Matthew Lewis

SEAD/DEAD Simulation Project

Project Introduction:

In a world of FPV drone warfare, and the trending buzz about drone swarms and other potential future technologies, this is my attempt at brainstorming a potential solution to real mission goals, and more importantly, creating a simulated battlespace environment to test and brainstorm different solutions in.

If the future is using economical, low production cost drone swarms, what is a practical application in which we can test this potential technology? Rather than thinking that drones can be a do-all solution, this is my attempt for planning, and simulating a drone swarm to fill a particular role.

For this project, let us imagine a mass manufactured, low cost fixed wing unmanned aerial vehicle. As a complete product, it has acceptable flight characteristics, comparable to what we have seen from Shahed UAV(s), flight control systems complete, and a maximum carry weight of 150kg.

The mission type I will try to approach is Suppression/Destruction of Enemy Air Defenses(SEAD/DEAD). This is a fairly typical mission type for any air force trying to fulfill its operational goals in any space, and a typical everyday need of an air force in conflict. The main goal of these missions is to locate and destroy land-based radar sites and surface to air missile sites, as well as other threats presented by the air defense systems of the adversarial force.

How can we most effectively use low-cost unmanned aerial vehicle(s) to fill this role?

This project will aim to create a simulation space where we can realistically test potential solutions, and allow for better brainstorming and iteration.

Solution and Core Concept:

To find the most efficient solutions, this project hopes to create a complete battlefield simulation system for SEAD/DEAD missions, and allow a user to iterate by trying out equipping these universal UAVs with different systems, which both change capabilities, and take up the limited weight of the system. There are universal constraints, such as the power of the motors, the size of control surfaces, or carry weight, which can be edited universally depending on the imagined drone design. There are also the modular systems which can be added to each drone or drone type, and they add capabilities and effects to the simulated UAV. In order to find the most efficient use, these systems can be given scripts or rules, and plugged into a neural network system, which will train iterations, and then eventually, the neural network system can be tested on different simulated battlespaces with different enemy behaviors or situations.

For my experiment, I started with a system where a swarm of single-role drones carried out mission actions as an autonomous swarm. By assigning each drone a role, cost and important components can be focused in a few important drones, and several drone frames could be used only for carrying explosive payload, and effectively be used as sacrificial tools for various roles they might be tasked for.

Roles:

Brain Drone:

This drone uses its weight to carry important computing and communication systems. While a few of these might be present for redundancy, one brain drone assumes control of the entire autonomous swarm at all times. It takes in information from all the drones, processes it, broadcasts it back to a friendly operator, and makes autonomous decisions about drone swarm movement and behavior, and transmits instructions back to the other drone in the swarm

Decoy Drone:

The decoy drone operates as a do it all drone for SEAD operations, and most importantly, contains a Signature Augmentation Subsystem (SAS), enabling it to simulate the presence of other aircraft. With several of these in a swarm, it could give the enemy the impression that instead of a drone swarm, there is a flight of strike aircraft: potentially tricking the enemy into turning on radar systems or attempting to engage the swarm from a SAM site, revealing their location for a follow-up strike. This drone shares many responsibilities with the Acquisition drone.

Acquisition Drone:

The acquisition drone focuses its payload on searching and finding sites of interest for destruction. When the decoy drone entices the enemy force into any type of action, it is the acquisition drone that has the computing power to locate potential sites, map their location, and present that information as a package to the Brain Drone

Sacrificial Drone:

The sacrificial drone has its payload focused on destructive potential and target tracking. After the brain drone decides which sacrificial drone is closest/best for a strike on a site, the sacrificial drone goes to the target site and destroys the site. The sacrificial drone could also be used in defensive operations, to intercept launched missiles or other threats, though the effectiveness of this is not known.

