Kathmandu University
Department of Computer Science and Engineering

Dhulikhel, Kavre

Report of Lab4: Graphics 2D Transformation

[Code No: COMP 342]

Submitted by:

Abiral Adhikari (02)

Submitted to:

Dhiraj Shrestha

Department of Computer Science and Engineering

Submission Date: 30/05/2024

Lab Exercise1:

Write a Program to implement:

a. 2D Translation
b. 2D Rotation
c. 2D Scaling
d. 2D Reflection
e. 2D Shearing
f. Composite Transformation (Should be able to perform atleast 3 transformations)

(For doing these Transformations consider any 2D shapes (Line, Triangle, Rectangle etc), and use Homogeneous coordinate Systems)

- **Source Code:**

```
import pygame

from pygame.locals import *

from OpenGL.GL import *

from OpenGL.GLUT import *

from OpenGL.GLU import *

import numpy as np


# Function to draw axes

def draw_axes():

    glBegin(GL_LINES)

    glColor3f(1.0, 1.0, 1.0)  # Set color to white

    glVertex2i(-400, 0)

    glVertex2i(400, 0)

    glVertex2i(0, -300)

    glVertex2i(0, 300)
```

```python
    glEnd()


# Function to draw a triangle
def draw_triangle(vertices=[[0, 0], [100, 0], [100, 100]], color=[1, 1, 1]):
    glBegin(GL_TRIANGLES)
    glColor3f(color[0], color[1], color[2])
    for vertex in vertices:
        glVertex2f(*vertex)
    glEnd()


# Transformation matrices
def translate(tx, ty):
    return np.array([
        [1, 0, tx],
        [0, 1, ty],
        [0, 0, 1]
    ])


def rotate(theta):
    cos_theta = np.cos(theta)
    sin_theta = np.sin(theta)
    return np.array([
        [cos_theta, -sin_theta, 0],
        [sin_theta, cos_theta, 0],
        [0, 0, 1]
    ])

def scale(sx, sy):
    return np.array([
        [sx, 0, 0],
        [0, sy, 0],
        [0, 0, 1]
    ])

def reflect_x():
    return np.array([
        [1, 0, 0],
        [0, -1, 0],
        [0, 0, 1]
    ])

def reflect_y():
    return np.array([
        [-1, 0, 0],
        [0, 1, 0],
        [0, 0, 1]
    ])

def reflect_xy():
    return np.array([
        [0, 1, 0],
        [1, 0, 0],
        [0, 0, 1]
    ])

def shear(kx, ky):
    return np.array([
        [1, kx, 0],
        [ky, 1, 0],
        [0, 0, 1]
    ])

def composite(*transformations):
    result = np.eye(3)
    for transformation in transformations:
        result = np.dot(transformation, result)
    return result

def display_menu():
    print("Choose an operation:")
    print("1. Translation")
    print("2. Rotation")
    print("3. Scaling")
    print("4. Reflection")
    print("5. Shearing")
    print("6. Composite Transformation")
    print("7. Exit")


def get_triangle_vertices():
    vertices = []
    for i in range(3):
        while True:
            try:
                x, y = map(float, input(f"Enter coordinate {i+1} (x,y): ").split(","))
                vertices.append([x, y])
                break
            except ValueError:
                print("Invalid input! Please enter numbers separated by comma.")
    return vertices



def get_input():
    operation = int(input("Enter operation number: "))
    if operation == 7:
        print("Exiting program.")
        return operation, None
    elif operation == 6:
        print("Enter operations to be composed (e.g., '1 2 3' for translation, rotation, scaling):")
        operations = list(map(int, input().split()))
        return operation, operations
    return operation, None


def main():
    pygame.init()
    display = (800, 600)
    pygame.display.set_mode(display, DOUBLEBUF | OPENGL)
    gluOrtho2D(-400, 400, -300, 300)  # Set up 2D coordinate system

    # The default coordinates to use
    vertices = [[0, 0], [100, 0], [100, 100]]
    # Get user input for coordinates
    # vertices=get_triangle_vertices()
    vertices_homogeneous = [[x, y, 1] for x, y in vertices]
    vertices_array = np.array(vertices_homogeneous)

    transformed_vertices = vertices  # Initialize with original vertices


    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                quit()

        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
        draw_axes()
        draw_triangle(vertices, [1, 1, 1])  # Draw original triangle in white
        draw_triangle(transformed_vertices, [1, 0, 0])  # Draw transformed triangle in red
        pygame.display.flip()

        display_menu()
        operation, operations = get_input()

        if operation == 7:
            break

        if operation == 6:
            transformations = []
            for op in operations:
                if op == 1:
                    tx, ty = map(int, input("Enter translation values (tx,ty): ").split(","))
                    transformations.append(translate(tx, ty))
                elif op == 2:
                    theta = float(input("Enter rotation angle (in degrees): "))
```

```
transformations.append(rotate(np.radians(theta)))
                elif op == 3:
                    sx, sy = map(float, input("Enter scaling factors
(sx,sy): ").split(","))
                    transformations.append(scale(sx, sy))
                elif op == 4:
                    axis = input("Enter reflection axis (x or y or
x=y): ")
                    if axis == 'x':
                        transformations.append(reflect_x())
                    if axis=="y":
                        transformations.append(reflect_y())
                    if axis=="x=y":
                        transformations.append(reflect_xy())
                elif op == 5:
                    kx, ky = map(float, input("Enter shearing
factors (kx,ky): ").split(","))
                    transformations.append(shear(kx, ky))

            composite_transform = composite(*transformations)
            print("Composite Transformation Matrix:")
            print(composite_transform)
            transformed_vertices = np.dot(composite_transform,
vertices_array.T).T[:, :2]
            transformed_vertices = transformed_vertices.tolist()  #
Update transformed vertices
```

```
        else:

            if operation == 1:

                tx, ty = map(int, input("Enter translation values
(tx,ty): ").split(","))

                transformation_matrix = translate(tx, ty)

            elif operation == 2:

                theta = float(input("Enter rotation angle (in
degrees): "))

                transformation_matrix = rotate(np.radians(theta))

            elif operation == 3:

                sx, sy = map(float, input("Enter scaling factors
(sx,sy): ").split(","))

                transformation_matrix = scale(sx, sy)

            elif operation == 4:

                axis = input("Enter reflection axis (x or y or x=y):
")

                if axis == 'x':

                    transformation_matrix = reflect_x()

                if axis=="y":

                    transformation_matrix=reflect_y()

                if axis=="x=y":

                    transformation_matrix=reflect_xy()

            elif operation == 5:

                kx, ky = map(float, input("Enter shearing factors
(kx,ky): ").split(","))

                transformation_matrix = shear(kx, ky)
            print("Transformation Matrix:")
            print(transformation_matrix)
            transformed_vertices = np.dot(transformation_matrix, vertices_array.T).T[:, :2]
            transformed_vertices = transformed_vertices.tolist()  # Update transformed vertices


        pygame.time.wait(10)


if __name__ == "__main__":
    main()
```

