Name: Nowshin Sumaiya
ID: 21301276
Sec:06

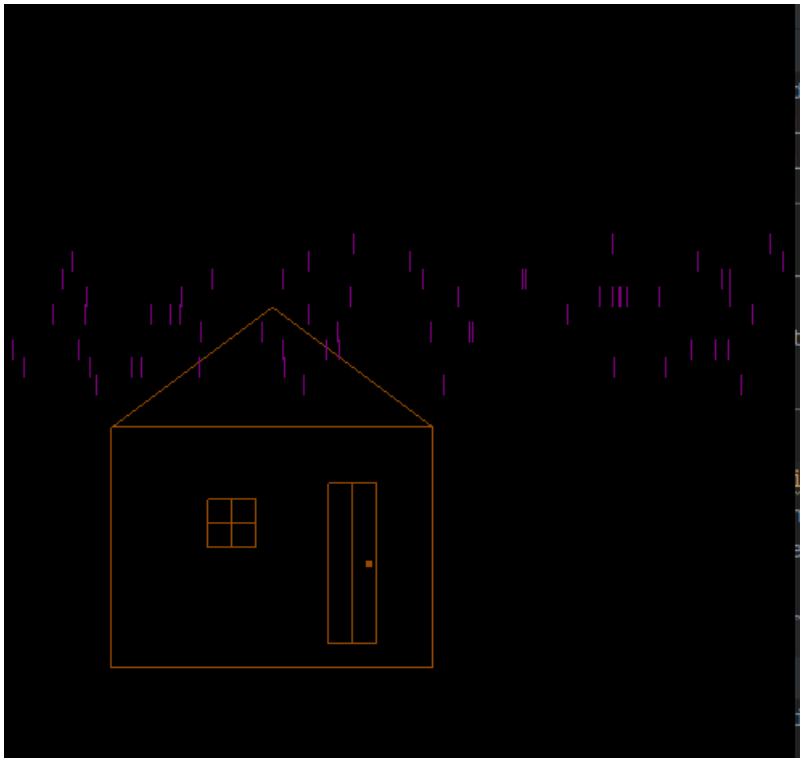
Ans to the question no :1

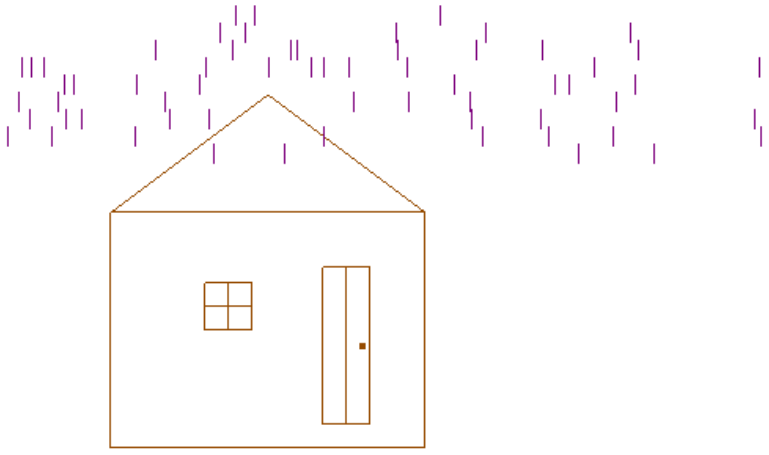
```
import random  # will use for raindrop positions and directions
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *

background_color = (0.0, 0.0, 0.0)  # background color is initially black
rain_drops = []
rain_direction = 0  # initial direction of raindrops.0 means straight down

# ----------------------------------------

def draw_line (x1, y1, x2, y2):
    glBegin(GL_LINES)
    glVertex2f(x1, y1) # glVertex2f is used to specify 2D coordinates

    glVertex2f(x2, y2)
    glEnd()

def draw_points(x, y, size=1.0):
    glPointSize(size)
    glBegin(GL_POINTS)

    glVertex2f(x, y) # place where pixel will show
    glEnd()

def iterate():
    glViewport(0, 0, 500, 500)
    glMatrixMode(GL_PROJECTION)
    glLoadIdentity()
```

```
    glOrtho(0.0, 500, 0.0, 500, 0.0, 1.0)
    glMatrixMode(GL_MODELVIEW) # selects the modelview matrix stack
    glLoadIdentity()

def showScreen():
    glClearColor(*background_color, 1.0)  # Background color 1.0 indicates the color is not
transparent
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)

    glLoadIdentity()
    iterate()
# ------------------------------------------------
    glColor3f(0.6, 0.3, 0.0)  # color of the house is brown

    # Roof of the house
    draw_line(75, 225, 175, 300)
    draw_line(275, 225, 175, 300)

    # Base of the house
    draw_line(75, 75, 275, 75)
    draw_line(75, 75, 75, 225)

    draw_line(275, 75, 275, 225)
    draw_line(75, 225, 275, 225)

    # Door
    draw_line(210, 90, 240, 90)
    draw_line(210, 90, 210, 190)

    draw_line(240, 90, 240, 190)
    draw_line(210, 190, 240, 190)
    draw_line(225, 90, 225, 190)

    # Door handle
    draw_points(235, 140, size=4.0)

    # lines in window
    draw_line(150, 150, 150, 180)
    draw_line(135, 165, 165, 165)
```

```
    # Window
    draw_line(135, 150, 165, 150)
    draw_line(135, 150, 135, 180)

    draw_line(165, 150, 165, 180)
    draw_line(135, 180, 165, 180)

    # Raindrops
    glColor3f(0.5, 0.0, 0.5) # color of rain purple

    for x, y in rain_drops:

        draw_line(x, y, x, y - 13) #loop for the rain

    glutSwapBuffers()
#---------------------------------
