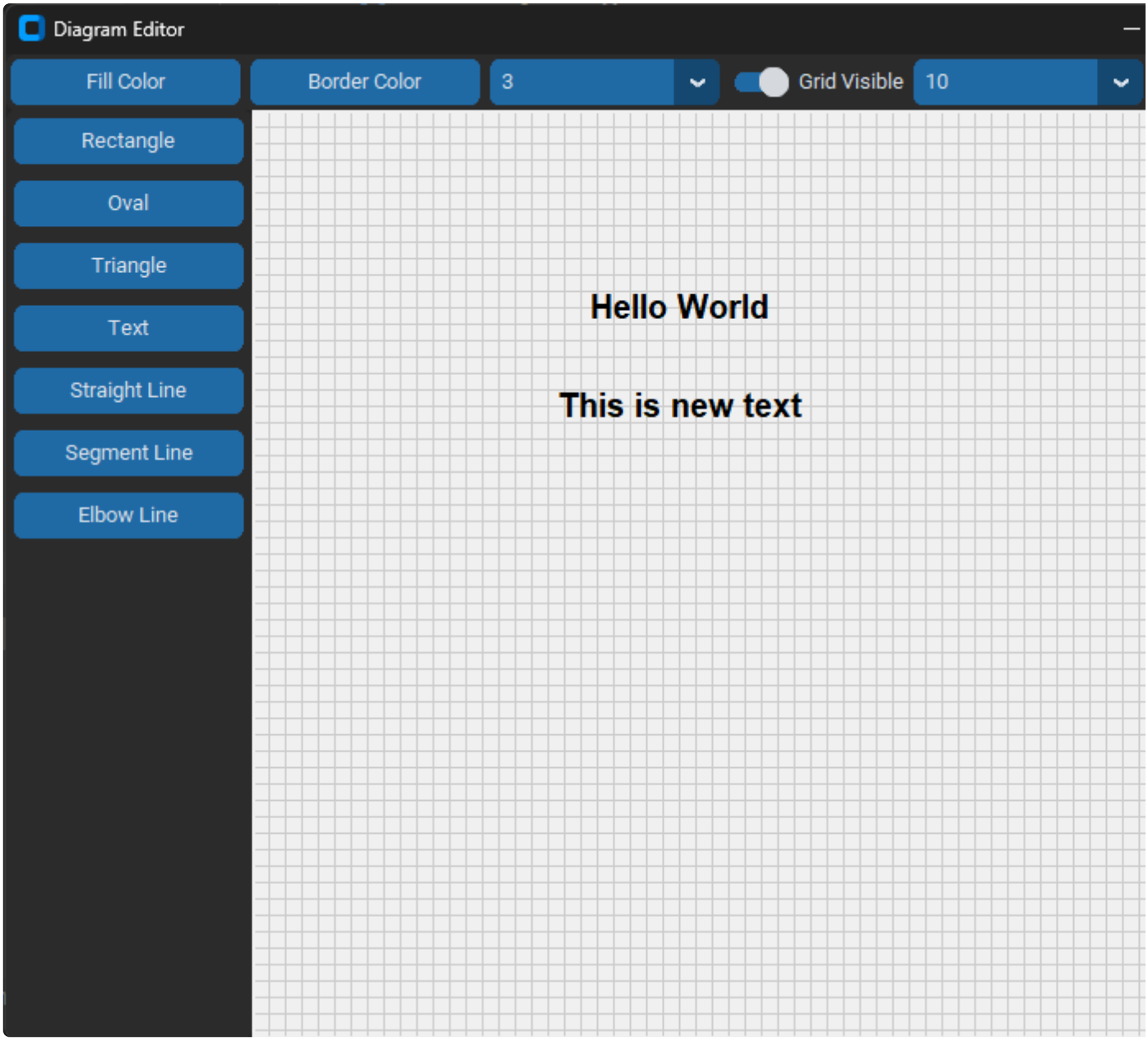


Image of the Edit Text dialog.

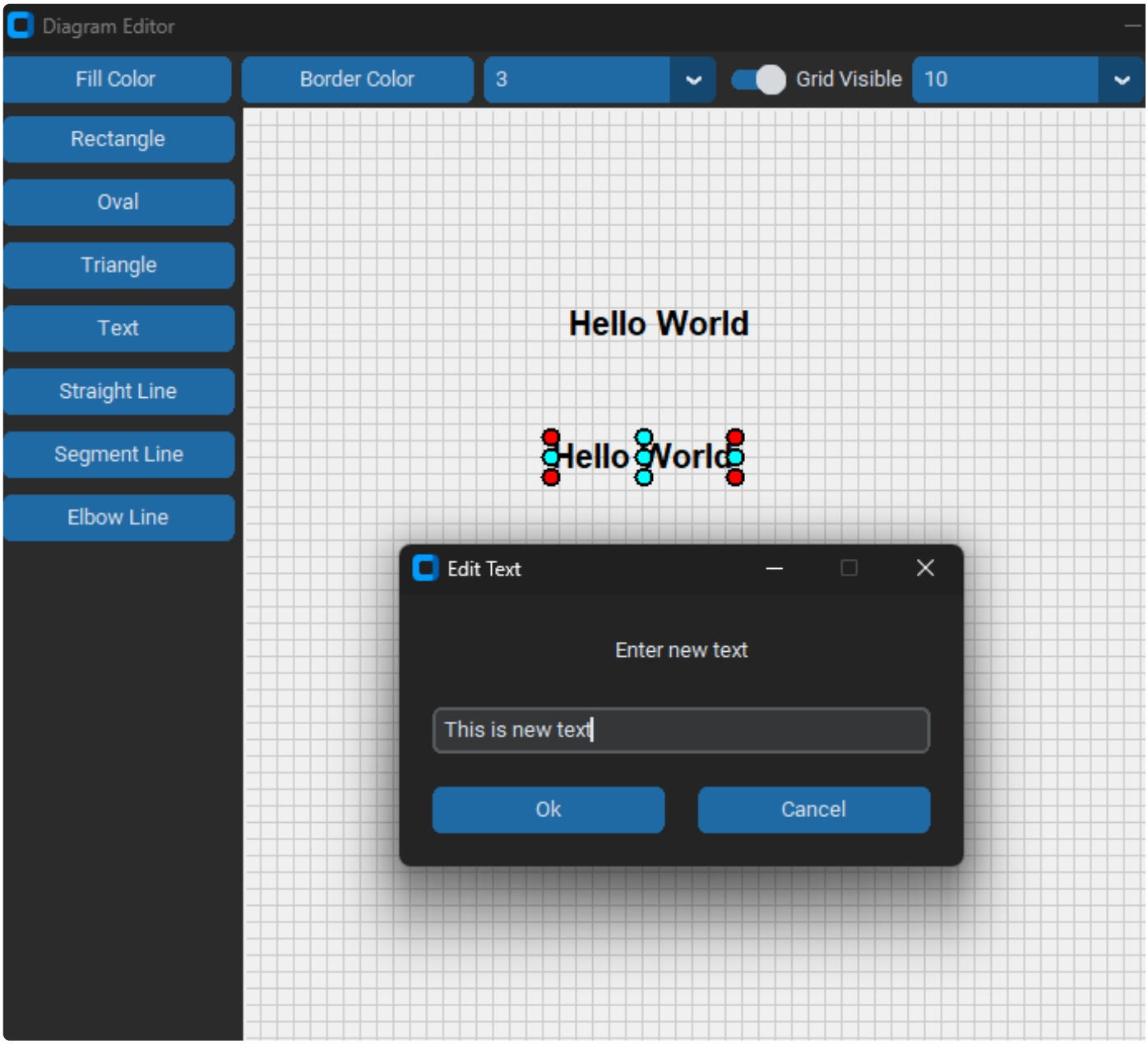


Image Class

The canvas also supports the creation of images. The image itself is loaded from computer disk drive using a tkinter fill dialog that asks the user to select a file to view. In many ways, an image operates in a similar way as a Text object so we will start with the Text Class and modify it to create the Picture Class, named to avoid conflicts with the Tkinter and CustomTkinter image classes.