My source code consist of functions:

draw_triangle(vertices,color) to draw triangle.

translate(tx,ty) for translation.

rotate(theta in degrees) for rotation.

scale(sx,sy) for scaling.

shear(kx,ky) for shearing.

reflect_x(), reflect_y() and reflect_xy() for reflection about X-axis, Y-axis, and x=y Line respectively.

composite(transformation list) for getting composite transformation matrix for list of transformations.

display_menu() to display the menu in the terminal for the user to choose desired transformation.

get_input() to take the user chosen operation number to call the desired function.

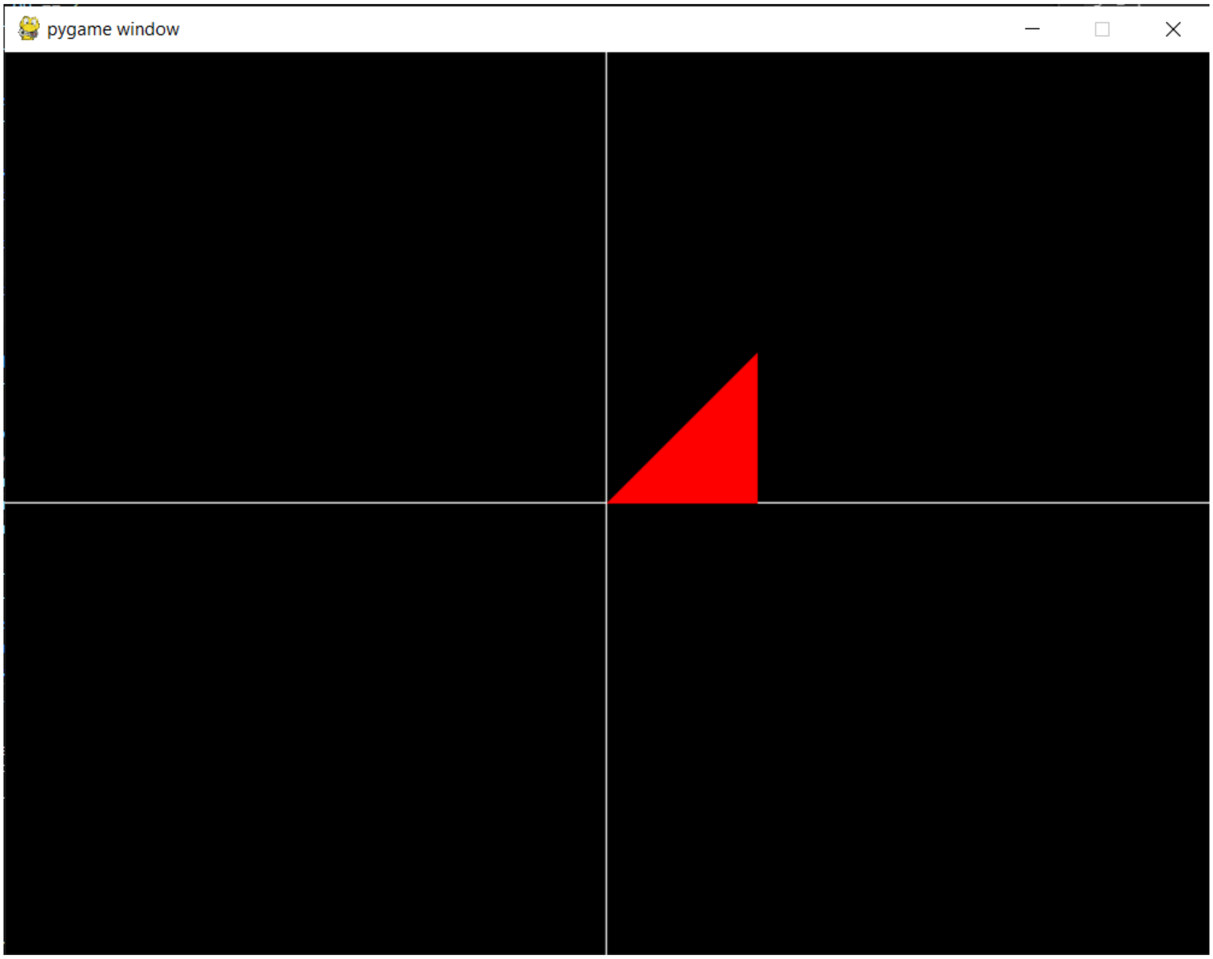
get_triangle_vertices() to get the coordinates of the triangle from the user.

main() function is where the program starts and Pygame window is initialized, 2D coordinate system is set up and the triangles are drawn by calling all above functions.

