E:\0NEU\CS5100\Mohammed\testing.py

```python
import pygame
import sys
import json
import math
import numpy as np
import os

WHITE = (255,255,255)
BLACK = (0,0,0)
GREY  = (200,200,200)

DRONE_COLORS = [
    (255, 0,   0),
    (0,   0, 255),
    (0, 255,   0),
    (255,255,  0),
    (255,  0,255),
    (0,  255,255),
    (255,128,  0),
    (128,  0,255),
    (0, 128,   0),
    (128,128,255),
    (255,128,128),
    (128,255,128),
    (255,255,128),
    (255,128,255),
    (128,255,255),
    (165,42,42),
    (0,128,128),
    (128,0,0),
]

class DroneAnimator:
    def __init__(self, grid_size=10, cell_size=50):
        pygame.init()
        self.grid_size = grid_size
        self.cell_size = cell_size
        self.screen_w  = self.grid_size*self.cell_size + 400
        self.screen_h  = self.grid_size*self.cell_size + 100
        self.screen    = pygame.display.set_mode((self.screen_w, self.screen_h))
        pygame.display.set_caption("Center-based Drone Visualization")

        self.font       = pygame.font.SysFont("Arial",16)
        self.title_font = pygame.font.SysFont("Arial",24,bold=True)

        self.coverage_grid = np.zeros((self.grid_size,self.grid_size), dtype=int)
        self.drone_positions=[]
        self.drone_sizes=[]
        self.obs_cells=[]
        self.final_reward=0.0

        self.current_drone=-1
        self.animation_active=False
        self.animation_speed=1.0
        self.animation_duration=0.6
        self.expanding=False
        self.expansion_timer=0.0

        self.coverage_history=[0]
        self.drone_coverage_cells=[]
        self.current_expanded=set()

        self.clock=pygame.time.Clock()

    def load_results(self,filename):
        if not os.path.exists(filename):
            print("[ERROR] file not found:",filename)
            return False
        try:
            with open(filename,"r") as f:
                data=json.load(f)
            self.grid_size = data.get("grid_size",10)
            self.final_reward = data.get("final_reward",0.0)
            self.drone_positions = data.get("drone_positions",[])
            self.drone_sizes     = data.get("drone_radii",[])
            self.obs_cells       = data.get("obstacles",[])

            self.drone_coverage_cells=[]
            for i,(cx,cy) in enumerate(self.drone_positions):
                s = self.drone_sizes[i]
                half=(s-1)//2
                cells=[]
                for dx in range(-half,half+1):
                    for dy in range(-half,half+1):
                        gx=cx+dx
                        gy=cy+dy
                        if 0<=gx<self.grid_size and 0<=gy<self.grid_size:
                            cells.append((gx,gy))
                self.drone_coverage_cells.append(cells)

            print(f"[INFO] Loaded {len(self.drone_positions)} drones from {filename}")
            return True
        except Exception as e:
            print("[ERROR] could not parse JSON =>", e)
            return False

    def reset_animation(self):
        self.coverage_grid[:]=0
        self.coverage_history=[0]
        self.current_drone=-1
        self.animation_active=False
        self.expanding=False
        self.expansion_timer=0.0
        self.current_expanded.clear()

    def place_next_drone(self):
        if self.current_drone+1 < len(self.drone_positions):
            self.current_drone+=1
            self.expanding=True
            self.expansion_timer=0.0
            self.current_expanded.clear()
            return True
        return False

    def update_expansion(self,dt):
        if not self.expanding:
            return
        i=self.current_drone
        if i<0 or i>=len(self.drone_positions):
            return
        self.expansion_timer+=dt
        frac=min(1.0,self.expansion_timer/self.animation_duration)

        all_cells=self.drone_coverage_cells[i]
        total=len(all_cells)
        reveal_count=int(frac*total)
        newly=all_cells[:reveal_count]

        # remove old partial coverage from that drone
        for (gx,gy) in self.current_expanded:
            self.coverage_grid[gx,gy]=0

        self.current_expanded=set(newly)
        for (gx,gy) in self.current_expanded:
            self.coverage_grid[gx,gy]=1

        if frac>=1.0:
            self.expanding=False
            cov=np.sum(self.coverage_grid)
            self.coverage_history.append(cov)

    def draw_scene(self):
        self.screen.fill(WHITE)

        for i in range(self.grid_size+1):
            pygame.draw.line(self.screen,BLACK,(i*self.cell_size,0),
                             (i*self.cell_size,self.grid_size*self.cell_size),1)
            pygame.draw.line(self.screen,BLACK,(0,i*self.cell_size),
                             (self.grid_size*self.cell_size,i*self.cell_size),1)

        for (ox,oy) in self.obs_cells:
            r=pygame.Rect(ox*self.cell_size,oy*self.cell_size,self.cell_size,self.cell_size)
            pygame.draw.rect(self.screen,(150,150,150),r)

        for gx in range(self.grid_size):
            for gy in range(self.grid_size):
                if self.coverage_grid[gx,gy]==1:
rect=pygame.Rect(gx*self.cell_size,gy*self.cell_size,self.cell_size,self.cell_size)
                    s=pygame.Surface((self.cell_size,self.cell_size),pygame.SRCALPHA)
                    s.fill((255,0,0,60))
                    self.screen.blit(s,rect)

        # highlight each drone's bounding box
        for i in range(self.current_drone+1):
            cx,cy = self.drone_positions[i]
            side  = self.drone_sizes[i]
            color = DRONE_COLORS[i%len(DRONE_COLORS)]
            half  = (side-1)//2
            left  = cx-half
            top   = cy-half

            if left<0: left=0
            if top<0:  top=0
            w = side*self.cell_size
            h = side*self.cell_size