Objectives:

- Create a new Picture Class starting with code from the Text Class

- Modify the Left Frame Menu with a Picture Button
- Modify the Mouse Class to create a Picture object in the Draw Left Down method
- Modify the image in our picture object using a right click to show a file dialog
- Allow the user to an image from the file system and update the image shown on the canvas

Create a new file called picture.py in the Shape_Lib directory. Open text.py and copy the code to picture.py.

picture.py - add imports for the PIL Image and `ImageTk` libraries as well as tkinter

```
from PIL import Image, ImageTk  # Change
import tkinter as tk  # Change

from Shape_Lib.shape import Shape
from Shape_Lib.selector import Selector
from Shape_Lib.connector import Connector
from Helper_Lib.point import Point
```

In the Picture Class initializer, create image variables and set the initial image filename.

```
class Picture(Shape):
    def __init__(self, canvas, x1, y1, x2=0, y2=0):
        super().__init__(canvas, x1, y1, x2, y2)
        self.selector = None
        self.a_image = None
        self.ph_image = None
        self.filename = "D:/EETools/DiagramEditor/images/hamburger.png"
        self.angle = 0

        self.shape_id = None
        self.bbox = None

        self.draw()

        # Create 4 selectors
        self.s1, self.s2, self.s3, self.s4 = None, None, None, None
        self.create_selectors()

        # Create 5 connectors
        self.c1, self.c2, self.c3, self.c4, self.c5 = None, None, None, None,
```

```
None
        self.create_connectors()
```

In the Picture Class draw() method, use `Image.open` to load the image file. Use the `image.rotate()` method to rotate it to the current angle, create a `PhotoImage` for the `canvas.create_image()` method. Images are created in a similar way as text, where the Image is created at the x1, y1 position which can be anchored to various places, in this case anchor = center of the image. We need to call the `canvas.bbox()` method on the shape ID to get the image dimensions as we did with the Text Class.

```
    def draw(self):
        self.a_image = Image.open(self.filename)
        self.a_image = self.a_image.rotate(self.angle)
        self.ph_image = ImageTk.PhotoImage(self.a_image)

        self.shape_id = self.canvas.create_image(self.x1, self.y1,
anchor=tk.CENTER, image=self.ph_image, tags="pics")
        self.bbox = self.canvas.bbox(self.shape_id)

        self.canvas.ph_image = self.ph_image
        self.canvas.a_image = self.a_image

        if self.is_selected:
            self.draw_selectors()
            self.draw_connectors()

        if self.canvas.mouse.mode == "line_draw":
            self.draw_connectors()
```

No changes are needed for the `create_selectors()`, `draw_selectors()`, `create_connectors()`, `draw_connectors()`, `rotate()`, and `resize()` methods copied from the Text Class, so we are done with the Picture Class definition.

Modify the Left Frame Class to add a Picture button and a button handler to create a picture.

```
. . .
        text_button = ctk.CTkButton(self,
                                    text="Text",
                                    command=self.create_text)
```

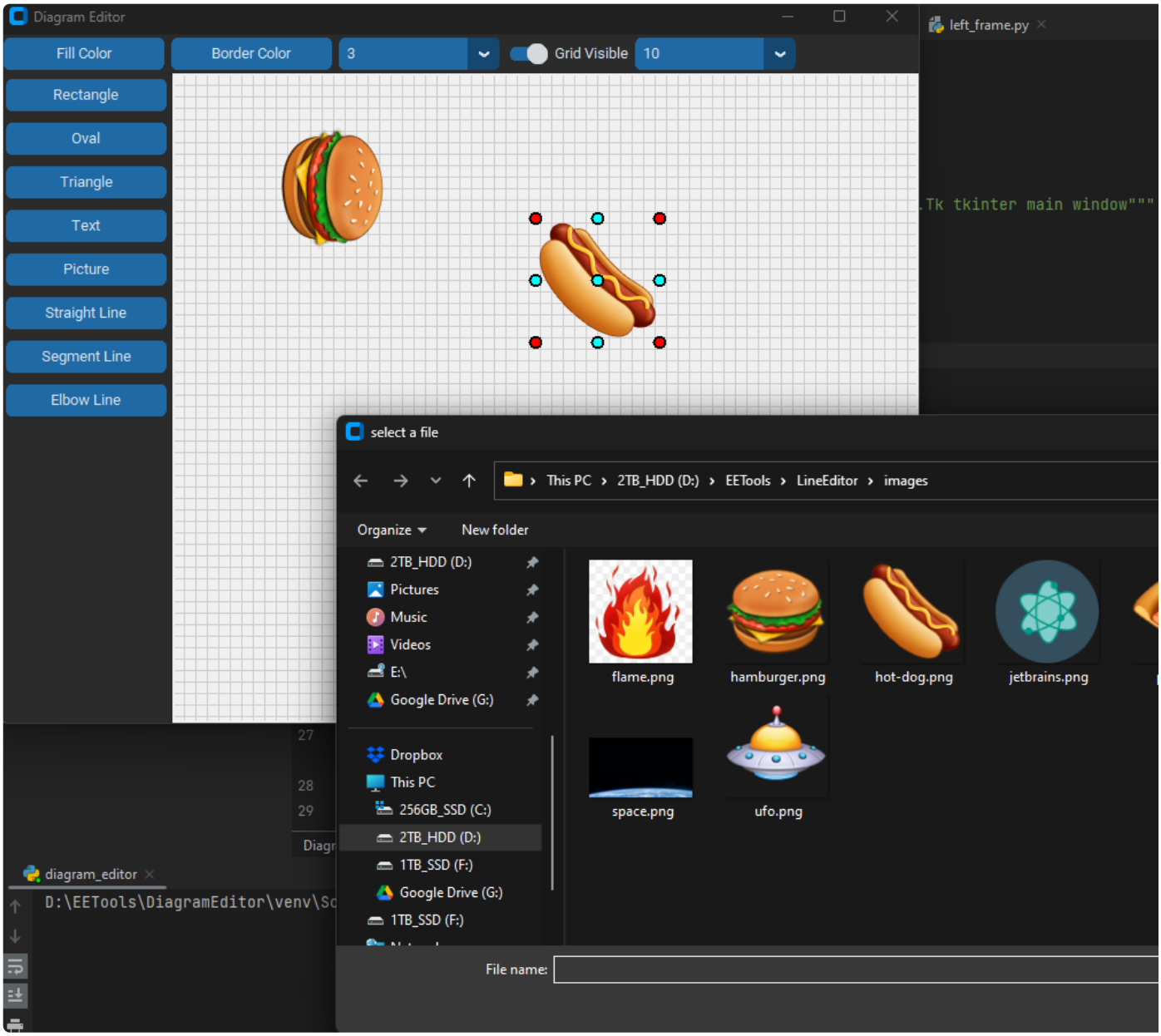
```
        text_button.pack(side=ctk.TOP, padx=5, pady=5)

        pic_button = ctk.CTkButton(self,  # Change
                                   text="Picture",
                                   command=self.create_picture)
        pic_button.pack(side=ctk.TOP, padx=5, pady=5)  # Change
. . .
    def create_text(self):
        self.canvas.mouse.mode = None
        self.canvas.draw_shapes()
        self.canvas.mouse.current_shape = "text"
        self.canvas.mouse.draw_bind_mouse_events()

    def create_picture(self):  # Change
        self.canvas.mouse.mode = None
        self.canvas.draw_shapes()
        self.canvas.mouse.current_shape = "picture"
        self.canvas.mouse.draw_bind_mouse_events()
. . .
```

Modify the Mouse Class to import the Picture Class, create a Picture in the Draw Left Down method, and use the `bbox` coordinates in the select_shape method.
mouse.py

```
from Helper_Lib.point import Point
from Helper_Lib.connection import Connection
from Shape_Lib.rectangle import Rectangle
from Shape_Lib.oval import Oval
from Shape_Lib.triangle import Triangle
from Shape_Lib.text import Text
from Shape_Lib.picture import Picture  # Change
from Shape_Lib.line import Line
from Shape_Lib.segment_line import SegmentLine
from Shape_Lib.elbow_line import ElbowLine
. . .
    def draw_left_down(self, event):
        self.unselect_all_shapes()
        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)

        if self.current_shape == "rectangle":
            self.current_shape_obj = Rectangle(self.canvas, self.start.x,
```