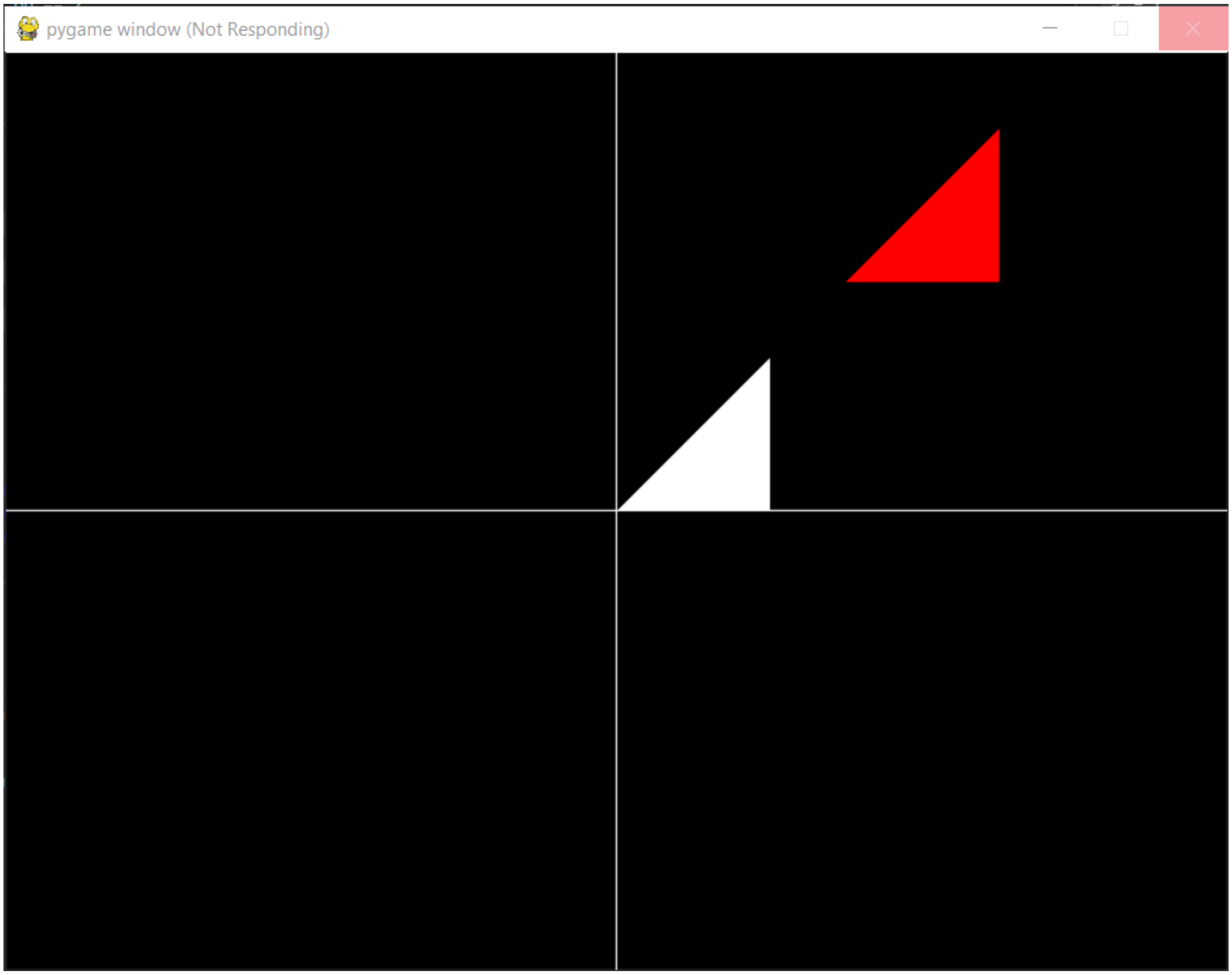
The code uses nested loops to call the correct function based on user input.

