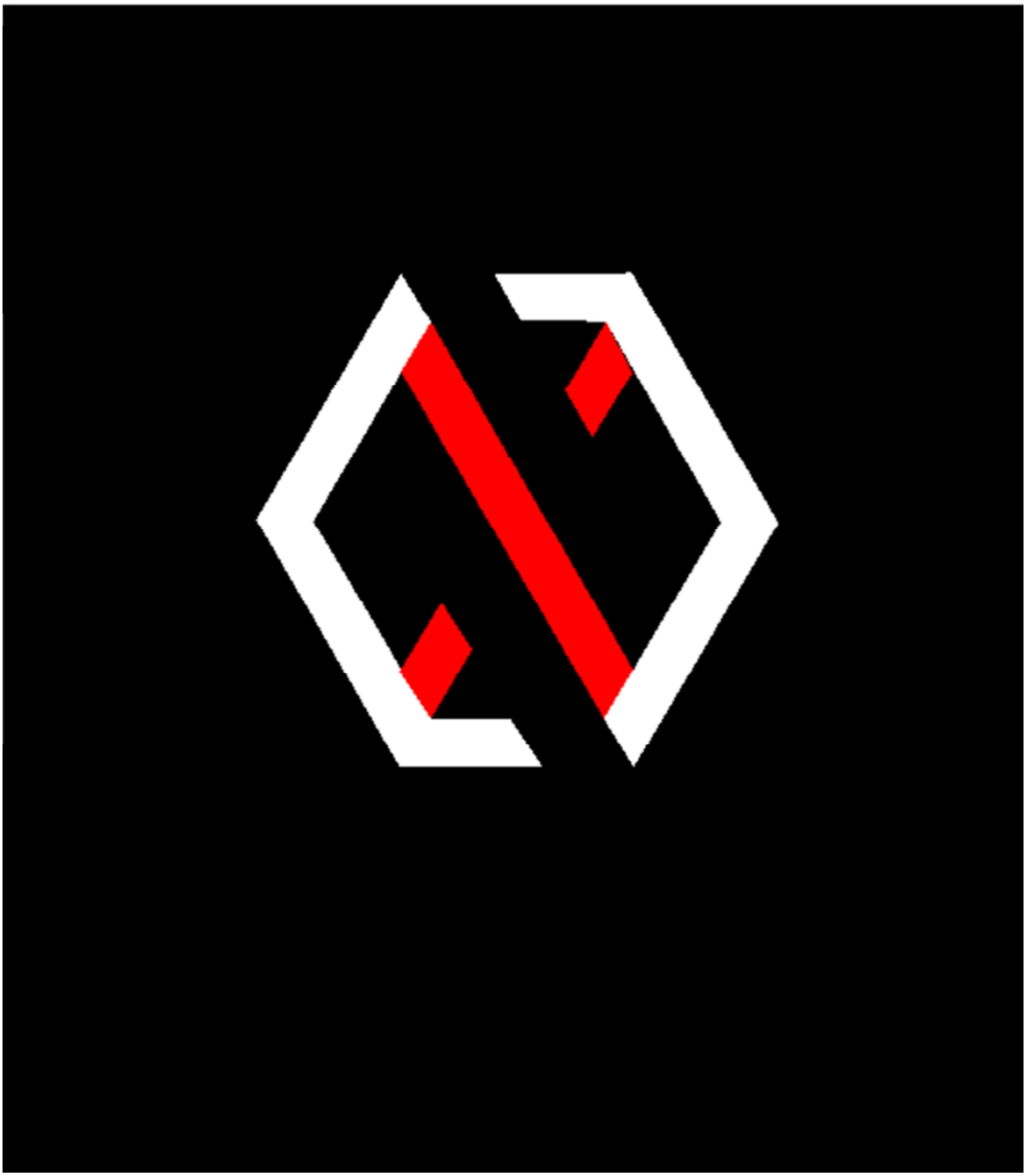
Nama : Ahmad Ilyas

NIM : 222410103049

Logo Xendit



```python
from GL_tools.GL_tools import *

def main():
    window = OpenGLInitializer(window_size=(800,800),window_title="Logo Xendit",kiri=0,kanan=9,atas=9,bawah=0)

    # Tameng 1
    window.object_manager.create_polygon([(2.5110828365283,1.6948025938735),(1.5996653302464,3.2438182723229),(1.959497413408,3.2544015688864),(2.7,2)])
    window.object_manager.create_polygon([(1.5996653302464,3.2438182723229),(2.5,4.8),(2.7,4.5),(1.959497413408,3.2544015688864)])
    window.object_manager.create_polygon([(2.5,4.8),(3.4,4.8),(3.2,4.5),(2.7,4.5),(2.7,4.5)])

    # X
    window.object_manager.create_polygon([(2.7,4.5),(2.9581243868851,4.0631863646461),(2.766121510799,3.7685755875341),(2.5,4.2)],color=colors['Red'])
    window.object_manager.create_polygon([(2.5098288347141,2.312488174728),(2.7,2),(3.9741603345122,4.1918285461149),(3.7875298444594,4.4982030235776)],color=colors['Red'])
    window.object_manager.create_polygon([(3.5437625328962,2.4330239999353),(3.7145537791955,2.7264346025521),(3.9685510172818,2.319163169069),(3.8,2)],color=colors['Red'])

    # Tameng 2
    window.object_manager.create_polygon([(3.0970777348824,1.6973078620303),(3.9641717545561,1.6929285993046),(3.8,2),(3.2634897184562,1.9863392019215)])
    window.object_manager.create_polygon([(3.8,2),(3.9641717545561,1.6929285993046),(4.8865764970506,3.2644342475013),(4.5241599157606,3.2591818332797)])
    window.object_manager.create_polygon([(4.8865764970506,3.2644342475013),(4.5241599157606,3.2591818332797),(3.7875298444594,4.4982030235776),(3.9751578728795,4.8072397296619)])


    window.initialize_window()
    glutMainLoop()
main()

```

```python
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *
import math
import numpy as np

colors = {
        'Red': (255, 0, 0),
        'Green': (0, 255, 0),