def animate(value):
    global rain_drops
    new_raindrops = [] #store the updated positions of  raindrops after they moved

    for x, y in rain_drops:
        y =y- 11  # decreases the y coordinate of each raindrop
        x += rain_direction  # adjusts the x-coordinate of each raindrop

        if y <= 0: #checks if a raindrop has moved off the top of the screen
            x = random.randint(0, 500)
            y = 500 # resetting the raindrop's position to the top of the screen
        new_raindrops.append((x, y))

    rain_drops[:] = new_raindrops
    glutTimerFunc(40, animate, 0)#schedules the animate function to be called again after 40ms
    glutPostRedisplay()#any changes made to the scene are reflect on the screen

def generate_raindrops():

    for x in range(0, 500, 8):
        y = random.randint(410, 500)
        rain_drops.append((x, y))

def specialKeyListener(key, x, y): #moves the raindrop position
```

```
    global rain_direction, background_color

    if key == GLUT_KEY_LEFT:
        rain_direction = -7  # change to move raindrops to the left

    elif key == GLUT_KEY_RIGHT:
        rain_direction = 7  # change to move raindrops to the right


def keyboardListener(key, x, y): #change background colour
    global background_color
    if key == b'a':
        background_color = (1.0, 1.0, 1.0)  #press 'a' to change it white means day

    elif key == b'b':
        background_color = (0.0, 0.0, 0.0)  # press 'b' to change it black means night
#------------------------------------------------------------

glutInit()
glutInitDisplayMode(GLUT_RGBA)
glutInitWindowSize(500, 500)
glutInitWindowPosition(0, 0)

wind = glutCreateWindow(b"Assignment 1_task-1") # #window name
glutDisplayFunc(showScreen)
glutSpecialFunc(specialKeyListener)
glutKeyboardFunc(keyboardListener)#function will be called when keys are pressed

generate_raindrops()
glutTimerFunc(40, animate, 0)  # Starts  animation timer
glutMainLoop()
```





Answer to the question no :02

```
import random  # will use for Generating random colors points
import time  # will use for blinking effect on the points
from OpenGL.GL import *
from OpenGL.GLUT import *
from OpenGL.GLU import *

W_Width, W_Height = 700, 700  # width and height of the OpenGL window.