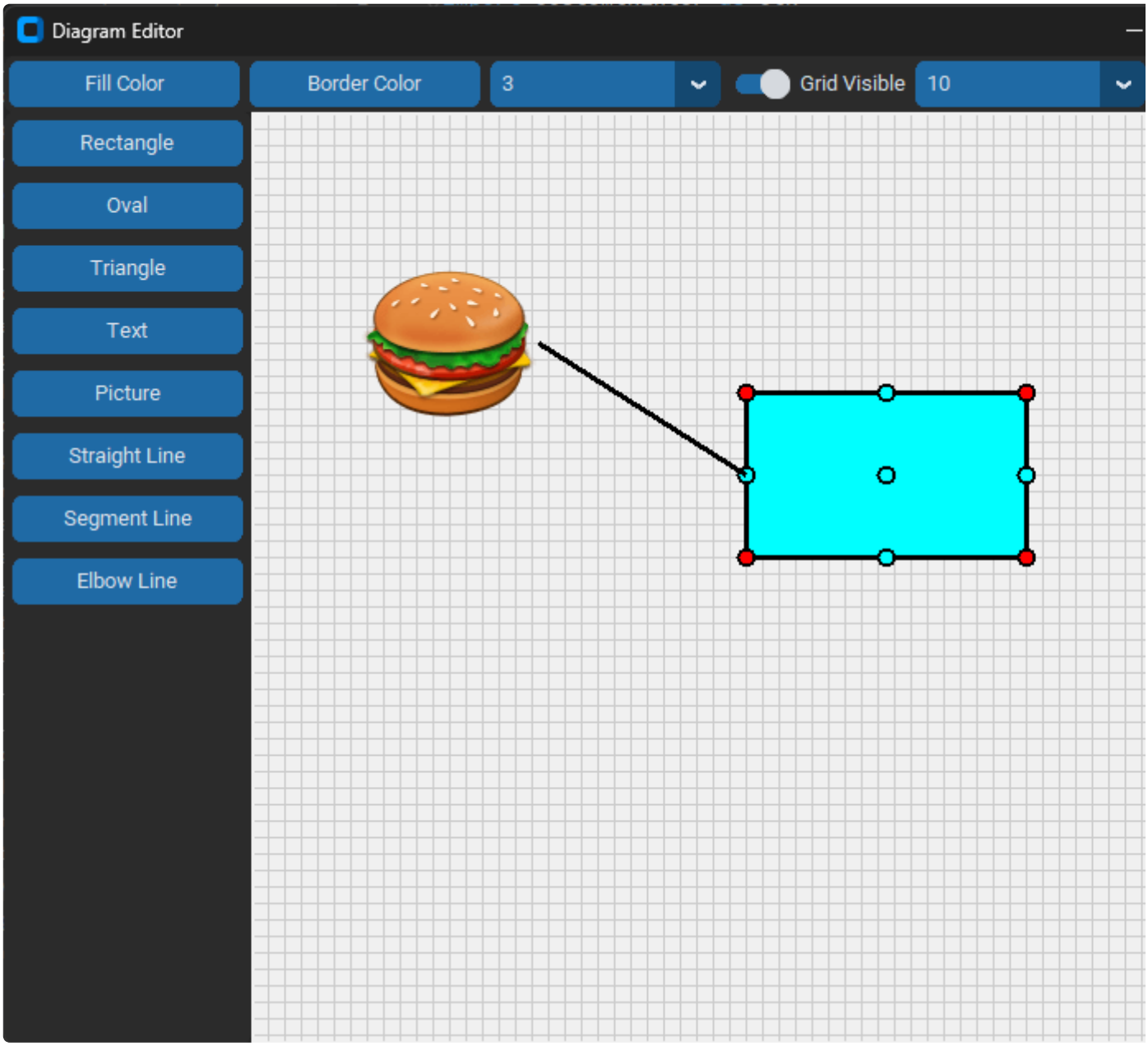
```
self.start.y, self.start.x, self.start.y)
        elif self.current_shape == "oval":
            self.current_shape_obj = Oval(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
        elif self.current_shape == "triangle":
            self.current_shape_obj = Triangle(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
        elif self.current_shape == "text":
            self.current_shape_obj = Text(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
        elif self.current_shape == "picture":  # Change
            self.current_shape_obj = Picture(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)  # Change
. . .
    def select_shape(self, x, y):
        self.unselect_all_shapes()
        for shape in self.canvas.shape_list:
            if isinstance(shape, Text) or isinstance(shape, Picture):  # Change
                if shape.bbox[0] <= x <= shape.bbox[2] and shape.bbox[1] <= y
<= shape.bbox[3]:
                    shape.is_selected = True
                    self.selected_obj = shape
                    self.canvas.draw_shapes()
            elif shape.x1 <= x <= shape.x2 and shape.y1 <= y <= shape.y2:
                shape.is_selected = True
                self.selected_obj = shape
                self.canvas.draw_shapes()
```

Run the program and create two images on the screen. Verify that you can select, move, and rotate one of the images. Select the other image, right click on the image to bring up a file dialog to load a new image.



Run the program again and create a diagram with one image, one rectangle, an one line draw from a connector on the image to a connector on the rectangle. Verify that if you

move the image or rectangle, the line is resized automatically.



Shape Appearance Menu Class

In the Line Editor project, we used simple buttons and option menu in the Top Frame Class for fill color, border color, and border width selection. In this project, we will create custom shape appearance frame with some nice features.

Objectives:

- Create a new class called Shape Appearance Frame derived from `ctk.CTkFrame`
- Add the `tktooltip` library using pip install `tkinter-tooltip` in a terminal

- Create a custom fill color control which has a color indicator rectangle to show which color was selected
- Create a custom border color control which has a color indicator rectangle to show which color was selected
- Create a custom border width control which shows a line width icon in a label next to an option menu with various border widths
- Add all 3 controls to the Shape Appearance Frame
- Add the Shape Appearance Frame to the Top Frame

Shape Appearance Frame Class

```
import customtkinter as ctk
from CTkColorPicker import *
from tktooltip import ToolTip
from PIL import Image


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

        self.fill_label = None
        self.border_label = None

        self.init_fill_color_control(self)
        self.init_border_color_control(self)
        self.init_border_width_control(self)

    def init_fill_color_control(self, shape_appearance_frame):
        # Fill color frame
        fill_frame = ctk.CTkFrame(shape_appearance_frame, width=150)
        fill_frame.pack(side=ctk.LEFT, padx=5, pady=5)

        # Add fill color picker
        def ask_fill_color():
            pick_color = AskColor()  # open the color picker
            color = pick_color.get()  # get the color string
            self.fill_label.configure(fg_color=color)
            for item in self.canvas.shape_list:
                if item.is_selected:
                    self.canvas.mouse.selected_obj.fill_color = color
```

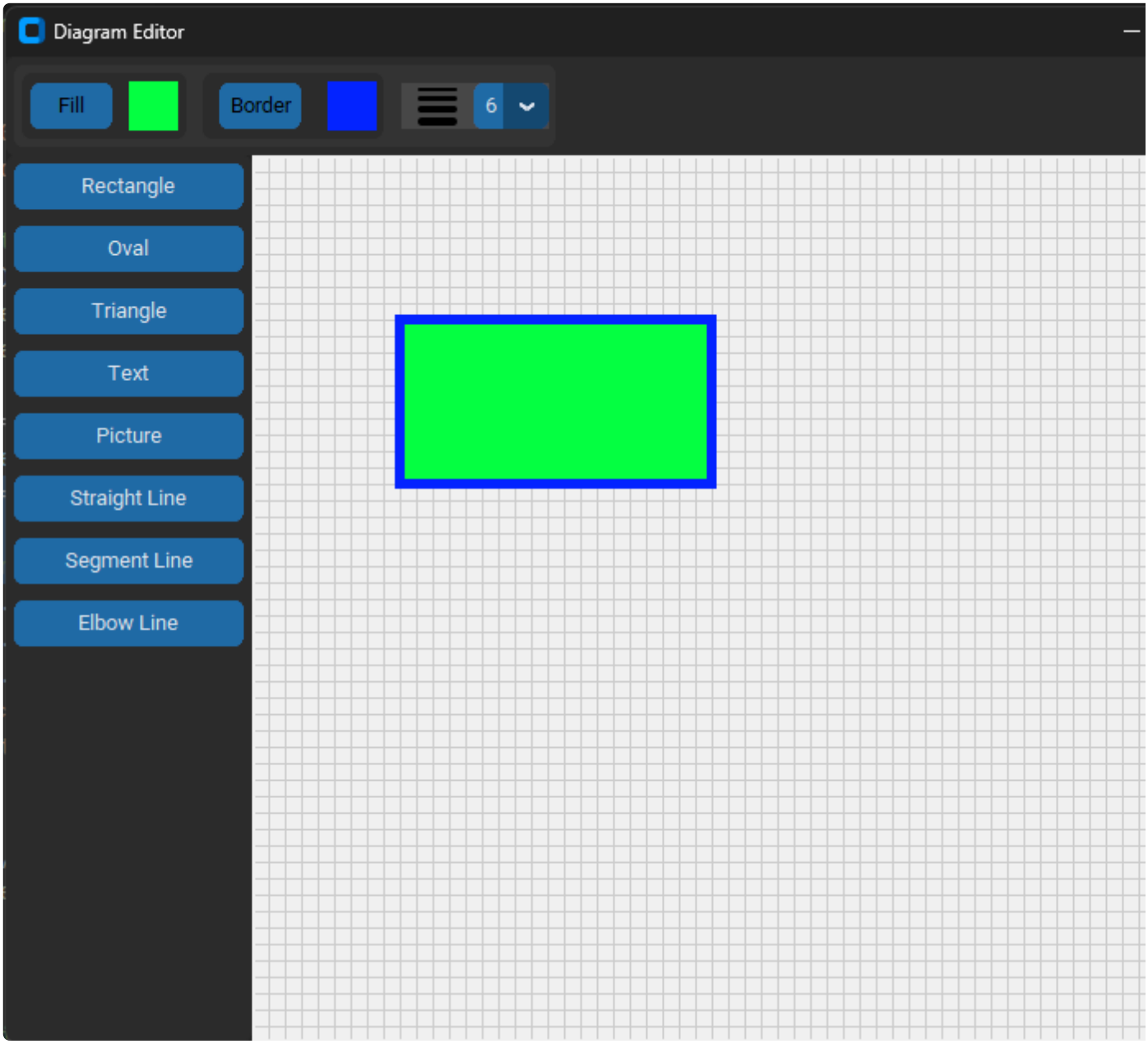
```
                    self.canvas.draw_shapes()

        fill_color_button = ctk.CTkButton(fill_frame, text="Fill",
text_color="black", width=50,
                                          command=ask_fill_color)
        fill_color_button.pack(side=ctk.LEFT, padx=5, pady=5)
        self.fill_label = ctk.CTkLabel(fill_frame, text="", width=30,
height=30, bg_color="white")
        self.fill_label.pack(side=ctk.LEFT, padx=5, pady=5)

        ToolTip(fill_color_button, msg="Fill color")

    def init_border_color_control(self, shape_appearance_frame):
        # Border color frame
        border_frame = ctk.CTkFrame(shape_appearance_frame, width=150)
        border_frame.pack(side=ctk.LEFT, padx=5, pady=5)

        # Add border (outline) color picker
        def ask_border_color():
            pick_color = AskColor()  # open the color picker
            color = pick_color.get()  # get the color string
            self.border_label.configure(fg_color=color)
            for item in self.canvas.shape_list:
                if item.is_selected:
                    self.canvas.mouse.selected_obj.border_color = color
                    self.canvas.draw_shapes()

        border_color_button = ctk.CTkButton(border_frame, text="Border",
text_color="black", width=50,
                                            command=ask_border_color)
        border_color_button.pack(side=ctk.LEFT, padx=10, pady=5)
        self.border_label = ctk.CTkLabel(border_frame, text="", width=30,
height=30, bg_color="white")
        self.border_label.pack(side=ctk.LEFT, padx=5, pady=5)

        ToolTip(border_color_button, msg="Border color")

    def init_border_width_control(self, shape_appearance_frame):
        # Border width frame
        width_frame = ctk.CTkFrame(shape_appearance_frame, width=150)
        width_frame.configure(fg_color=("gray28", "gray28"))  # set frame color
        width_frame.pack(side=ctk.LEFT, padx=5, pady=5)
        my_image = ctk.CTkImage(light_image=Image.open
                                ("D:/EETools/DiagramEditor/icons/line-
width.png"),
                                dark_image=Image.open
```

```
                                ("D:/EETools/DiagramEditor/icons/line-
width.png"),
                                size=(24, 24))

        image_label = ctk.CTkLabel(width_frame, image=my_image, text="",
corner_radius=10)
        image_label.pack(side=ctk.LEFT)

        # Add OptionMenu to top frame
        def option_menu_callback(choice):
            for item in self.canvas.shape_list:
                if item.is_selected:
                    self.canvas.mouse.selected_obj.border_width = choice
                    self.canvas.draw_shapes()

        option_menu = ctk.CTkOptionMenu(width_frame, values=["1", "2", "3",
"4", "5", "6", "7", "8", "9", "10"],
                                        width=32, command=option_menu_callback)
        option_menu.pack(side=ctk.LEFT)
        option_menu.set("3")

        ToolTip(option_menu, msg="Border width")
```

