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

```python
#!/usr/bin/env python
"""
Drone Coverage with Q-Learning (Center-based squares)
"""

import argparse
import os
import json

from train_agent import CONFIG, Q_learning_adaptive_limited, evaluate_policy
from testing import run_visualization

def parse_arguments():
    parser = argparse.ArgumentParser(description="Q-Learning for Drone Coverage (Center-based squares)")
    parser.add_argument("--train", action="store_true", help="Run Q-learning training")
    parser.add_argument("--visualize", action="store_true", help="Run coverage visualization")
    parser.add_argument("--grid-size", type=int, default=None, help="Override CONFIG['N']/'M']")
    parser.add_argument("--drones", type=int, default=None, help="Override CONFIG['max_drones']")
    parser.add_argument("--episodes", type=int, default=None, help="Override CONFIG['num_episodes']")
    return parser.parse_args()

def main():
    args = parse_arguments()

    # If no flags => do both training and visualization
    if not args.train and not args.visualize:
        args.train = True
        args.visualize = True

    if args.grid_size is not None:
        CONFIG["N"] = args.grid_size
        CONFIG["M"] = args.grid_size
    if args.drones is not None:
        CONFIG["max_drones"] = args.drones
    if args.episodes is not None:
        CONFIG["num_episodes"] = args.episodes

    results_file = os.path.join(os.getcwd(), "drone_coverage_results.json")

    # ------------------------------------------------------------------------
    # TRAIN
    # ------------------------------------------------------------------------
    if args.train:
```

```
        print("=" * 60)
        print("Training Q-learning with:")
        print(f"  Grid = {CONFIG['N']}x{CONFIG['M']}")
        print(f"  Max Drones = {CONFIG['max_drones']}")
        print(f"  Episodes = {CONFIG['num_episodes']}")
        print("=" * 60)

        # Train => returns best Q-table found
        Q_table = Q_learning_adaptive_limited(CONFIG)

        # Evaluate final => with some forced spawns and mild epsilon
        total_reward, final_obs = evaluate_policy(Q_table, CONFIG)
        final_drones = final_obs["drones"]
        obstacles = final_obs.get("obstacles",[])

        print(f"\n[INFO] Final Q-policy => reward: {total_reward:.3f}")
        print("[INFO] Drones:", final_drones)

        coverage_data = {
            "grid_size": CONFIG["N"],
            "final_reward": total_reward,
            "drone_positions": [],
            "drone_radii": [],
            "obstacles": obstacles
        }
        for (cx,cy,sz,act) in final_drones:
            coverage_data["drone_positions"].append((cx,cy))
            coverage_data["drone_radii"].append(sz)

        try:
            with open(results_file, "w") as f:
                json.dump(coverage_data, f, indent=4)
            print(f"[INFO] Coverage results saved => {results_file}")
        except Exception as e:
            print(f"[WARNING] Could not save => {e}")

    # ------------------------------------------------------------------------
    # VISUALIZE
    # ------------------------------------------------------------------------
    if args.visualize:
        if not os.path.exists(results_file):
            print(f"[ERROR] Results file not found: {results_file}")
            print("Either run with --train first or ensure the file exists.")
            return

        print("\nLaunching coverage visualization ...")
        run_visualization(results_file=results_file, grid_size=CONFIG["N"])

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