- **Output:**

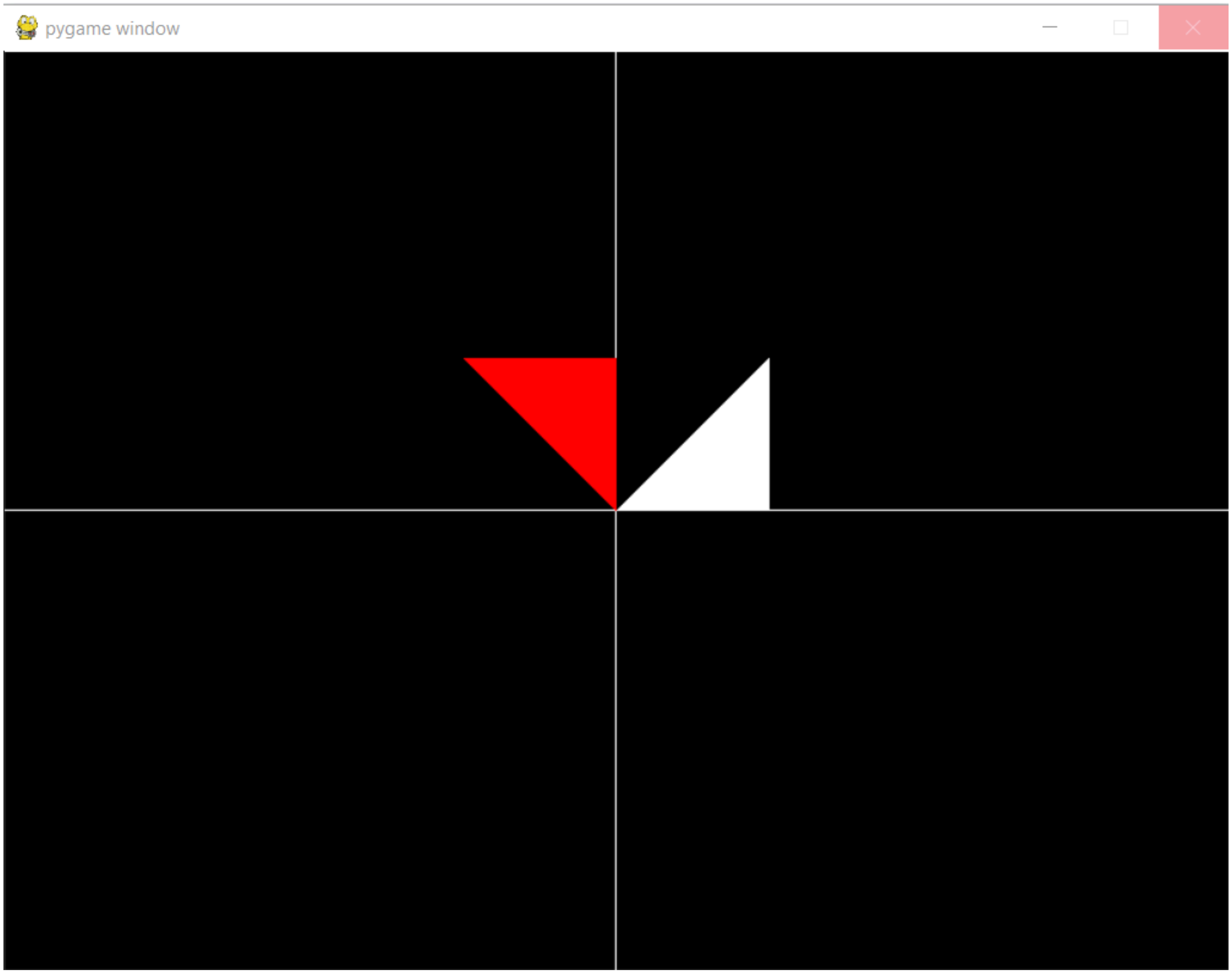
  a. **Original Triangle:**



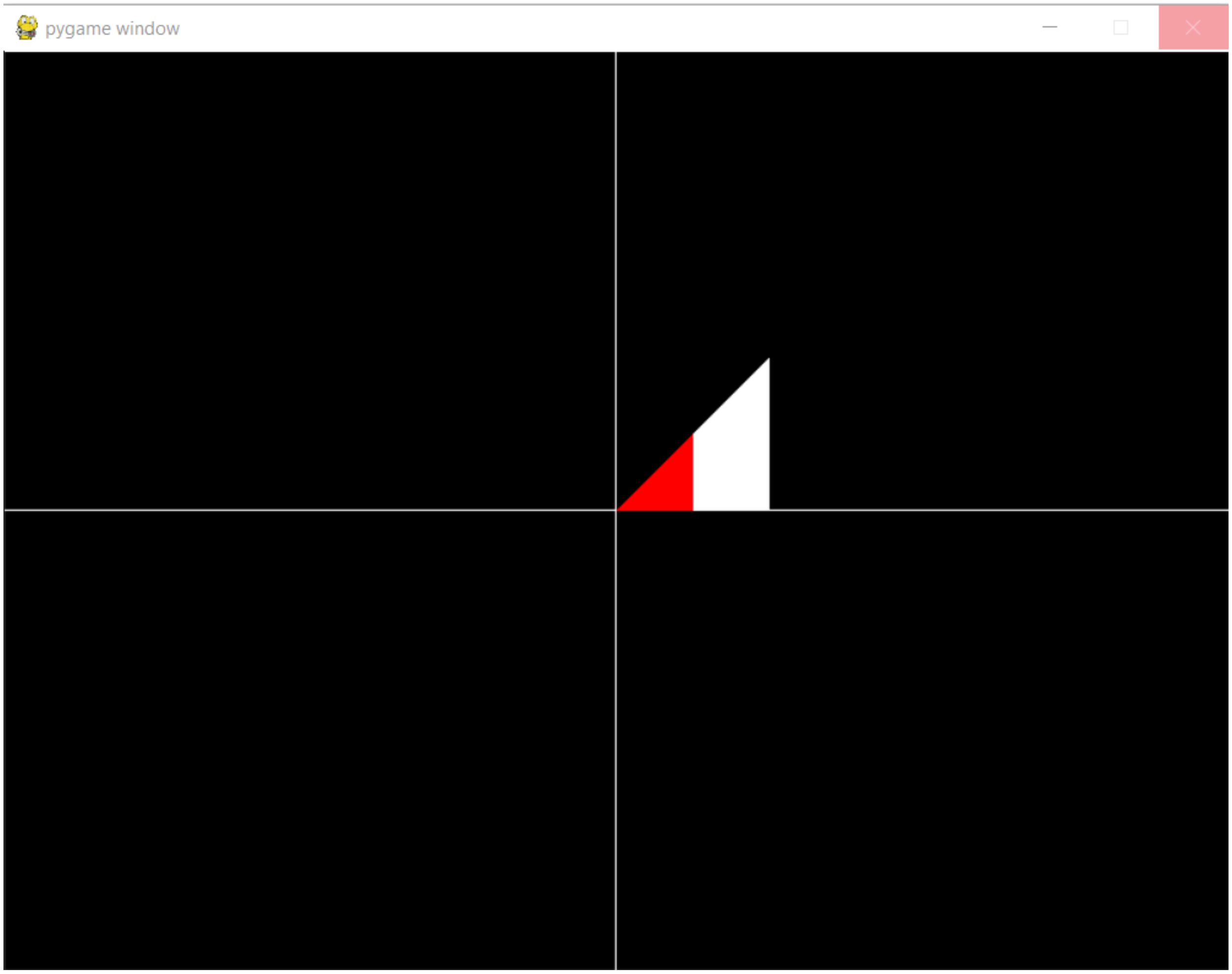
b. Translation(150,150):