        'Blue': (0, 0, 255),
        'Yellow': (255, 255, 0),
        'Cyan': (0, 255, 255),
        'Magenta': (255, 0, 255),
        'White': (255, 255, 255),
        'Black': (0, 0, 0),
        'Gray': (128, 128, 128),
        'Aqua': (0, 255, 255),
        'Lime': (0, 255, 0),
        'Navy': (0, 0, 128),
        'Fuchsia': (255, 0, 255),
        'Olive': (128, 128, 0),
        'Teal': (0, 128, 128),
        'Maroon': (128, 0, 0),
    }
# Initialization and Window Configuration
class OpenGLInitializer:
    """
    Initializes and configures the OpenGL window.
    """
    def __init__(self, window_size=(1280, 720), window_title="OpenGL Window",kiri=-500.0,kanan=500,atas=500,bawah=-500,z_start=0.0,z_end=1.0):
        self.window_size = window_size
        self.window_title = window_title
        self.fullscreen = False
        self.object_manager = ObjectManager()
        self.transform = Transform(self.object_manager.get_Objects())
        self.kiri=kiri
        self.kanan=kanan
        self.atas=atas
        self.bawah=bawah
        self.z_start=z_start
        self.z_end=z_end

    def initialize_window(self):
        """
        Initialize the OpenGL window using the appropriate library (e.g., GLUT).
        """
        glutInit()
        glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH)
        glutInitWindowSize(*self.window_size)
        glutCreateWindow(self.window_title)
        glutDisplayFunc(self.display)
        self.animation()

        # Fungsi keyboard untuk menangani tombol 'f' untuk fullscreen
        glutKeyboardFunc(self.keyboard)
        glutIdleFunc(self.animation)

    def set_window_properties(self, size, title):
        """
        Set window properties such as size and title.

        Args:
            size (tuple): The size of the window in (width, height) format.
            title (str): The title of the window.