            if left+side>self.grid_size:
                w=(self.grid_size-left)*self.cell_size
            if top+side>self.grid_size:
                h=(self.grid_size-top)*self.cell_size

            drone_rect = pygame.Rect(left*self.cell_size, top*self.cell_size, w, h)
            drone_surf = pygame.Surface((w, h), pygame.SRCALPHA)
            drone_surf.fill((color[0], color[1], color[2], 100))
            self.screen.blit(drone_surf, (drone_rect.x, drone_rect.y))

            # label in center
            label_str = str(i+1)
            label_surf= self.font.render(label_str, True, (255,255,255))
            label_rect= label_surf.get_rect(center=drone_rect.center)
            self.screen.blit(label_surf, label_rect)

    def draw_info_panel(self):
        px=self.grid_size*self.cell_size+10
        py=10
        pw=380
        ph=self.grid_size*self.cell_size

```

```python
        pygame.draw.rect(self.screen,GREY,(px,py,pw,ph))
        pygame.draw.rect(self.screen,BLACK,(px,py,pw,ph),2)

        title=self.title_font.render("Drone Placement Results",True,BLACK)
        self.screen.blit(title,(px+10,py+10))

        coverage_count=np.sum(self.coverage_grid)
        total_cells=self.grid_size*self.grid_size
        drone_count=self.current_drone+1

        lines=[
            f"Grid Size: {self.grid_size}x{self.grid_size}",
            f"Total Drones: {len(self.drone_positions)}",
            f"Placing Drone #: {drone_count}/{len(self.drone_positions)}",
            f"Final Reward: {self.final_reward:.3f}",
            f"Coverage: {coverage_count}/{total_cells}",
            f"Coverage %: {100.0*coverage_count/total_cells:.1f}%"
        ]
        offset=60
        for ln in lines:
            surf=self.font.render(ln,True,BLACK)
            self.screen.blit(surf,(px+10,py+offset))
            offset+=25

        chart_x=px+20
        chart_y=py+240
        chart_w=pw-40
        chart_h=150

        pygame.draw.rect(self.screen,WHITE,(chart_x,chart_y,chart_w,chart_h))
        pygame.draw.rect(self.screen,BLACK,(chart_x,chart_y,chart_w,chart_h),1)

        chart_title=self.font.render("Coverage Progress",True,BLACK)
        self.screen.blit(chart_title,(chart_x,chart_y-25))

        hist=self.coverage_history[:drone_count+1]
        if self.expanding and drone_count>0:
            if len(hist)>0:
                hist[-1]=coverage_count

        if len(hist)>1:
            maxcov=total_cells
            step_x=chart_w/(len(hist)-1)
            pts=[]
            for i,cov_val in enumerate(hist):
                frac=cov_val/maxcov
                pxp=chart_x+i*step_x
                pyp=chart_y+chart_h-(frac*chart_h)
                pts.append((pxp,pyp))
            pygame.draw.lines(self.screen,(255,0,0),False,pts,2)
            for i,pt in enumerate(pts):
                if i==0:
                    ccol=(0,0,255)
                else:
                    idx=i-1
                    if idx<len(self.drone_sizes):
                        s=self.drone_sizes[idx]
                        if s<5:
                            ccol=(0,0,255)
                        else:
                            ccol=(255,0,0)
                    else:
                        ccol=(255,0,0)
                pygame.draw.circle(self.screen, ccol,(int(pt[0]),int(pt[1])),5)

        legend_y=chart_y+chart_h+10
        pygame.draw.circle(self.screen,(0,0,255),(chart_x+15,legend_y),4)
        s1=self.font.render("Small (<5)",True,BLACK)
        self.screen.blit(s1,(chart_x+30,legend_y-8))

        pygame.draw.circle(self.screen,(255,0,0),(chart_x+120,legend_y),6)
        s2=self.font.render("Large (>=5)",True,BLACK)
        self.screen.blit(s2,(chart_x+135,legend_y-8))

        instructs=[
            "Space = play/pause auto-advance",
            "R = reset animation",
            "+ / - = speed up / slow down",
            "Esc = exit"
        ]
        sy=py+ph-110
        for line in instructs:
            sr=self.font.render(line,True,BLACK)
            self.screen.blit(sr,(px+10,sy))
            sy+=22

    def run(self):
        running=True
        time_acc=0.0
        while running:
            dt=self.clock.tick(30)/1000.0
            for e in pygame.event.get():
                if e.type==pygame.QUIT:
                    running=False
                elif e.type==pygame.KEYDOWN:
                    if e.key==pygame.K_ESCAPE:
                        running=False
                    elif e.key==pygame.K_SPACE:
                        self.animation_active=not self.animation_active
                    elif e.key==pygame.K_r:
                        self.reset_animation()
                    elif e.key in [pygame.K_PLUS,pygame.K_EQUALS]:
                        self.animation_speed=max(0.05,self.animation_speed-0.1)
                    elif e.key in [pygame.K_MINUS,pygame.K_UNDERSCORE]:
                        self.animation_speed=min(2.0,self.animation_speed+0.1)

            if self.current_drone<0 and not self.expanding and not self.animation_active:
                if len(self.drone_positions)>0:
                    self.place_next_drone()

            self.update_expansion(dt)

            if not self.expanding and self.animation_active:
                time_acc+=dt
                if time_acc>self.animation_speed:
                    time_acc=0.0
                    advanced=self.place_next_drone()
                    if not advanced:
                        self.animation_active=False

            self.draw_scene()
            self.draw_info_panel()
            pygame.display.flip()

        pygame.quit()


def run_visualization(results_file="drone_coverage_results.json", grid_size=None):
    animator=DroneAnimator(grid_size=grid_size,cell_size=50)
    if not animator.load_results(results_file):
        return
    animator.run()
```