# Strand Drawing Tool Documentation

**By Yonatan Setbon**

## Overview

The Strand Drawing Tool is a PyQt5-based application that enables users to create and manipulate interconnected strands on a canvas. It offers two primary modes: Attach Mode for creating and connecting strands, and Move Mode for adjusting existing strands. The tool utilizes object-oriented programming principles to manage complex strand structures and interactions.

## File Structure and Detailed Functionality

### main.py

Purpose: main.py is the entry point of the application, setting up the main window, canvas, and mode switching functionality. It orchestrates the overall flow of the application.

#### StrandDrawingCanvas class

* - Inherits from QWidget to create a custom drawing area.
* - Manages a list of Strand objects (self.strands).
* - Implements paintEvent method to render strands and attached strands recursively.
* - Handles mouse events (mousePressEvent, mouseMoveEvent, mouseReleaseEvent) and delegates them to the current mode.
* - Provides methods for strand management: add\_strand(strand), remove\_strand(strand), clear\_strands().
* - Implements get\_strand\_at\_position(pos) and get\_attached\_strand\_at\_position(attached\_strands, pos) for hit detection.
* - Allows customization of strand properties through methods like set\_strand\_width(), set\_strand\_color(), etc.

#### MainWindow class

* - Inherits from QMainWindow to create the main application window.
* - Sets up the UI layout with a QVBoxLayout containing mode switching buttons and the canvas.
* - Implements set\_attach\_mode() and set\_move\_mode() to switch between modes.

#### Functionality

* - Initializes the PyQt5 application and sets up the main window.
* - Creates instances of AttachMode and MoveMode, switching between them based on user input.
* - Manages the rendering loop, calling update() to refresh the canvas when strands are modified.

### attach\_mode.py

Purpose: attach\_mode.py contains the logic for creating new strands and attaching them to existing ones. It defines the structure and behavior of strands.

#### Strand class

* - Represents a basic strand with start and end points (QPointF objects).
* - Manages strand properties: width, color, stroke\_color, stroke\_width.
* - Maintains a list of attached\_strands for complex structures.
* - Implements update\_shape() to calculate the strand's corners based on its width and orientation.
* - Provides get\_path() method to create a QPainterPath for rendering.
* - Implements draw(painter) method to render the strand and its attached strands.

#### AttachedStrand class (inherits from Strand)

* - Represents a strand attached to another strand.
* - Adds properties like angle and length to manage its position relative to the parent strand.
* - Implements update(mouse\_pos) to adjust the strand based on mouse movement.
* - Overrides draw(painter) to include a side line indicating the attachment point.

#### AttachMode class

* - Manages the state of strand creation and attachment.
* - Implements mouse event handlers: mousePressEvent(event), mouseMoveEvent(event), mouseReleaseEvent(event).
* - Provides methods like update\_first\_strand(end\_pos) and handle\_strand\_attachment(pos) to manage the attachment process.

#### Functionality

* - Implements the logic for creating the initial strand on an empty canvas.
* - Manages the process of attaching new strands to existing ones, including collision detection and snapping.
* - Handles the drawing and updating of strands during the creation and attachment process.
* - Implements angle snapping (to 5-degree increments) for smoother strand placement.

### move\_mode.py

Purpose: move\_mode.py contains the logic for moving and adjusting existing strands on the canvas.

#### MoveMode class

* - Manages the state of strand movement.
* - Implements mouse event handlers: mousePressEvent(event), mouseMoveEvent(event), mouseReleaseEvent(event).
* - Provides methods like handle\_strand\_movement(pos) and try\_move\_strand(strand, pos) to manage the movement process.
* - Implements update\_strand\_position(new\_pos) to calculate and apply the new position of the moved strand.
* - Manages the updating of attached strands through update\_attached\_strands(strand, old\_start, old\_end).

#### Functionality

* - Allows users to click and drag strand endpoints to new positions.
* - Implements a selection mechanism using get\_end\_rectangle(strand, side) to create clickable areas at strand endpoints.
* - Maintains the connections between attached strands during movement, updating their positions recursively.
* - Implements a snapping mechanism to round strand angles (to 10-degree increments) for neater positioning.
* - Updates the visual representation of strands as they are moved, including the position of the selection rectangle.

## User Interaction Flow

### Application Start

* - The main window is created, and the canvas is initialized in Attach Mode.

### Attach Mode

* - First Strand Creation:
* - User clicks and drags on the canvas.
* - AttachMode.mousePressEvent() initializes a new Strand.
* - AttachMode.mouseMoveEvent() updates the end position of the strand.
* - AttachMode.mouseReleaseEvent() finalizes the strand and adds it to the canvas.
* - Subsequent Strand Attachment:
* - User clicks near an existing strand's endpoint.
* - AttachMode.handle\_strand\_attachment() checks for valid attachment points.
* - If valid, a new AttachedStrand is created and linked to the existing strand.
* - Mouse movement updates the new strand's position and angle.
* - Mouse release finalizes the attachment.

### Move Mode

* - User clicks on a strand's endpoint (indicated by a red square).
* - MoveMode.try\_move\_strand() identifies the clicked strand and initializes movement.
* - As the user drags, MoveMode.update\_strand\_position() calculates the new position.
* - The strand and all its attached strands are updated recursively.
* - Mouse release finalizes the new positions.

### Mode Switching

* - User clicks the mode buttons in the main window.
* - MainWindow.set\_attach\_mode() or set\_move\_mode() is called.
* - The canvas's current mode is updated, changing how it responds to mouse events.

## Customization and Extension

The tool's modular design allows for easy customization and extension:

* - Strand appearance can be modified by adjusting properties in StrandDrawingCanvas.
* - New strand types can be created by subclassing Strand or AttachedStrand.
* - Additional modes can be implemented by creating new mode classes and integrating them into main.py.

## Conclusion

The Strand Drawing Tool demonstrates effective use of object-oriented design and event-driven programming in PyQt5. Its modular structure, split across three files, allows for clear separation of concerns and easy maintenance. The tool provides a flexible framework for creating and manipulating complex strand structures, making it adaptable for various diagramming and design tasks.