        """
        self.window_size = size
        self.window_title = title

    def toggle_fullscreen(self):
        """ set window to fullscreen """
        self.fullscreen = not self.fullscreen
        if self.fullscreen:
            glutFullScreen()
        else:
            glutReshapeWindow(*self.window_size)
            glutPositionWindow(100, 100)

    def set_modelView(self):
        """
        set window about transformation or viewport in openGL
        """
        glLoadIdentity()
        glViewport(0,0,self.window_size[0],self.window_size[1])
        glMatrixMode(GL_PROJECTION)
        glLoadIdentity()
        glOrtho(self.kiri,self.kanan,self.atas,self.bawah,self.z_start,self.z_end)


    def display(self):
        """
        function for eksecution objek in objek_manajer
        """
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        self.set_modelView()
        self.object_manager.draw_object()
        glutSwapBuffers()

    def keyboard(self, key, x, y):
        if key == b'f':
            self.toggle_fullscreen()
            if self.fullscreen:
                print("Mode Layar Penuh Aktif")
            else:
                print("Mode Layar Penuh Nonaktif")
    def animation(self):
        self.object_manager.set_objects(self.transform.apply_transf())
        glutPostRedisplay()

# Object Management
class ObjectManager:
    """
    Manages objects in the OpenGL scene.
    """
    def __init__(self,objects=[]):
        self.objects = objects

    def get_Objects(self):
        return self.objects

    def remove_objects(self, name="None"):
        """
        Remove objects from the Object Manager.

        Args:
            name (str, optional): The name or type of objects to remove. Default is "None" which does nothing.
                You can use "All" to remove all objects, or specify the type of objects to remove
                (e.g., "circle", "rectangle", "triangle", "line", "polygon", "point").

        Example:
            # Remove all objects
            object_manager.remove_objects("All")

            # Remove all circle objects
            object_manager.remove_objects("circle")

            # Remove all objects of a specific name (user-defined)
            object_manager.remove_objects("my_object")

            # Do nothing (default behavior)
            object_manager.remove_objects()
        """
        if name == "All":
            self.objects.clear()
        else:
            # Create a list to store objects to be removed
            to_remove = []

            for obj in self.objects:
                if name == "None" or obj['type'] == name or obj['name'] == name:
                    to_remove.append(obj)

            # Remove objects from the to_remove list
            for obj in to_remove:
                self.objects.remove(obj)

    def set_objects(self,objects):
        self.objects=objects

    def create_rectangle(self, x, y, width, height,name="default", color=colors['White']):
        """
        Create a rectangle object.

        Args:
            x (float): X-coordinate of the rectangle's origin.
            y (float): Y-coordinate of the rectangle's origin.
            width (float): Width of the rectangle.
            height (float): Height of the rectangle.
            color (tuple, optional): Color of the rectangle in RGB format. Default is white (1.0, 1.0, 1.0).
        """
        obj = {
            "name": name,
            "type": "rectangle",
            "x": x,
            "y": y,
            "width": width,
            "height": height,
            "point":[(x, y),(x + width, y),(x + width, y + height),(x, y + height)],
            "color": color
        }
        self.objects.append(obj)

    def create_circle(self, x, y, radius, name="default", color=colors['White'],opsi=1,k=6.276):
        """
        Create a circle object.

        Args:
            x (float): X-coordinate of the circle's center.
            y (float): Y-coordinate of the circle's center.
            radius (float): Radius of the circle.
            color (tuple, optional): Color of the circle in RGB format. Default is white (1.0, 1.0, 1.0).
        """
        poin = []
        dg = int(361*opsi)
        for i in range(dg):
            angle = k-i * 3.1415926 / 180
            poin.append((x + radius * math.cos(angle), y + radius * math.sin(angle)))
        obj = {
            "name": name,
            "type": "circle",
            "x": x,
            "y": y,
            "radius": radius,
            "point":poin,
            "color": color
        }
        self.objects.append(obj)

    def create_triangle(self, x, y, side_length, name="default",color=colors['White']):
        """
        Create a triangle object.