Top Frame Class

```
import customtkinter as ctk
from CTkColorPicker import *

from UI_Lib.shape_appearance_frame import ShapeAppearanceFrame


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

        self.shape_appearance_frame = ShapeAppearanceFrame(self, self.canvas)
        self.shape_appearance_frame.pack(side=ctk.LEFT, padx=5, pady=5)
```

Run the program, draw a rectangle, select the rectangle and verify that the fill color, border color, and border width can be changed. Hover the mouse over the appearance controls and confirm that a tooltip is displayed.



Snap Menu Class

Lets create a custom snap frame that gives the user control over the move snap size and resize snap size rather than using the grid size. The rotation is fixed at 90 degree increments for this project.

Objectives:

- Create a new class called Snap Frame derived from `ctk.CTkFrame`

- Create a grid size control which has a grid icon next to an option menu with one option: set all snaps to the grid size
- Create a draw snap control which has a draw shapes icon next to an option menu with various draw snap settings
- Create a move snap control which has a move icon next to an option menu with various move snap settings
- Create a resize snap control which has a resize icon next to an option menu with various resize snap settings
- Implement the grid, draw, move and resize snaps in the Grid Class.
- Add the 4 snap controls to the Snap Frame
- Add the Snap Frame to the Top Frame

Snap Frame Class

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


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

        self.init_grid_snap_control(self)
        self.init_draw_snap_control(self)
        self.init_move_snap_control(self)
        self.init_resize_snap_control(self)

    def init_grid_snap_control(self, snap_frame):
        move_frame = ctk.CTkFrame(snap_frame, width=150)
        move_frame.configure(fg_color=("gray28", "gray28"))  # set frame color
        move_frame.pack(side=ctk.LEFT, padx=5, pady=5)

        my_image = ctk.CTkImage(light_image=Image.open
                                ("D:/EETools/DiagramEditor/icons/grid.png"),
                                dark_image=Image.open
                                ("D:/EETools/DiagramEditor/icons/grid.png"),
                                size=(24, 24))

        image_label = ctk.CTkLabel(move_frame, image=my_image, text="",
```

```
corner_radius=10)
        image_label.pack(side=ctk.LEFT)

        # Add OptionMenu to top frame
        def option_menu_callback(choice):
            self.canvas.grid.grid_snap = self.canvas.grid.grid_size
            self.canvas.grid.draw_snap = self.canvas.grid.grid_size
            self.canvas.grid.move_snap = self.canvas.grid.grid_size
            self.canvas.grid.resize_snap = self.canvas.grid.grid_size

        option_menu = ctk.CTkOptionMenu(move_frame, values=["Grid Size"],
width=32,
                                        command=option_menu_callback)
        option_menu.pack(side=ctk.LEFT)
        option_menu.set("Grid Size")

        ToolTip(option_menu, msg="Set all snaps to Grid size")

    def init_draw_snap_control(self, snap_frame):
        move_frame = ctk.CTkFrame(snap_frame, width=150)
        move_frame.configure(fg_color=("gray28", "gray28"))  # set frame color
        move_frame.pack(side=ctk.LEFT, padx=5, pady=5)

        my_image = ctk.CTkImage(light_image=Image.open
                                ("D:/EETools/DiagramEditor/icons/draw.png"),
                                dark_image=Image.open
                                ("D:/EETools/DiagramEditor/icons/draw.png"),
                                size=(24, 24))

        image_label = ctk.CTkLabel(move_frame, image=my_image, text="",
corner_radius=10)
        image_label.pack(side=ctk.LEFT)

        # Add OptionMenu to top frame
        def option_menu_callback(choice):
            self.canvas.grid.draw_snap = int(choice)

        option_menu = ctk.CTkOptionMenu(move_frame, values=["5", "10", "20",
"30"], width=32,
                                        command=option_menu_callback)
        option_menu.pack(side=ctk.LEFT)
        option_menu.set("10")

        ToolTip(option_menu, msg="Draw snap")

    def init_move_snap_control(self, snap_frame):
```

```
        move_frame = ctk.CTkFrame(snap_frame, width=150)
        move_frame.configure(fg_color=("gray28", "gray28"))  # set frame color
        move_frame.pack(side=ctk.LEFT, padx=5, pady=5)

        my_image = ctk.CTkImage(light_image=Image.open
                                ("D:/EETools/DiagramEditor/icons/move.png"),
                                dark_image=Image.open
                                ("D:/EETools/DiagramEditor/icons/move.png"),
                                size=(24, 24))

        image_label = ctk.CTkLabel(move_frame, image=my_image, text="",
corner_radius=10)
        image_label.pack(side=ctk.LEFT)

        # Add OptionMenu to top frame
        def option_menu_callback(choice):
            self.canvas.grid.move_snap = int(choice)

        option_menu = ctk.CTkOptionMenu(move_frame, values=["5", "10", "20",
"30"], width=32,
                                        command=option_menu_callback)
        option_menu.pack(side=ctk.LEFT)
        option_menu.set("10")

        ToolTip(option_menu, msg="Move snap")

    def init_resize_snap_control(self, snap_frame):
        resize_frame = ctk.CTkFrame(snap_frame, width=150)
        resize_frame.configure(fg_color=("gray28", "gray28"))  # set frame
color
        resize_frame.pack(side=ctk.LEFT, padx=5, pady=5)

        my_image = ctk.CTkImage(light_image=Image.open
                                ("D:/EETools/DiagramEditor/icons/resize.png"),
                                dark_image=Image.open
                                ("D:/EETools/DiagramEditor/icons/resize.png"),
                                size=(24, 24))

        image_label = ctk.CTkLabel(resize_frame, image=my_image, text="",
corner_radius=10)
        image_label.pack(side=ctk.LEFT)

        # Add OptionMenu to top frame
        def option_menu_callback(choice):
            self.canvas.grid.resize_snap = int(choice)
```

```
        option_menu = ctk.CTkOptionMenu(resize_frame, values=["5", "10", "20",
"30"], width=32,
                                        command=option_menu_callback)
        option_menu.pack(side=ctk.LEFT)
        option_menu.set("10")

        ToolTip(option_menu, msg="Resize snap")
```

Top Frame Class

```
import customtkinter as ctk
from CTkColorPicker import *

from UI_Lib.shape_appearance_frame import ShapeAppearanceFrame
from UI_Lib.snap_frame import SnapFrame