random_points = []  # store information about randomly generated points
blink_timer = 0  # will use to keep track of the time duration for the blinking effect
pause = False
movement_speed = 0.02  # initial speed at which points move


# ---------------------------------------------------------

def draw_points(x, y, size, color):  # will use for draw points
    glPointSize(size)
    glBegin(GL_POINTS)
    glColor3f(*color)
    glVertex2f(x, y)
    glEnd()


total_point_generated = False  # initially  no point has been generated


def reset_point_generation_value(value):  # to reset the flag whenever we want to generate a new
point
    global total_point_generated
    total_point_generated = False


def mouseListener(button, state, x, y):  # the state of the button,the coordinates of the mouse
click
    global total_point_generated, blink_timer

    if button == GLUT_RIGHT_BUTTON and state == GLUT_DOWN and not
total_point_generated:
```

```
        # checks if the right mouse button is clicked,the button is pressed down and no point has
been generated yet
        # If all conditions is true,then create a new point
        c_X = (x - W_Width / 2) / (W_Width / 2)
        c_Y = (W_Height / 2 - y) / (W_Height / 2)

        color = [random.uniform(0, 1), random.uniform(0, 1), random.uniform(0, 1)]
        # generates a random color for the new point
        direction = generate_random_direction(c_X, c_Y)  # generates a random direction for the
new point
        random_points.append({'x': c_X, 'y': c_Y, 'color': color, 'direction': direction})

        total_point_generated = True  # it means a point has been generated
        glutPostRedisplay()

        glutTimerFunc(400, reset_point_generation_value, 0)
        # is responsible for resetting the total_point_generated flag to False after a 400ms delay


    elif button == GLUT_LEFT_BUTTON and state == GLUT_DOWN and not
total_point_generated:
        # it handles the left mouse button click
        blink_timer = time.time() + 0.7  # blink for 0.6sec extra
        total_point_generated = True
        glutPostRedisplay()
        glutTimerFunc(400, reset_point_generation_value, 0)



# ------------------------------------------------------------
def generate_random_direction(x, y):
    # generates a random floating-point number between -1 and 1
    new_x = random.uniform(-1, 1)
    new_y = random.uniform(-1, 1)
    length = (new_x ** 2 + new_y ** 2) ** 0.5  # calculates the length
    new_x /= length
    new_y /= length
    # normalize the direction vector
    # the resulting vector points in a random direction ,it has a length of 1

    return new_x, new_y
```

```
def specialKey(key, x, y):
    global movement_speed, blink_timer, pause

    if key == GLUT_KEY_UP:
        movement_speed *= 3  # pressing the up arrow key will increase the speed

    elif key == GLUT_KEY_DOWN:
        movement_speed /= 3  # pressing the down arrow key will decrease the speed


def keyboard(key, x, y):  # to pause and resume the points movement
    global pause

    if key == b' ':
        pause = not pause


# ------------------------------------
def display():
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)
    current_time = time.time()  # used to determine whether the points should blink or not
    for point_data in random_points:
        # checks whether it's time for the points to blink

        if current_time < blink_timer:
            color = [0, 0, 0]  # sets the color to black

        else:
            color = point_data['color']  # sets the color to the original color

        draw_points(point_data['x'], point_data['y'], 10, color)  # size of the points is 10

    glutSwapBuffers()


def create_animate(value):
    if not pause:
```

```
        for point_data in random_points:  # updates the positions of points bay their direction and movement speed
            point_data['x'] += movement_speed * point_data['direction'][0]
            point_data['y'] += movement_speed * point_data['direction'][1]

            # wraps the points back to the screen if they move beyond the window boundaries
            point_data['x'] = point_data['x'] % 2 - 1
            point_data['y'] = point_data['y'] % 2 - 1

    glutPostRedisplay()

    glutTimerFunc(9, create_animate, 0)
    # sets a timer for 9ms to call animate function again


# ----------------------------------------
def init():
    glClearColor(0, 0, 0, 0)  # sets the clear color of the OpenGL window to black
    glMatrixMode(GL_PROJECTION)  # transforming 3D coordinates into 2D coordinates

    glLoadIdentity()
    gluOrtho2D(-1, 1, -1, 1)  # specify the left, right, bottom, and top coordinates

    glMatrixMode(GL_MODELVIEW)
    glLoadIdentity()  # resetting the modelview matrix to its default state


glutInit()
glutInitWindowSize(W_Width, W_Height)  # sets the initial size of the window to be created

glutInitWindowPosition(0, 0)  # window positioned at the top-left corner of the screen
glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB)
wind = glutCreateWindow(b"Assignment-1 task-2")

init()
glutDisplayFunc(display)
glutTimerFunc(10, create_animate, 0)

glutMouseFunc(mouseListener)
glutSpecialFunc(specialKey)
```

```
glutKeyboardFunc(keyboard)
glutMainLoop()  # remains in this loop until the window is closed
```