**First Attempt: DronePlacementEnv – Basic Single-Agent Exploration**

This initial environment focused on single-agent exploration using drones of two types (3×3 and 5×5 scanning range). The agent received rewards based on how many unexplored cells it revealed. The environment supported curriculum learning by training on a small grid first, then transferring encoder weights to a larger grid. This setup served as a proof-of-concept for map coverage and reward-based learning.

**Drawbacks:**

* Lacked realistic constraints like battery usage or communication;
* Could not support collaboration or distributed behavior;
* Exploration became saturated quickly, limiting the depth of learning.

A screenshot of a computer

AI-generated content may be incorrect.

**Second Attempt: AdvancedDroneEnv – Resource-Constrained Single Agent**

Building on the first version, this environment introduced realistic constraints such as battery capacity, high-priority cells, role types (explorer vs relay), and a basic connectivity check using communication range. The agent had to balance energy usage, prioritize certain areas, and maintain connection to the base station.

**Drawbacks:**

* Communication logic was overly simplistic (Manhattan distance only);
* Still single-agent—couldn’t leverage actual relay support;
* Rewards became sensitive and brittle under increased complexity.

A screenshot of a computer

AI-generated content may be incorrect.

**Third Attempt: DynamicLandingEnv + MultiDroneEnv – Multi-Agent Cooperative Exploration**

This final version implemented a multi-agent setup using the PettingZoo framework, where explorers and relays were treated as distinct agents. It supported dynamic obstacle flipping, safe landing zones, and prioritized cells. Training was parallelized with SuperSuit wrappers and vectorized environments, enabling more realistic mission-scale collaboration.

**Drawbacks:**

* Training became significantly slower and more computationally demanding;
* Shared reward across agents led to unstructured cooperative behavior;
* Relay logic remained simplistic and lacked real dependency modeling.

A screenshot of a computer program

AI-generated content may be incorrect.

While each version progressively improved in realism and challenge, they all shared some core limitations:

* **Single-agent architectures** couldn’t generalize to real-world team-based deployment;
* **Relay mechanisms** lacked true message-passing or network logic;
* **Reward design** became too entangled, affecting learning stability.

In the end, we decided to move toward **Graph Neural Networks (GNNs) and decentralized control policies** to better model agent roles, inter-agent communication, and spatial dependencies in a principled way.