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

        self.shape_appearance_frame = ShapeAppearanceFrame(self, self.canvas)
        self.shape_appearance_frame.pack(side=ctk.LEFT, padx=5, pady=5)

        self.snap_frame = SnapFrame(self, self.canvas)
        self.snap_frame.pack(side=ctk.LEFT, padx=5, pady=5)
```

Grid Class

```
class Grid:
    type = "line"

    def __init__(self, canvas, grid_size):
        self.canvas = canvas
        self.grid_size = grid_size
        self.grid_visible = True
        self.dash_list = None

        self.grid_snap = self.grid_size
```

```
        self.draw_snap = 10
        self.move_snap = 10
        self.resize_snap = 10

    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

            if Grid.type == "dot":
                self.dash_list = [1, 1]
            elif Grid.type == "line":
                self.dash_list = None

            # 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)], dash=self.dash_list,
                        fill='#cccccc', tag='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)], dash=self.dash_list,
                        fill='#cccccc', tag='grid_line')

    def snap_to_grid(self, x, y, op):  # where op = "grid" or "draw" or "move"
or "resize"
        if self.grid_visible:
            if op == "grid":
                x = round(x / self.grid_size) * self.grid_size
                y = round(y / self.grid_size) * self.grid_size
            elif op == "draw":
                x = round(x / self.draw_snap) * self.draw_snap
                y = round(y / self.draw_snap) * self.draw_snap
            elif op == "move":
                x = round(x / self.move_snap) * self.move_snap
                y = round(y / self.move_snap) * self.move_snap
            elif op == "resize":
                x = round(x / self.resize_snap) * self.resize_snap
                y = round(y / self.resize_snap) * self.resize_snap
        return x, y
```

We need to modify all shape classes in the resize() method to add the op value of "resize" to the snap to grid method call.
Rectangle Class, Oval Class, Triangle Class modify the resize() method.

```
    def resize(self, offsets, event):
        offset_x1, offset_y1, offset_x2, offset_y2 = offsets
        if self.selector == "s1":
            x1 = event.x - offset_x1
            y1 = event.y - offset_y1
            x1, y1 = self.canvas.grid.snap_to_grid(x1, y1, "resize")
            self.x1, self.y1 = x1, y1
        elif self.selector == "s2":
            x2 = event.x - offset_x2
            y1 = event.y - offset_y1
            x2, y1 = self.canvas.grid.snap_to_grid(x2, y1, "resize")
            self.x2, self.y1 = x2, y1
        elif self.selector == "s3":
            x2 = event.x - offset_x2
            y2 = event.y - offset_y2
            x2, y2 = self.canvas.grid.snap_to_grid(x2, y2, "resize")
            self.x2, self.y2 = x2, y2
        elif self.selector == "s4":
            x1 = event.x - offset_x1
            y2 = event.y - offset_y2
            x1, y2 = self.canvas.grid.snap_to_grid(x1, y2, "resize")
            self.x1, self.y2 = x1, y2
```

Line Class, Segment Class, Elbow Class modify the resize method()

```
    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
            x2, y2 = self.canvas.grid.snap_to_grid(x2, y2, "resize")
            self.x2, self.y2 = x2, y2
            self.canvas.mouse.select_connector(self, "end", x2, y2)
        elif self.selector == "begin":
            x1 = event.x - offset_x1
            y1 = event.y - offset_y1
            x1, y1 = self.canvas.grid.snap_to_grid(x1, y1, "resize")
            self.x1, self.y1 = x1, y1
            self.canvas.mouse.select_connector(self, "begin", x1, y1)
```

Mouse Class modify draw, move, and resize method

```
        def draw_left_down(self, event):
            self.unselect_all_shapes()
            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, "draw")  # Change

            if self.current_shape == "rectangle":
                self.current_shape_obj = Rectangle(self.canvas, self.start.x,
self.start.y, self.start.x, self.start.y)
```

```
    def draw_left_drag(self, event):
        if self.current_shape_obj:
            x, y = event.x, event.y
            x, y = self.canvas.grid.snap_to_grid(x, y, "draw")  # Change
            self.current_shape_obj.x1, self.current_shape_obj.y1 =
self.start.x, self.start.y
            self.current_shape_obj.x2, self.current_shape_obj.y2 = x, y
            self.canvas.draw_shapes()
```

```
    def move_left_down(self, event):
        if self.selected_obj:
            x, y = event.x, event.y
            sel = self.selected_obj.check_selector_hit(x, y)
            if sel:
                self.selected_obj.selector = sel.name
                self.unbind_mouse_events()
                self.bind_resize_mouse_events()
                self.resize_left_down(event)
                return
            else:
                a_shape = self.selected_obj
                a_shape.is_selected = False
                self.selected_obj = None
                self.canvas.draw_shapes()

        x, y = event.x, event.y
        self.select_shape(x, y)
        if self.selected_obj:
            if not isinstance(self.selected_obj, Line):
                self.mode = None
            x1, y1 = self.selected_obj.x1, self.selected_obj.y1
            x1, y1 = self.canvas.grid.snap_to_grid(x1, y1, "move")  # Change
            x2, y2 = self.selected_obj.x2, self.selected_obj.y2
```

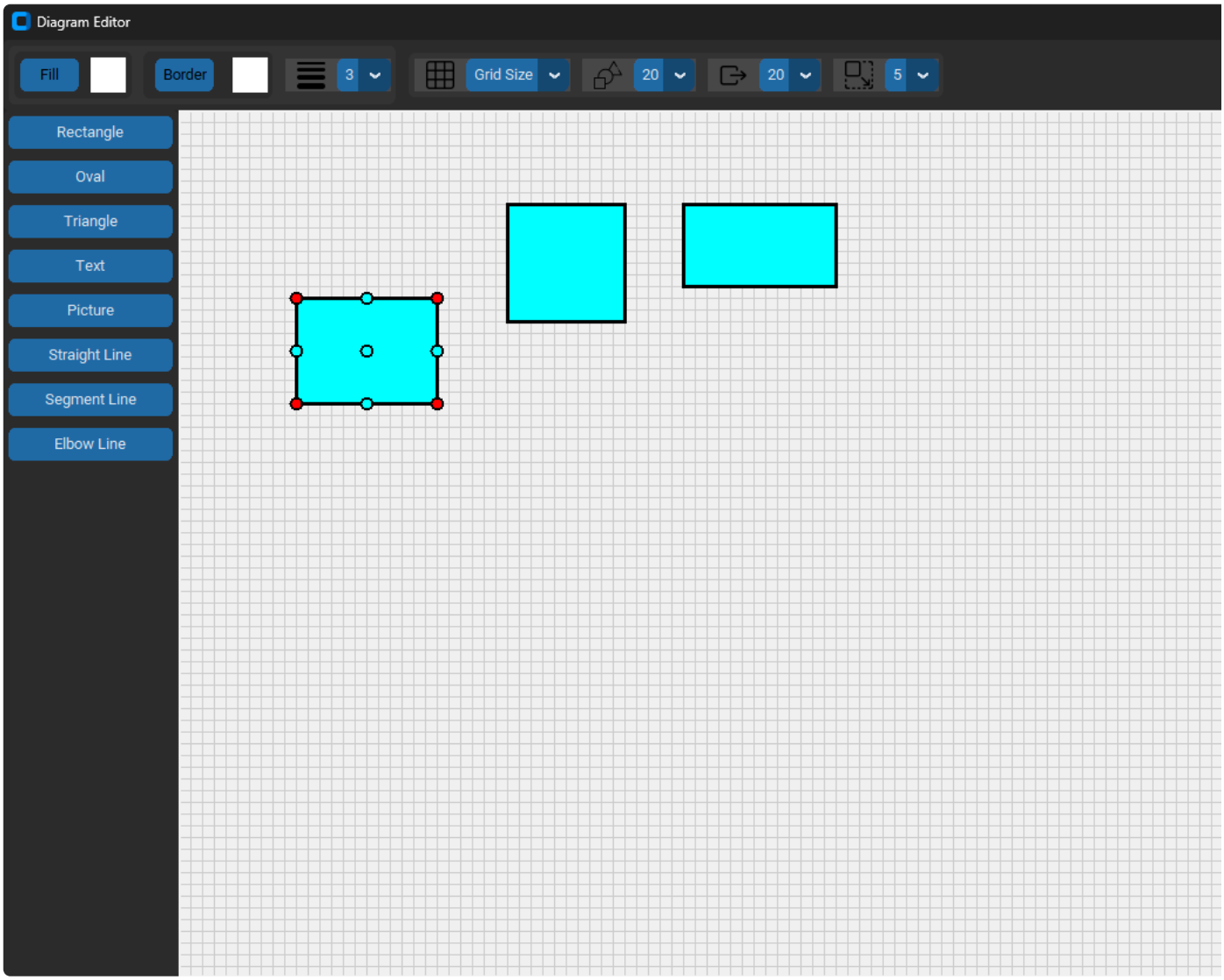
```
            x2, y2 = self.canvas.grid.snap_to_grid(x2, y2, "move")  # Change
            self.offset1.x = x - x1
            self.offset1.y = y - y1
            self.offset2.x = x - x2
            self.offset2.y = y - y2
            self.canvas.draw_shapes()
```