By utilizing a large swarm with several types of drones, we can distribute responsibilities and also end up with specialized drone units that give us the best capabilities per weight. A centralized neural controller was chosen over fully decentralized swarm intelligence in order to reduce training complexity, improve interpretability, and better reflect realistic command-and-control constraints. This mirrors real-world systems where high-level decision authority is centralized, while execution can be distributed across simpler autonomous agents.

Project Goals:

- Create a battlefield simulation
- Accurately simulate systems functions(radar,electronic transmissions)
- Accurately model enemy forces/sites
- Accurately simulate enemy behavior
- Create an enemy Command and Control AI
- Create enemy communications network
- Create scene randomizer
    - Once model trained, allows for randomization to test with new scenes
- Create tag system to tag friendly elements with different capabilities
    - Allow for iteration and testing
- Create friendly neural network system
    - Allows user to plug in capabilities into the neural network, or mess with weights

Scope Clarification and Non-Goals

This project intentionally abstracts or omits certain real-world systems in order to

prioritize iteration speed, interpretability, and tactical decision-making analysis. The simulation does not attempt to model full electromagnetic wave propagation, classified radar behavior, or real missile flight dynamics. Instead, systems are represented through parameterized abstractions that preserve causal relationships (detection, tracking, engagement, degradation) while remaining computationally tractable and debuggable.

These constraints are intentional and allow the simulation to function as a design and reasoning tool rather than a physics-perfect battlefield replica.

Project Elements Breakdown and Planning:

1. Create a Battlefield Simulation

The foundation of the entire project is a flexible battlefield simulation environment that can represent realistic SEAD/DEAD scenarios while still being modular enough for iteration and experimentation.

This will be implemented in Unity, leveraging its real-time 3D engine and physics system to simulate air, ground, and electromagnetic interactions. The terrain will be generated using Unity Terrain Tools, allowing for adjustable heightmaps, slopes, and elevation changes that directly affect radar line-of-sight, missile engagement envelopes, and flight paths. Terrain layers (urban, desert, forest) can be swapped to test different theaters of operation.

A skybox system will be added to represent different weather and lighting conditions (day/night, haze, cloud cover), which can later influence sensor effectiveness. Universal physics will rely on Unity's built-in physics engine, with custom scripts layered on top to approximate fixed-wing flight behavior, drag, lift, and turning radius constraints.

The goal here is not perfect aerodynamics, but a consistent, repeatable environment where tactical decisions have visible consequences.

2. Accurately Simulate System Functions (Radar, Electronic Transmissions)

A key part of SEAD/DEAD is not just kinetic effects, but the invisible battlespace of sensors and emissions.

Radar systems will be simulated using scripted detection cones and volumetric ranges, rather than full ray-traced radar physics. Each radar site will have configurable parameters such as range, sweep speed, frequency band, and detection thresholds. These will be visualized in debug mode using Unity Gizmos, making it easy to see why a drone was or was not detected.

Electronic transmissions (communication links, radar emissions, decoy signals) will be represented as abstracted signal objects with properties like power, signature type, and duration. Decoy drones with a Signature Augmentation Subsystem (SAS) will intentionally broadcast exaggerated radar signatures to emulate manned aircraft or strike packages.

This abstraction allows rapid iteration while still preservin**g the cause-and-effect relationships** central to electronic warfare decision-making.

3. Accurately Model Enemy Forces and Sites

Enemy air defense sites (radars, SAM batteries, command posts) will be modeled as modular prefabs rather than static scenery.

Each site will consist of:

- A physical structure (radar dish, launcher, control building)
- A sensor component
- A communication component
- A behavior controller

Using prefab-based design allows the same core logic to be reused across different threat levels. For example, an early-warning radar may prioritize long-range detection, while a SAM site focuses on fire-control tracking.

Damage modeling will be simplified but functional: destroying a radar dish disables detection, while destroying a control node disables coordination. This supports meaningful partial kills rather than all-or-nothing outcomes.

4. Accurately Simulate Enemy Behavior

Enemy behavior will be handled through a finite state machine (FSM) or behavior tree system, implemented through custom scripts.