        Args:
            x (float): X-coordinate of the triangle's origin.
            y (float): Y-coordinate of the triangle's origin.
            side_length (float): Length of each side of the equilateral triangle.
            color (tuple, optional): Color of the triangle in RGB format. Default is white (1.0, 1.0, 1.0).
        """

        obj = {
            "name": name,
            "type": "triangle",
            "x": x,
            "y": y,
            "side_length": side_length,
            "point":[(x, y),(x + side_length, y),(x + side_length / 2, y + (side_length * math.sqrt(3)) / 2)],
            "color": color
        }
        self.objects.append(obj)

    def create_point(self, x, y, name="default",color=colors['White']):
        """
        Create a point object.

        Args:
            x (float): X-coordinate of the point.
            y (float): Y-coordinate of the point.
            color (tuple, optional): Color of the point in RGB format. Default is white (1.0, 1.0, 1.0).
        """
        obj = {
            "name": name,
            "type": "point",
            "x": x,
            "y": y,
            "color": color
        }
        self.objects.append(obj)

    def create_polygon(self, vertices:list, name="default", color=colors['White']):
        """
        Create a polygon object.

        Args:
            vertices (list of tuple): List of vertex positions in the format [(x1, y1), (x2, y2), ...].
            color (tuple, optional): Color of the polygon in RGB format. Default is white (1.0, 1.0, 1.0).
        """
        obj = {
            "name": name,
            "type": "polygon",
            "point": vertices,
            "color": color
        }
        self.objects.append(obj)

    def create_line(self, x1=0, y1=0, x2=0, y2=0, length_line=0, degree=0, color=colors['White'],lines=[],name="default",):
        """
        Create a line object.

        Args:
            x1 (float): X-coordinate of the starting point of the line.
            y1 (float): Y-coordinate of the starting point of the line.
            x2 (float): X-coordinate of the ending point of the line.
            y2 (float): Y-coordinate of the ending point of the line.
            length_line (float): Length of the line (optional).
            degree (float): Rotation angle in degrees (optional).
            color (tuple, optional): Color of the line in RGB format. Default is white (1.0, 1.0, 1.0).
        """
        point = []
        if x2 != 0 and y2 != 0 and lines==[]:
            point.append((x1,y1))
            point.append((x2,y2))
        else:
            x2 = x1 + length_line * math.cos(degree * math.pi / 180.0)
            y2 = y1 + length_line * math.sin(degree * math.pi / 180.0)
            point.append((x1,y1))
            for i in lines:
                point.append(i)
            point.append((x2,y2))
        obj = {
            "name": name,
            "type": "line",
            "x1": x1,
            "y1": y1,
            "x2": x2,
            "y2": y2,
            'point':point,
            "n_point":lines,
            "length_line": length_line,
            "degree": degree,
            "color": color
        }
        self.objects.append(obj)

    def draw_object(self):
        """
        Draw the specified objects on the screen.
        """
        for obj in self.objects:
            if obj["type"] == "rectangle":
                self.draw_rectangle(obj)
            elif obj["type"] == "circle":
                self.draw_circle(obj)
            elif obj["type"] == "triangle":
                self.draw_triangle(obj)
            elif obj["type"] == "point":
                self.draw_point(obj)
            elif obj["type"] == "line":
                self.draw_line(obj)
            elif obj["type"] == "polygon":
                self.draw_polygon(obj)

    def draw_rectangle(self, obj):
        """
        Draw a rectangle object.
        """
        color = obj["color"]

        glColor3f(*color)
        glBegin(GL_QUADS)
        for i in range(len(obj['point'])):
            glVertex2f(*obj['point'][i])
        glEnd()

    def draw_circle(self, obj):
        """
        Draw a circle object.
        """
        color = obj["color"]
        glColor3f(*color)
        glBegin(GL_TRIANGLE_FAN)
        for i in range(len(obj['point'])):
            glVertex2f(*obj['point'][i])
        glEnd()

    def draw_triangle(self, obj):
        """
        Draw a triangle object.