```
    def move_left_drag(self, event):
        if self.selected_obj:
            x = event.x - self.offset1.x
            y = event.y - self.offset1.y
            x, y = self.canvas.grid.snap_to_grid(x, y, "move")  # Change
            self.selected_obj.x1, self.selected_obj.y1 = x, y
            x = event.x - self.offset2.x
            y = event.y - self.offset2.y
            x, y = self.canvas.grid.snap_to_grid(x, y, "move")  # Change
            self.selected_obj.x2, self.selected_obj.y2 = x, y
            self.canvas.draw_shapes()
```

```
    def resize_left_down(self, event):
        if self.selected_obj:
            if not isinstance(self.selected_obj, Line):
                self.mode = None
            x1, y1 = self.selected_obj.x1, self.selected_obj.y1
            x1, y1 = self.canvas.grid.snap_to_grid(x1, y1, "resize")  # Change
            x2, y2 = self.selected_obj.x2, self.selected_obj.y2
            x2, y2 = self.canvas.grid.snap_to_grid(x2, y2, "resize")  # Change
            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.canvas.draw_shapes()
```

Run the program and draw a few shapes. Experiment with changing the draw snap, move snap, and resize snap to confirm that the snap changes based on the selected operation. Set the snaps back to the grid size. This gives the user finer control over the various snaps

if desired. This feature also demonstrates how to show an icon next to option menus.



File Menu Class

This is essentially a file menu designed as a modern ctk frame. The Top Frame button is a ctk button with an image created from a hamburger icon. When selected, a menu frame is shown with buttons for New, Open, Save, Exit functions. The new file option create a new diagram by clearing the canvas. The open file option opens a file dialog and allows the user to select a `json` file to be loaded. The save file option opens a file dialog and allows the user to save a `json` diagram file. The exit option closes the application. This is a new feature not included in the Shape Editor and Line Editor projects.

Objectives:

- Create a custom File Menu Frame Class
- Create a button to show a drop down menu frame below the button

- Add buttons to the menu frame for New, Open, Save, Exit
- Implement the menu frame options using `json` for file save and open file formats
- Add the File Menu Frame to the Top Frame on the left side

File Menu Class

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

from Shape_Lib.rectangle import Rectangle
from Shape_Lib.oval import Oval
from Shape_Lib.triangle import Triangle
from Shape_Lib.text import Text
from Shape_Lib.line import Line
from Shape_Lib.picture import Picture


class FileMenuFrame(ctk.CTkFrame):
    def __init__(self, window, parent, canvas):
        super().__init__(parent)
        self.canvas = canvas
        self.obj_type_dict = {'rectangle': Rectangle,
                              'oval': Oval,
                              'triangle': Triangle,
                              'text': Text,
                              'line': Line,
                              'pic': Picture}

        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

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

("D:/EETools/DiagramEditor/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.save_diagram()
        self.canvas.delete("all")
        self.canvas.shape_list = []

    def load_diagram(self):
        try:
            filetypes = (('json files', '*.json'), ('All files', '*.*'))
            f = fd.askopenfilename(filetypes=filetypes, initialdir="./")
            with open(f) as file:
                obj_dict = json.load(file)
                for obj_type, attributes in obj_dict.items():
                    obj = self.obj_type_dict[obj_type.split()[0]](self.canvas,
attributes[0], attributes[1],

attributes[2], attributes[3])
                    self.canvas.shape_list.append(obj)
                self.canvas.draw_shapes()

        except FileNotFoundError:
            with open('untitled.canvas', 'w') as file:
                pass
            self.canvas.shape_list = []

    def save_diagram(self):
        filetypes = (('json files', '*.json'), ('All files', '*.*'))
        f = fd.asksaveasfilename(filetypes=filetypes, initialdir="./")
        with open(f, 'w') as file:
```

```
            obj_dict = {f'{obj.type} {id}': (obj.x1, obj.y1, obj.x2, obj.y2)
for id, obj in
                        enumerate(self.canvas.shape_list)}
            json.dump(obj_dict, file)

    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
```

Top Frame Class

```
import customtkinter as ctk
from CTkColorPicker import *

from UI_Lib.file_menu_frame import FileMenuFrame
from UI_Lib.shape_appearance_frame import ShapeAppearanceFrame
from UI_Lib.snap_frame import SnapFrame


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

        file_menu_frame = FileMenuFrame(parent, self, self.canvas)
        file_menu_frame.pack(side=ctk.LEFT, padx=5, pady=5)

        shape_appearance_frame = ShapeAppearanceFrame(self, self.canvas)
        shape_appearance_frame.pack(side=ctk.LEFT, padx=5, pady=5)

        snap_frame = SnapFrame(self, self.canvas)
        snap_frame.pack(side=ctk.LEFT, padx=5, pady=5)
```

Run the program, add shapes to the diagram, save the diagram as a `json` file, select the New menu option, save the file again as a `json` file, verify that the canvas is cleared, open the `json` file and verify that the diagram is restored. Select the Exit option to close the

application.

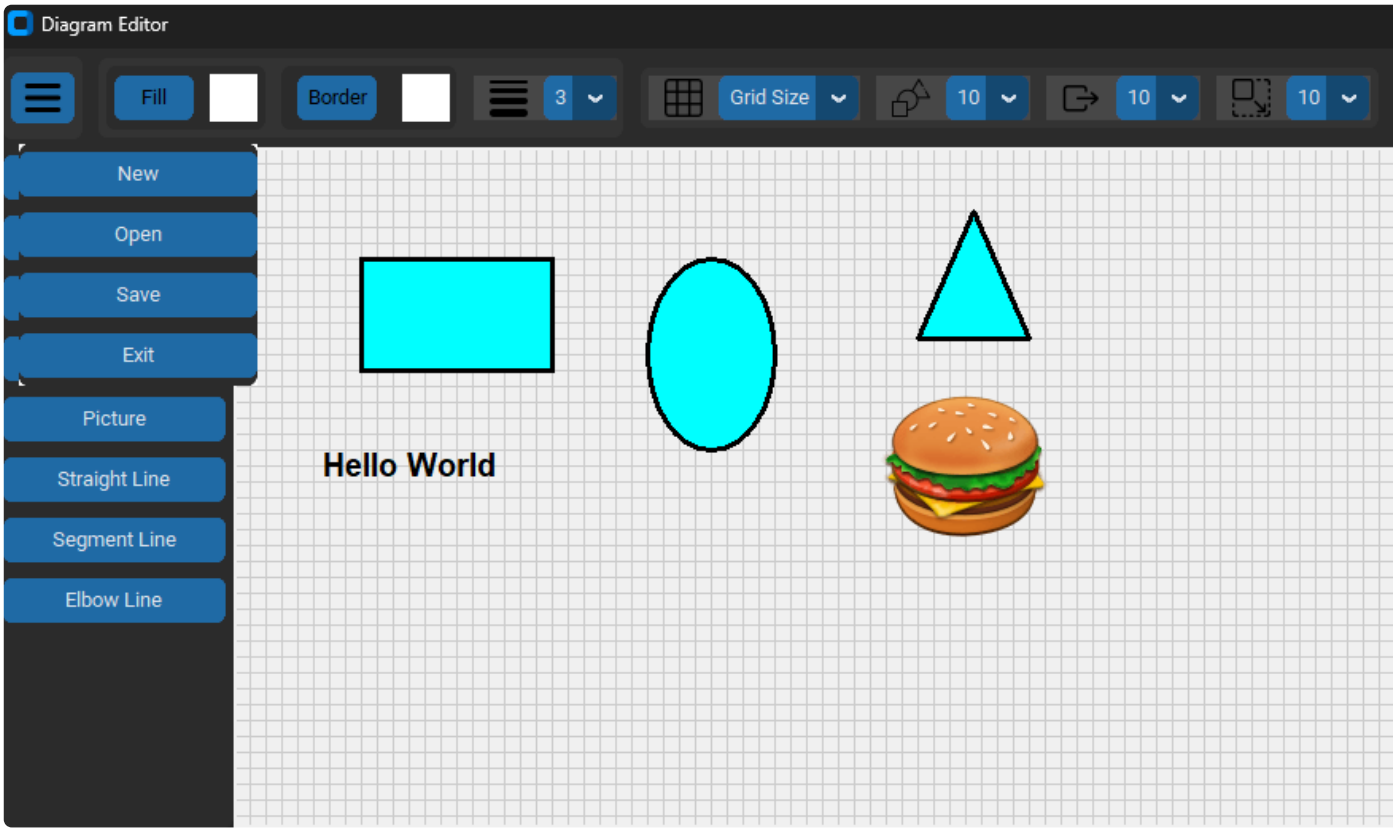


diagram1.json

```
{"rectangle 0": [80, 70, 200, 140], "oval 1": [260, 70, 340, 190], "triangle 2": [430, 40, 500, 120], "text 3": [110, 200, 110, 200], "picture 4": [460, 200, 460, 200]}
```

Settings Menu Class

Objectives:

