HW4: Implement CTDE method in MPE simple_spread environment

In this homework, you will implement Centralized Training with Decentralized Execution (CTDE) methods to train agents in the Multi-Agent Particle Environment (MPE) simple_spread scenario. The goal is to control multiple agents to cover all landmarks while avoiding collisions.

1 Environment Details

Agents:

- 3 agents by default
- All agents need to cover landmarks while avoiding collisions

Observation Space:

- 18-dimensional continuous space for each agent
- Includes: self velocity, self position, relative positions of landmarks, relative positions of other agents, communication

Action Space:

- 5 discrete actions for each agent: no action, move left, move right, move down, move up

Rewards:

- **Global reward**: Negative of the sum of minimum distances from agents to landmarks
- **Local penalty**: -1 for each agent collision
- The trade-off between these rewards can be controlled with the `local_ratio` parameter

> Note
>
> For detailed information, refer to: https://pettingzoo.farama.org/environments/mpe/simple_spread/

2 Tasks

Step 1: Environment Setup

- Install the Multi-Agent Particle Environment (MPE).

> Note
>
> For detailed information, refer to: https://github.com/openai/multiagent-particle-envs

Step 2: CTDE Implementation

- Choose and implement **ONE** of the following CTDE algorithms for discrete action spaces:
  - COMA (https://arxiv.org/pdf/1705.08926.pdf)
  - VDN (https://arxiv.org/pdf/1706.05296.pdf)
  - QMIX (https://arxiv.org/pdf/1803.11485.pdf)

Step 3: Results and Analysis

- Plot the learning curves (reward vs episode or timestep)
- Discuss findings or potential improvements, if there are any.

3 Code Demo

Here's a simple code example to get you started. You can also implement it in your own style.

```
from pettingzoo.mpe import simple_spread_v3
import numpy as np
import torch

# Create the environment
env = simple_spread_v3.env()

# This will differ based on your chosen CTDE method
for episode in range(num_episodes):
    env.reset()
    episode_rewards = []
    observations = {}
    for agent in env.agent_iter():
        observation, reward, termination, truncation, info = env.last()
        observations[agent] = observation
        if termination or truncation:
            action = None
        else:
            action = my_policy.select_action(agent, observation)
        env.step(action)
        episode_rewards.append(reward)

    my_policy.update(observations, episode_rewards)
```

4 Submission Requirements

Submit a ZIP file containing:

- Implementation code.
- A brief PDF report containing instructions for running your code, along with your results and discussions.

Submit to the course platform before **May 14, 2025, 23:59 PM**.