Assignment: A11

Air-Table: Rotating Tubes

This exercise introduces:

- Vector operation such as addition and rotation.
- Drawing of polygons through use of a vertices list.

Python language topics:

- Operator overloading.

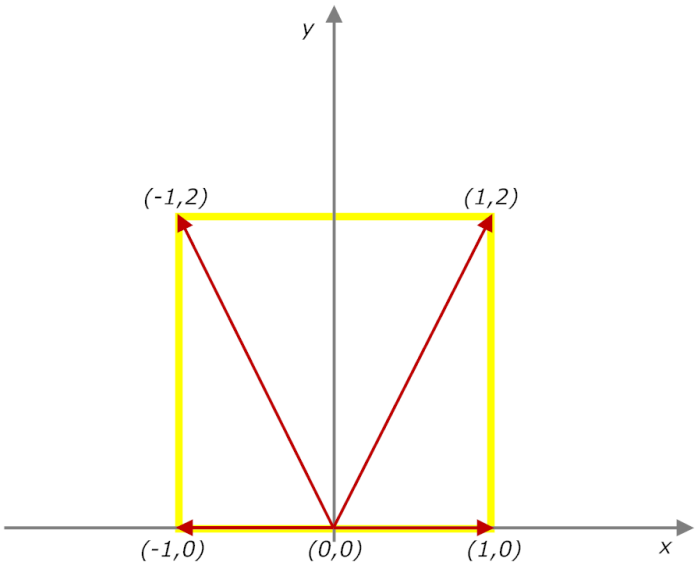
Problem statement:

(Again, start with a new Python file.)

Add algorithmic content to the baseline air-table file (A10_2D_baseline_server.py) to display and rotate a 2D tube (a long rectangle). Display this ONLY on the #7 demo. Rotate the tube around the center of one of its ends (its base-point). Use the "a" and "d" keys to rotate clockwise and counter clockwise. Use the Vec2D class to facilitate the vector rotations. Create at least two instances of the tubes. Locate their base-points at the following (x,y) locations: (1.0,1.0), (2.0,2.0), (3.0,3.0), etc. Each tube should rotate around its own base-point position.

Algorithmic description:

Create a RotatingTube class. Attributes of this class should include a 4-point vertices list that will represent the tube (yellow rectangle in drawing to the right). Each vertex will be represented by a 2D vector. The basic algorithmic flow of the class methods is in the comments of the code images below.



Python code:

The following code (images) is not a complete solution to the problem. It shows additional content relative to the baseline air-table file (A10_2D_baseline_server.py available on the classes share). There is some obfuscation this time, and you have to figure out where these pieces of code should go. The indent levels should be a clue to you. These are not necessarily in order, so of course the neighboring images are not necessarily a continuation from the image above.

You will also have to call some of these class methods from the "main" function. And of course you will have to instantiate the tubes somewhere. Both of these steps are not in the images below.

```
class RotatingTube:
    def __init__(self, tube_base_2d_m):
        self.color = THECOLORS["yellow"]

        # Degrees of rotation per rendering cycle.
        self.rotation_deg = 1.8

        # Scaling factors to manage the aspect ratio of the tube.
        self.sf_x = 0.15
        self.sf_y = 0.50

        self.tube_base_2d_m = tube_base_2d_m

        # Notice the counter-clockwise drawing pattern. Four vertices for a rectangle.
        # Each vertex is represented by a vector.
        self.tube_vertices_2d_m = [Vec2D(-0.50 * self.sf_x, 0.00 * [illegible]),
                                   Vec2D( 0.50 * self.sf_x, 0.00 * [illegible]),
                                   Vec2D( 0.50 * self.sf_x, 1.00 * [illegible]),
                                   Vec2D(-0.50 * self.sf_x, 1.00 * [illegible])]

        # Define a normal (1 meter) pointing vector to keep track of the direction of the jet.
        self.direction_2d_m = Vec2D(0.0, 1.0)
```

```
    def rotate_vertices(self, vertices_2d_m, angle_deg):
        # Put modified vectors in a new list.
        rotated_vertices_2d_m = []
        for vertex_2d_m in vertices_2d_m:
            rotated_vertices_2d_m.append( [illegible])
        return rotated_vertices_2d_m

    def rotate_everything(self, angle_deg):
        # Rotate the pointer.
        self.direction_2d_m = self.direction_2d_m.rotated( angle_deg)

        # Rotate the tube.
        self.tube_vertices_2d_m = [illegible]

    def client_rotation_control(self, client_name):
        # Rotate clockwise (D) and counter-clockwise (A).
        if (env.clients[client_name].key_a == "D"):
            [illegible]
            self.rotate_everything( -1 * self.rotation_deg)

    def convert_from_world_to_screen(self, vertices_2d_m, base_point_2d_m):
        vertices_2d_px = []
        for [illegible]
            # Calculate absolute position of this vertice.
            vertices_2d_px.[illegible]
        return vertices_2d_px

    def draw(self):
        # Draw the tube on the game-window surface.
        pygame.draw.[illegible],
                          [illegible], 3)
```