- Application light/dark appearance mode
- Grid visibility control
- Grid size control

diagram_editor.py

```
import customtkinter as ctk
from UI_Lib import *
```

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


class DiagramEditorApp(ctk.CTk):
```

Settings Frame Class

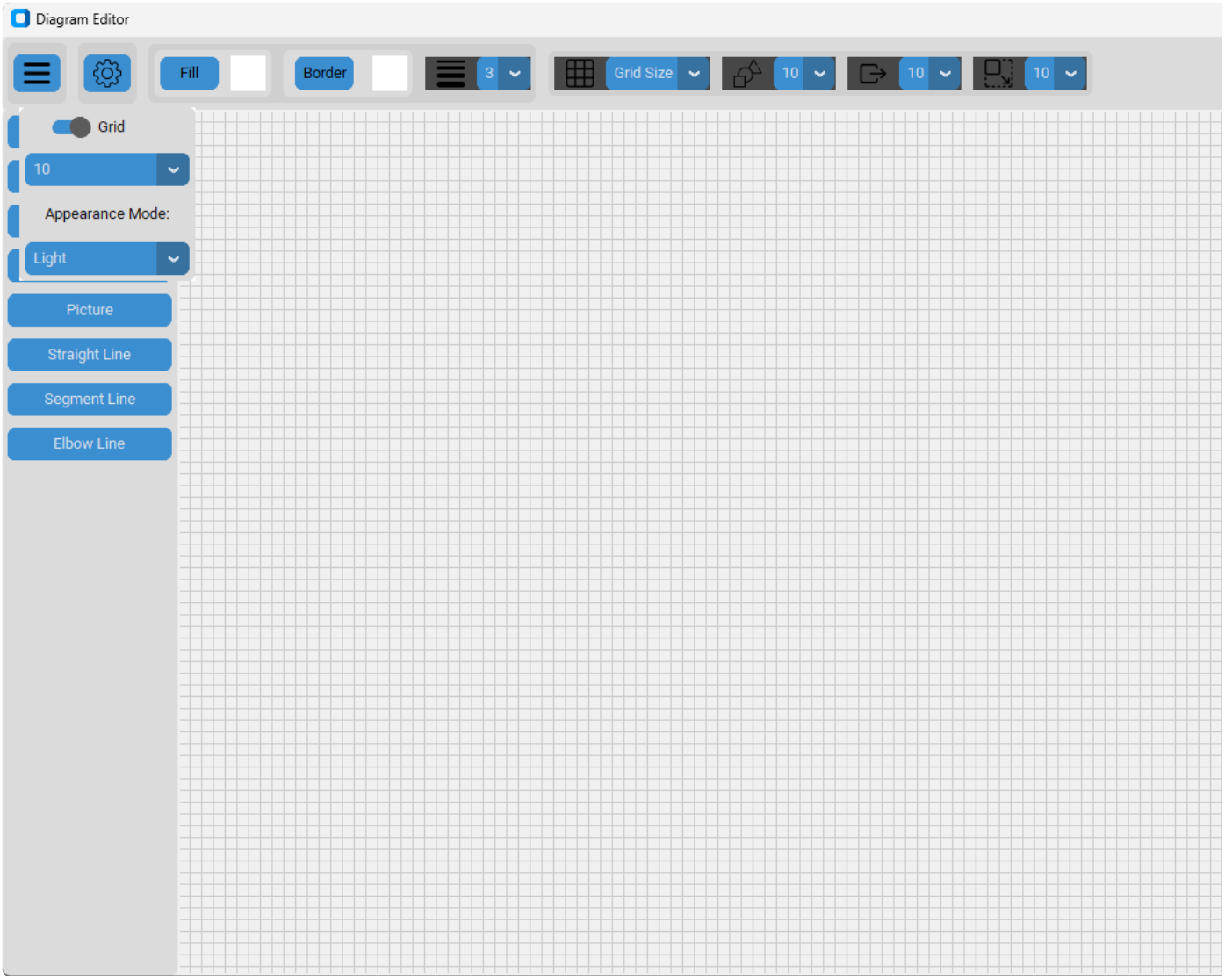
```
import customtkinter as ctk
from PIL import Image


class SettingsFrame(ctk.CTkFrame):
    def __init__(self, window, parent, canvas):
        super().__init__(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:
                canvas.grid.grid_visible = False
                self.canvas.mouse.move_snap = 1
                self.canvas.mouse.resize_snap = 1
                self.canvas.mouse.rotate_snap = 90
            else:
                canvas.grid.grid_visible = True
                self.canvas.mouse.move_snap = 10
                self.canvas.mouse.resize_snap = 10
                self.canvas.mouse.rotate_snap = 90
            self.canvas.draw_shapes()

        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)
```

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

        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")

        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

    def change_appearance_mode_event(self, new_appearance_mode: str):
        ctk.set_appearance_mode(new_appearance_mode)
```

Top Frame Class

```
import customtkinter as ctk

from UI_Lib.file_menu_frame import FileMenuFrame
from UI_Lib.shape_appearance_frame import ShapeAppearanceFrame
from UI_Lib.snap_frame import SnapFrame
from UI_Lib.settings_frame import SettingsFrame  # Change


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

        file_menu_frame = FileMenuFrame(parent, self, self.canvas)
        file_menu_frame.pack(side=ctk.LEFT, padx=5, pady=5)

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

        shape_appearance_frame = ShapeAppearanceFrame(self, self.canvas)
        shape_appearance_frame.pack(side=ctk.LEFT, padx=5, pady=5)

        snap_frame = SnapFrame(self, self.canvas)  # Change
        snap_frame.pack(side=ctk.LEFT, padx=5, pady=5)  # Change
```

This image is the application with the appearance mode set to light mode.



Help Menu Class

Objectives:

- Add a help menu to the top frame
- Add an about option to the help menu

Help Frame Class

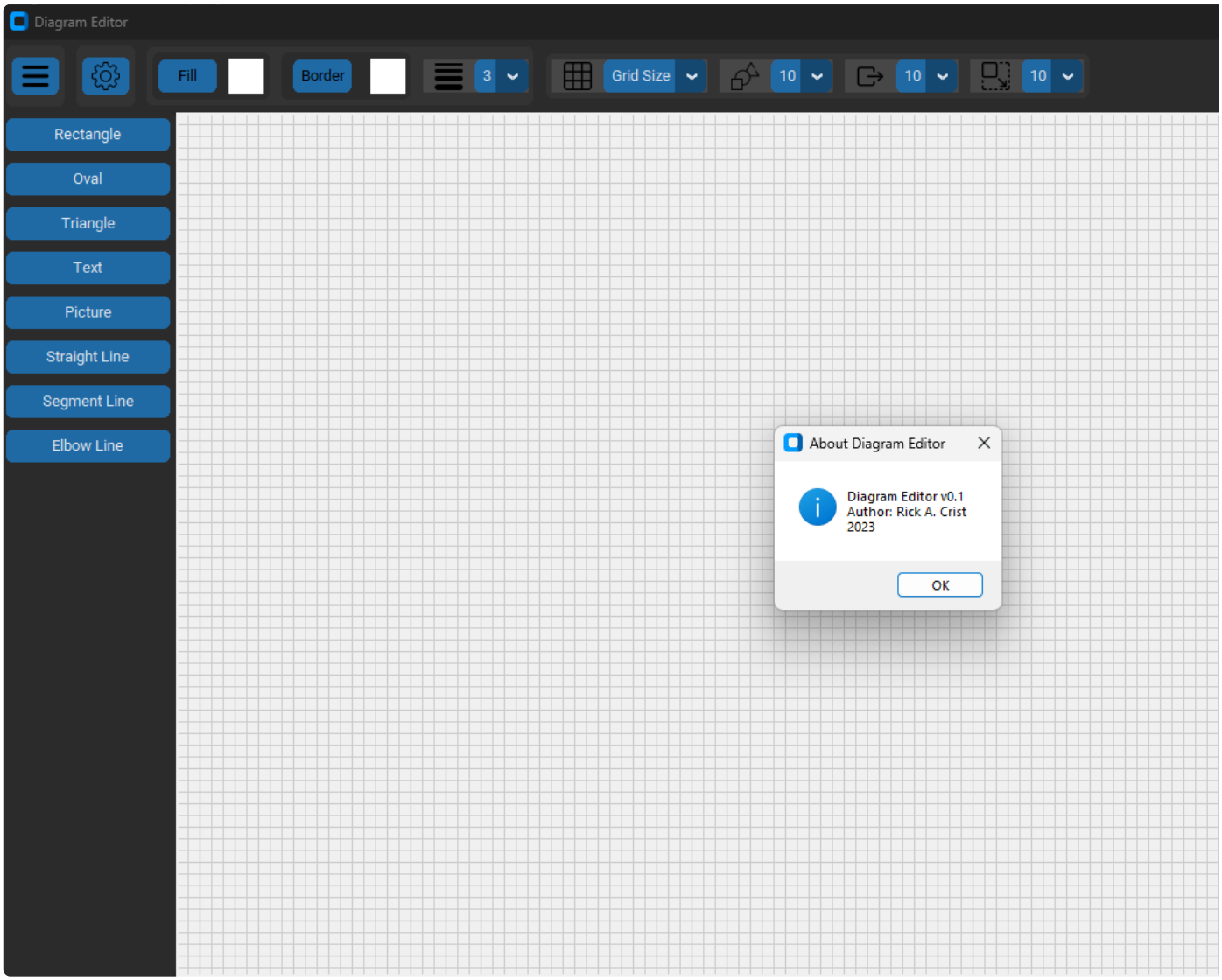
```
import customtkinter as ctk
from tkinter import messagebox
from PIL import Image
```