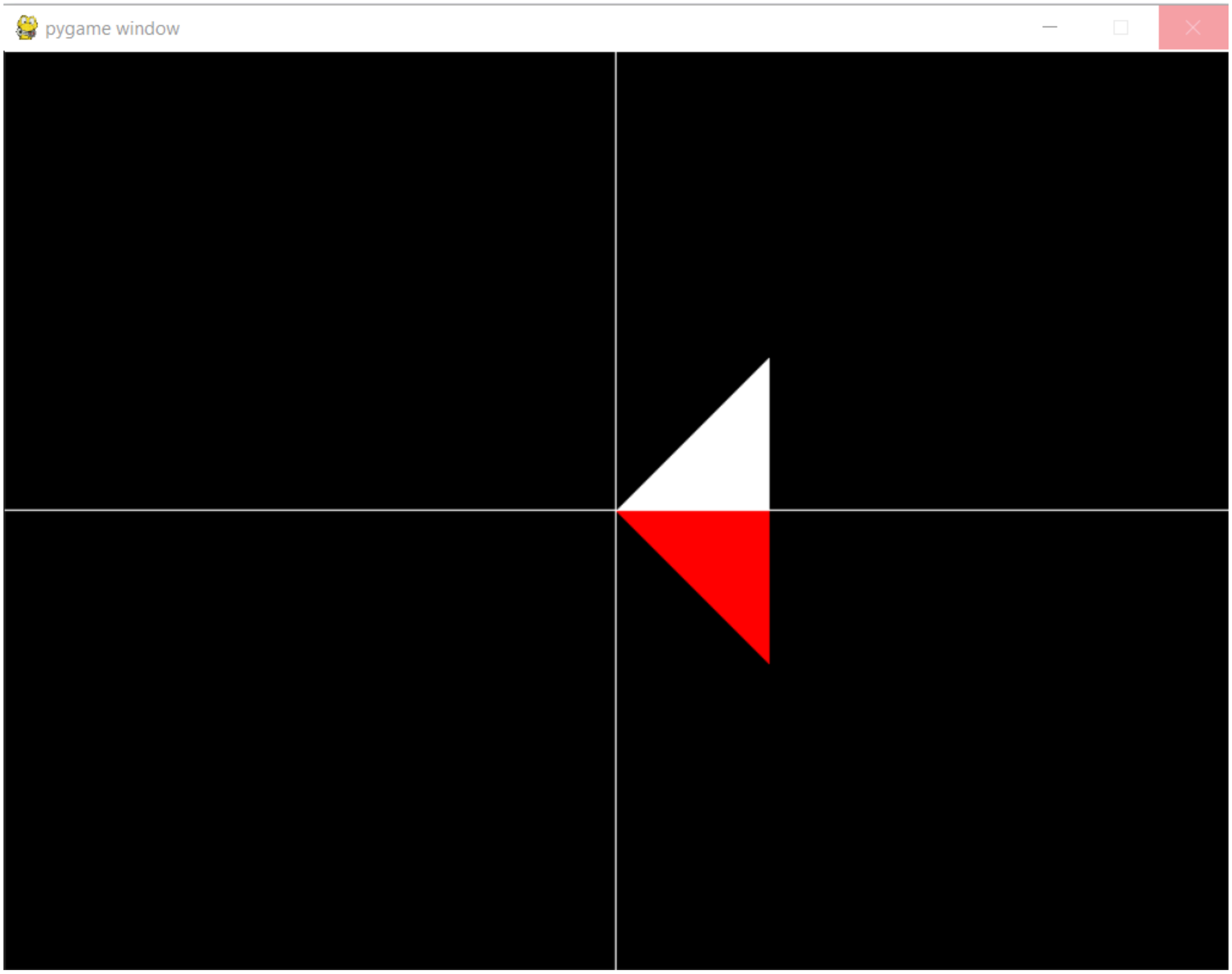
c. Rotation(90):



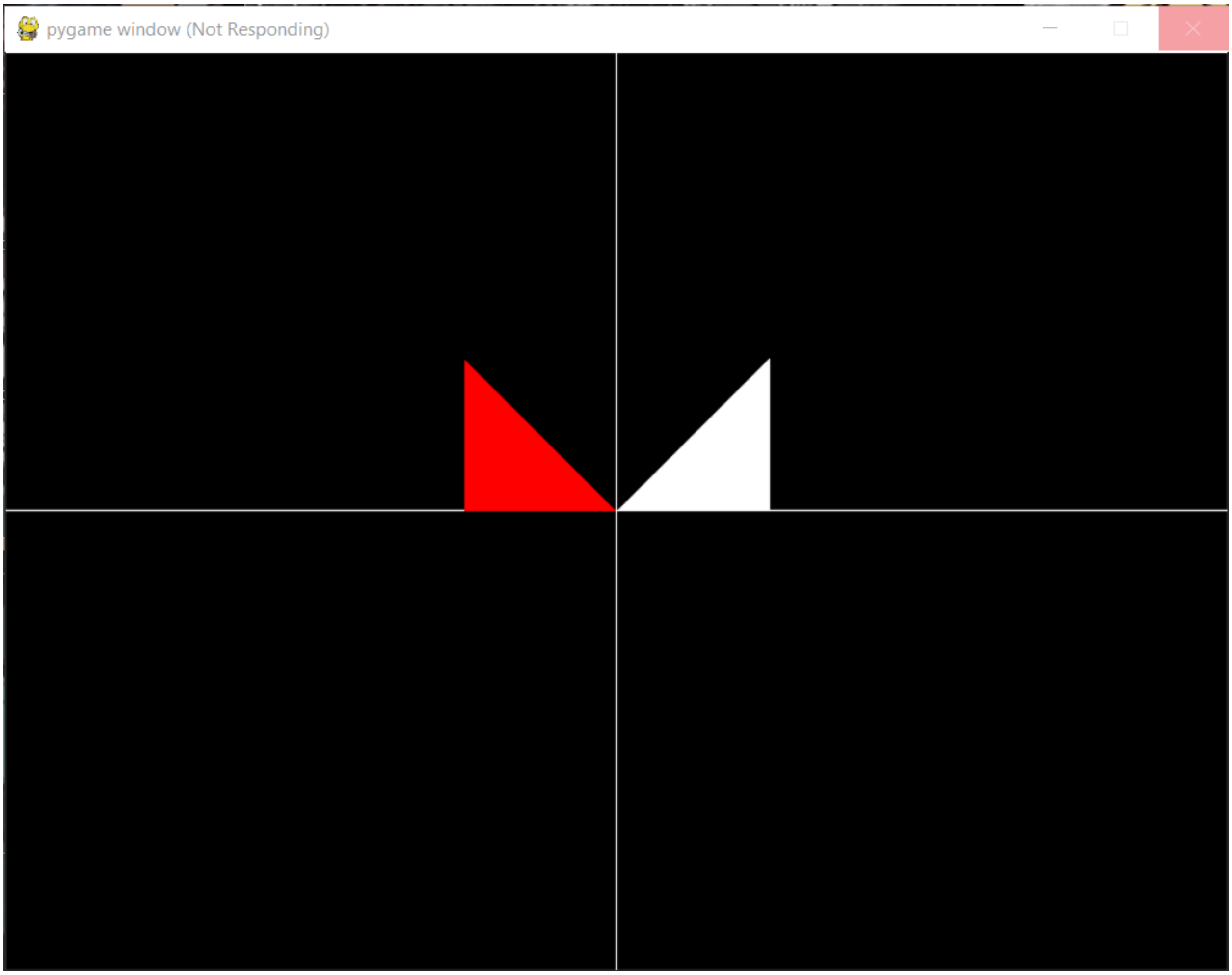
d. Scaling(0.5,0.5):