        """
        color = obj["color"]
        glColor3f(*color)
        glBegin(GL_TRIANGLES)
        for i in range(len(obj['point'])):
            glVertex2f(*obj['point'][i])
        glEnd()

    def draw_point(self, obj):
        """
        Draw a point object.
        """
        x = obj["x"]
        y = obj["y"]
        color = obj["color"]

        glColor3f(*color)
        glBegin(GL_POINTS)
        glVertex2f(x, y)
        glEnd()

    def draw_polygon(self, obj):
        """
        Draw a polygon object.
        """
        vertices = obj["point"]
        color = obj["color"]

        glColor3f(*color)
        glBegin(GL_POLYGON)
        for vertex in vertices:
            glVertex2f(*vertex)
        glEnd()

    def draw_line(self, obj):
        """
        Draw a line object.
        """
        glColor3f(*obj['color'])
        glBegin(GL_LINES)
        for i in range(len(obj['point'])):
            if i+1 != len(obj['point']):
                glVertex2f(*obj['point'][i])
                glVertex2f(*obj['point'][i+1])
        glEnd()

# Transform
class Transform:
    def __init__(self, Objects=[]):
        """
        Initializes a new instance of the Transform class.

        The Transform class provides methods to perform various transformations (translation, scaling, rotation)
        on objects or shapes in a 2D OpenGL scene.

        Args:
            Objects (list of dict, optional): A list of objects to be manipulated. Default is an empty list.

        Returns:
            None
        """
        self.Object = Objects

    def translate(self, dx: float, dy: float, name="default"):
        """
        Translate an object or shape by a specified displacement.

        Args:
            dx (float): The displacement along the x-axis.
            dy (float): The displacement along the y-axis.
            name (str, optional): The name or type of object to be translated. Default is "default" (all objects).

        Returns:
            None
        """
        for obj in range(len(self.Object)):
            new_point = []
            if self.Object[obj]['name'] in name or self.Object[obj]['type'] == name:
                for i in range(len(self.Object[obj]['point'])):
                    new_point.append((self.Object[obj]['point'][i][0] + dx,self.Object[obj]['point'][i][1] + dy))
                self.Object[obj]['point'] = new_point


    def scale(self, scale_factor,name="default"):
        """
        Scale an object or shape by a specified scaling factor.

        Args:
            scale_factor (float): The scaling factor.

        Returns:
            None
        """
        for obj in range(len(self.Object)):
            new_point = []
            if self.Object[obj]['name'] in name or self.Object[obj]['type'] == name:
                for i in range(len(self.Object[obj]['point'])):
                    new_point.append((self.Object[obj]['point'][i][0] * scale_factor,self.Object[obj]['point'][i][1] * scale_factor))
                self.Object[obj]['point'] = new_point

    def rotate(self, angle_degrees, name="default"):
        """
        Rotate an object or shape by a specified angle (in degrees) around a specified center point.

        Args:
            angle_degrees (float): The rotation angle in degrees.
            center_x (float, optional): The x-coordinate of the center of rotation. Default is 0.0.
            center_y (float, optional): The y-coordinate of the center of rotation. Default is 0.0.
            name (str): The name of the object to be rotated. Default is "default".

        Returns:
            None
        """
        pointCenter = 0
        for obj in range(len(self.Object)):
            new_point = []
            if self.Object[obj]['name'] == name or self.Object[obj]['type'] == name:
                angle_radians = np.radians(angle_degrees)
                x_total = 0
                y_total = 0
                for x,y in range(len(self.Object[obj]['point'])):
                    x_total += x
                    y_total += y
                mid = (x_total/len(self.Object[obj]['point']),y_total/len(self.Object[obj]['point']))

                self.Object[obj]['point'] = new_point
        pass




    def apply_transf(self):
        """
        Returns the list of objects after applying transformations.

        Args:
            None

        Returns:
            list of dict: The list of objects after applying transformations.
        """
        return self.Object
```