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

        self.menu_on = False

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

        about_button = ctk.CTkButton(self.menu_frame, text="About",
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/DiagramEditor/icons/help.png"),
                                dark_image=Image.open
                                ("D:/EETools/DiagramEditor/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, y=60)
            self.menu_frame.tkraise()
            self.menu_on = True
        else:
            self.menu_frame.place_forget()
            self.menu_on = False

    def show_about_dialog(self):
        messagebox.showinfo("About Diagram Editor", "Diagram Editor v0.1\n" +
"Author: Rick A. Crist\n" + "2023")
```

Top Frame Class

```
import customtkinter as ctk

from UI_Lib.file_menu_frame import FileMenuFrame
```

```python
from UI_Lib.shape_appearance_frame import ShapeAppearanceFrame
from UI_Lib.snap_frame import SnapFrame
from UI_Lib.settings_frame import SettingsFrame
from UI_Lib.help_frame import HelpFrame  # Change


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

        file_menu_frame = FileMenuFrame(parent, self, self.canvas)
        file_menu_frame.pack(side=ctk.LEFT, padx=5, pady=5)

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

        shape_appearance_frame = ShapeAppearanceFrame(self, self.canvas)
        shape_appearance_frame.pack(side=ctk.LEFT, padx=5, pady=5)

        snap_frame = SnapFrame(self, self.canvas)
        snap_frame.pack(side=ctk.LEFT, padx=5, pady=5)

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

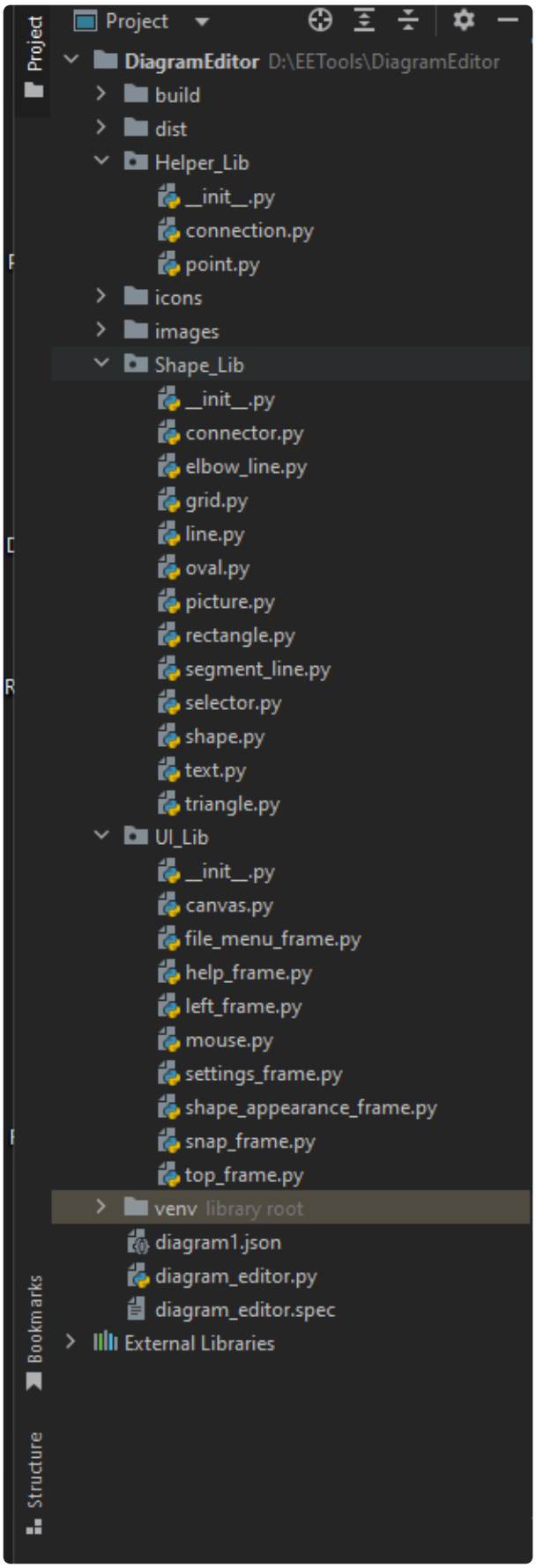


Application Deployment

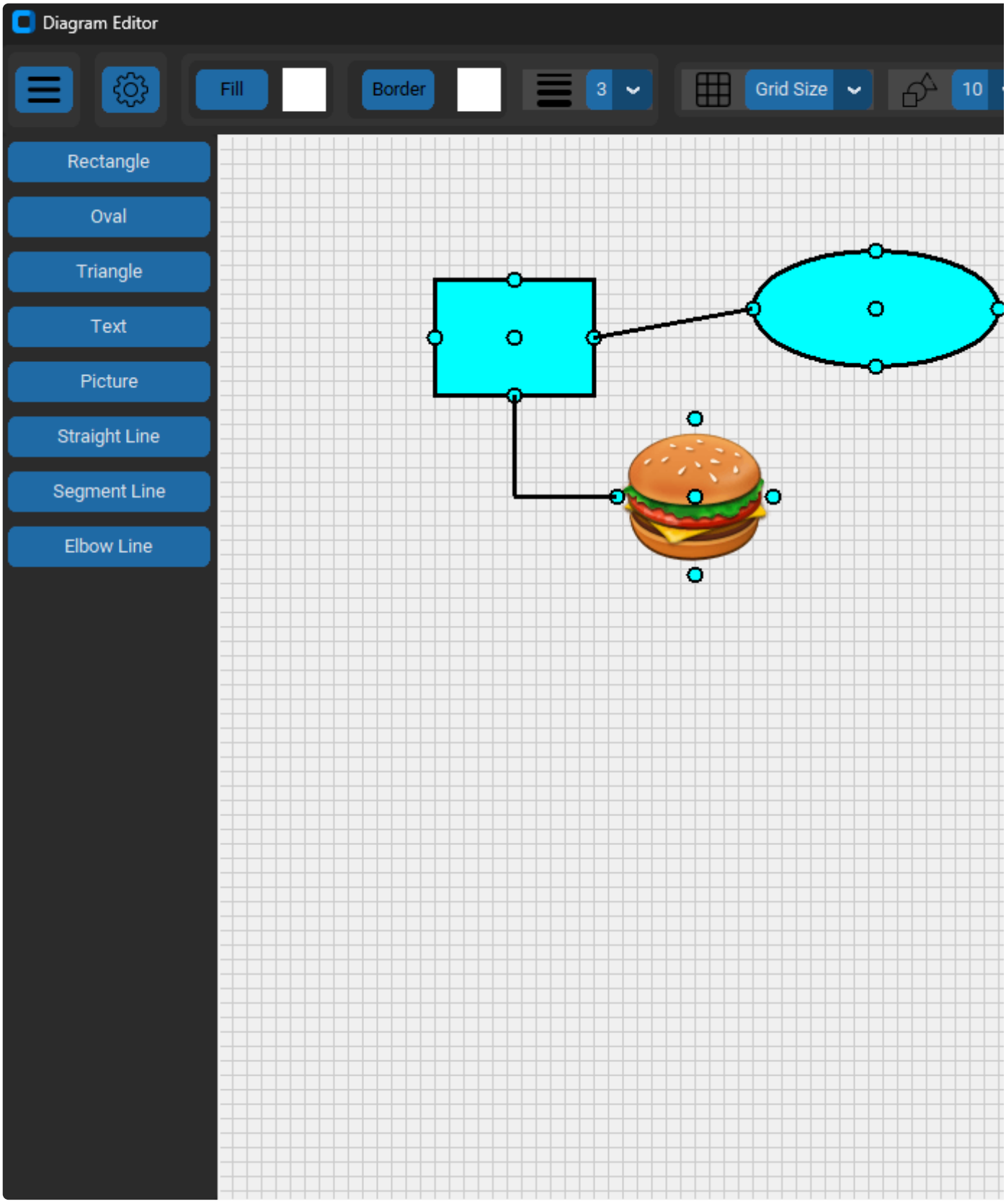
Open the terminal in `D:\EETools\DiagramEditor` directory

pyinstaller diagram_editor.py

.exe is created in the dist directory



Copy dist directory to My Applications folder and run diagram_editor.exe



Summary

This concludes the development of the Diagram Editor. The Diagram Editor is the basis for the advanced electrical engineering projects. You can model a circuit simulator as a

Diagram Editor with a simulation capability. The next project is a Digital Circuit Simulator so lets proceed.