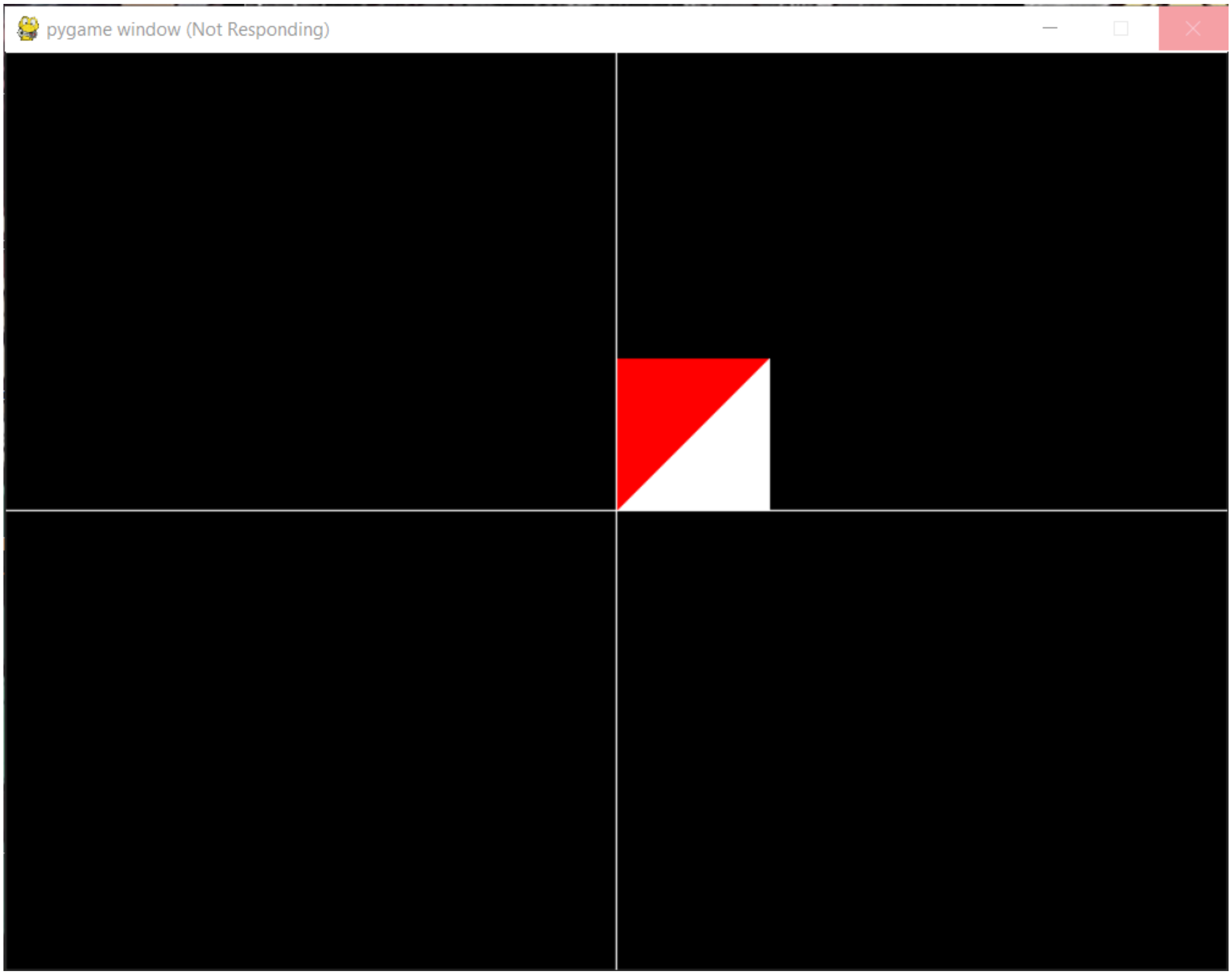
e. Reflection about X- axis:



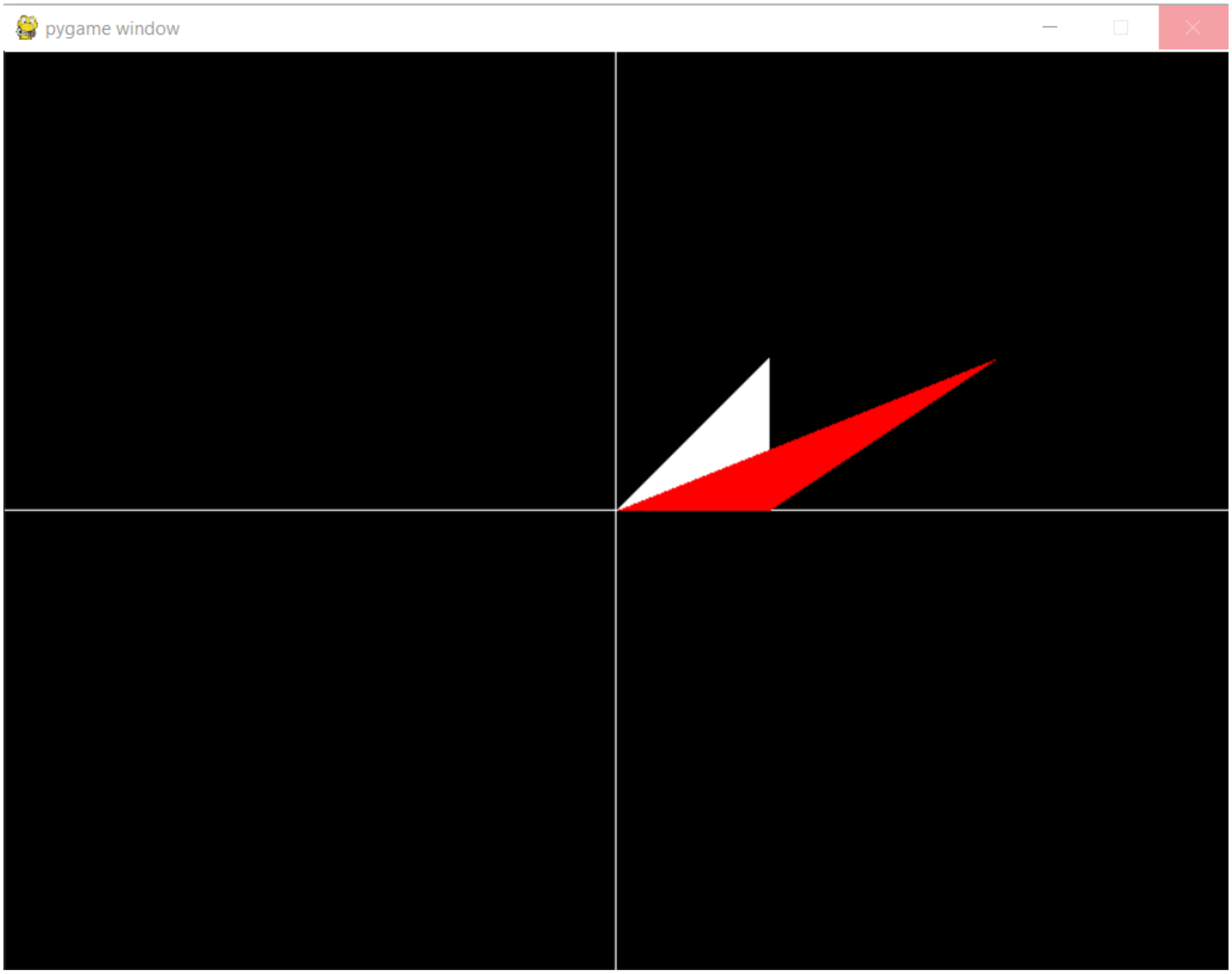
f. Reflection about Y-axis:



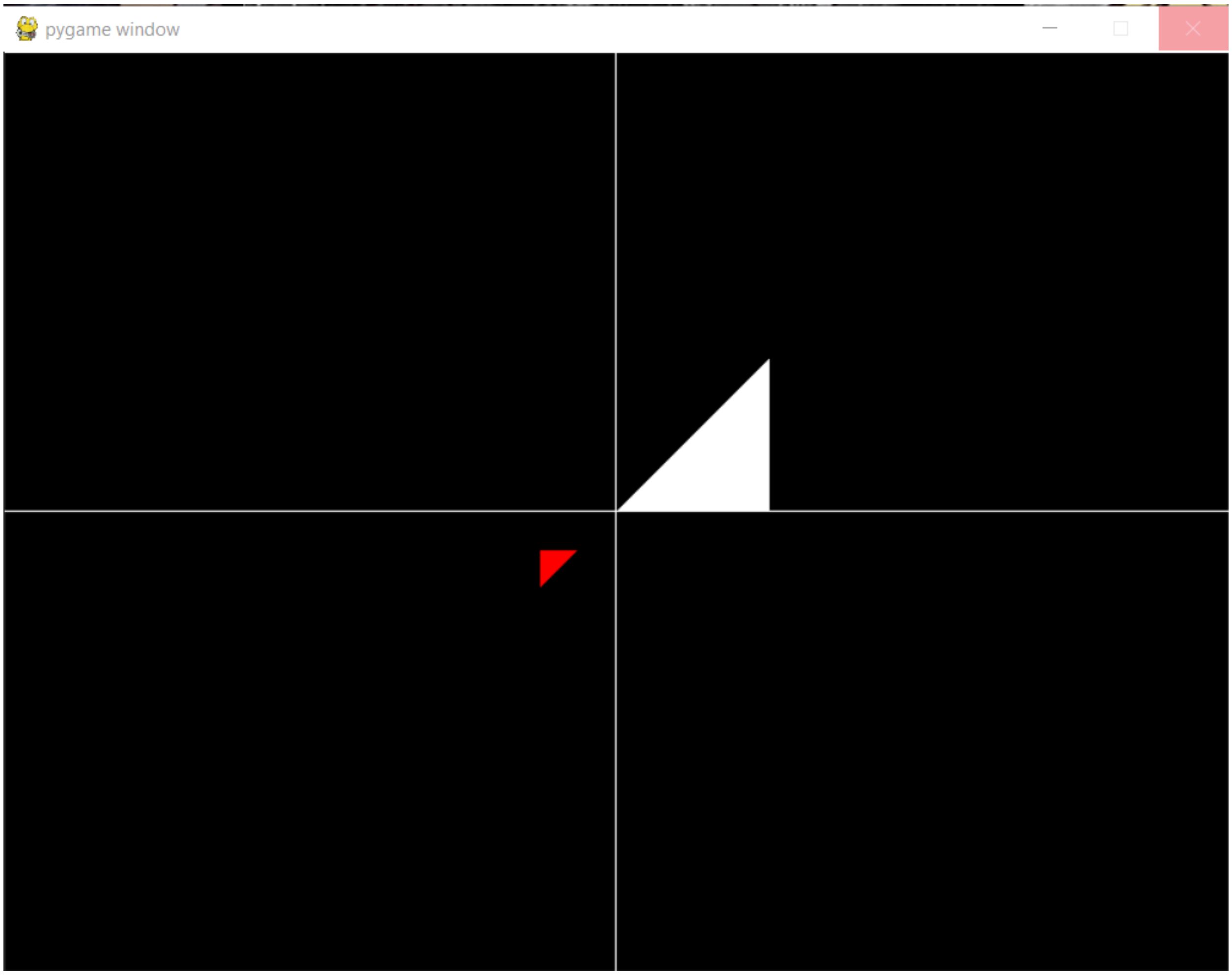
g. Reflection about y=x line:



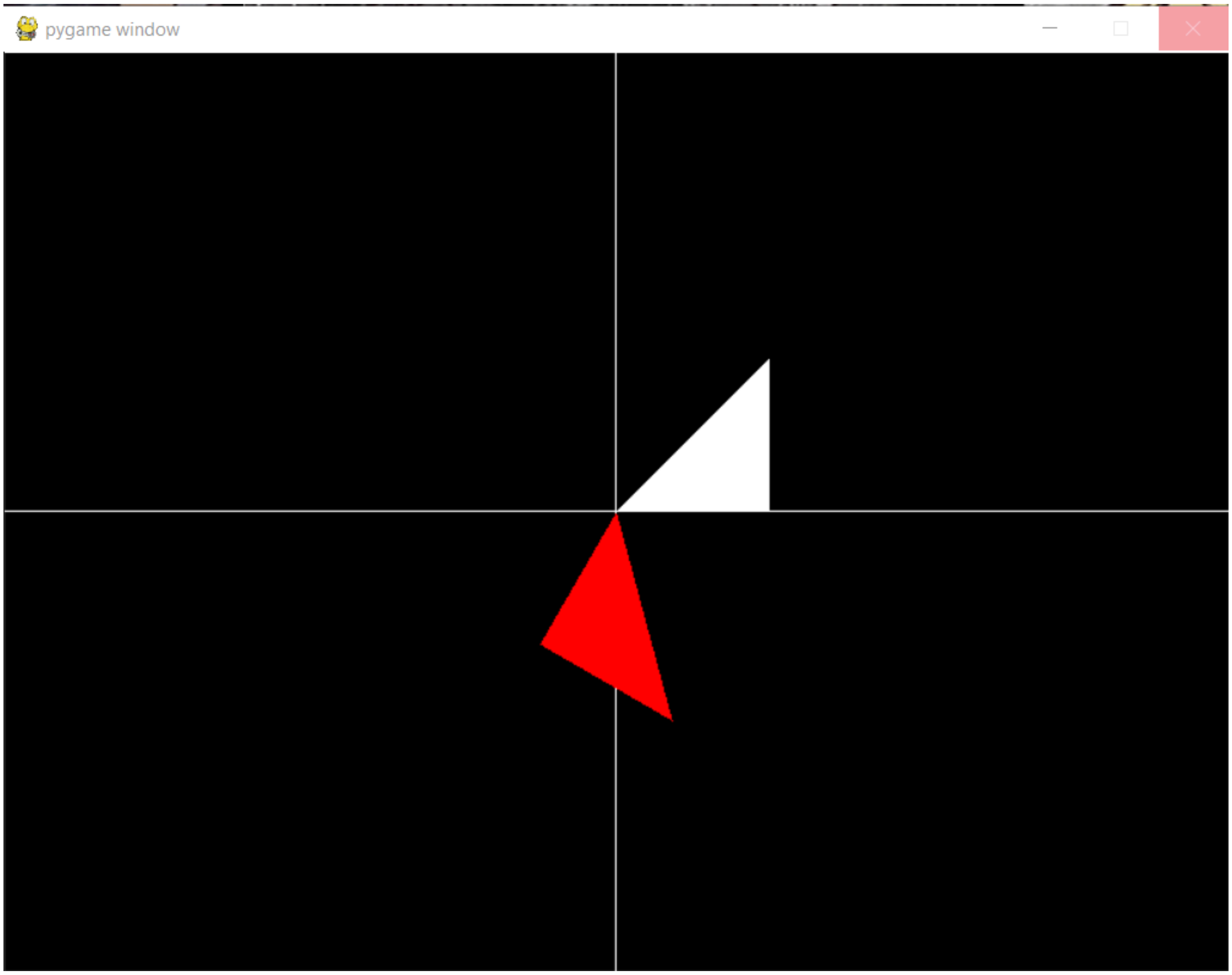
h. Shearing(1.5,0):



i. Composite Transformation 1(T(100,100),Rot(180),S(0.25,0.25)):



j. Composite Transformation 2(Ref(x=y),T(-100,-100),Rot(60)):


pygame window