Enemy units will transition between states such as:

- Passive monitoring
- Heightened alert
- Active engagement
- Emission control (radar off)
- Post-attack recovery

This allows the simulation to reflect real-world tradeoffs, such as turning on radar to engage a threat while simultaneously exposing the site to detection. These behaviors will be parameter-driven so different doctrines can be tested without rewriting logic.

This is critical for preventing “scripted” or predictable enemies and instead creating adversaries that react to player and AI-driven actions.

5. Create an Enemy Command and Control (C2) AI

Rather than each site acting independently, an enemy C2 AI will oversee the entire air defense network.

This system will aggregate sensor data from multiple sites, decide when to activate or deactivate radars, assign engagement priorities, and coordinate responses. Implementation-wise, this will exist as a centralized manager object that receives updates from individual sites and broadcasts decisions back down the chain.

This mirrors real integrated air defense systems (IADS) and enables emergent behaviors, such as sacrificing one radar site to preserve higher-value assets elsewhere.

6. Create an Enemy Communications Network

Enemy communications will be modeled as a network graph rather than point-to-point links.

Each site will have communication dependencies; destroying or jamming relay nodes will degrade the network's effectiveness. This introduces realistic vulnerabilities and allows SEAD tactics beyond simple destruction, such as isolation or confusion.

Latency, reliability, and redundancy can be abstracted as numerical modifiers rather than real-time packet simulation, keeping performance reasonable while preserving strategic depth.

7. Create a Scene Randomizer

To prevent overfitting both human and AI strategies, a scene randomizer will be implemented.

This system will procedurally place enemy sites, terrain features, and environmental conditions within defined constraints. Randomization seeds can be saved, allowing exact scenarios to be replayed for debugging or comparison.

Once neural networks are trained, this system becomes essential for stress-testing performance against unfamiliar layouts rather than memorized maps.

8. Create a Tag and Capability System for Friendly Units

Each friendly drone will use a tag-based system to define its role and capabilities.

Tags such as Sensor, Decoy, Explosive, Relay, or Brain allow behavior scripts and AI logic to dynamically adapt based on what assets are available. This makes swarm composition flexible and enables rapid experimentation without rewriting logic for each new drone type.

This also allows degraded operation — if a brain drone is destroyed, another tagged unit can assume partial control.

9. Allow for Iteration and Testing

A core design goal is fast iteration.

Inspector-exposed variables, debug overlays, and runtime toggles will allow parameters like sensor range, payload weight, or communication strength to be adjusted without recompiling

code. This supports rapid experimentation and makes the simulation useful as a thinking tool, not just a demo.

Data from each run can be logged for comparison across iterations.

10. Create a Friendly Neural Network System

The friendly swarm AI will eventually transition from scripted logic to a neural network-based decision system.

Initial implementations may use Unity-compatible ML frameworks (such as ML-Agents) to train behaviors like formation movement, decoy timing, and strike selection. Inputs will include sensor data, enemy emissions, and swarm status, while outputs will control movement vectors and role-specific actions.

Crucially, the system will allow manual tweaking of weights and capabilities, enabling hybrid control between human intuition and learned behavior.

Implementation

Tech Stack

In Unity

- **U**nity ML-Agents (main training framework)
    - Gives us Agents, Observations, Actions, reward hooks, training integration
- Unity Physics + custom flight controller scripts
    - The “motor” the policy drives (don’t train raw aerodynamics from scratch, train decisions)
- Gizmos + Debug overlays
    - Visualize detections, engagement zones, reward events, and agent decisions
- ScriptableObjects
    - Store scenario configs + swarm loadouts + randomized parameters cleanly

Outside Unity (training side)

- Python (ML-Agents trainers)
    - Training loop, hyperparameters, logging
- PyTorch (under the hood with ML-Agents)
- TensorBoard (training curves + debugging reward shaping)
- Export to ONNX (model format)
- Unity Barracuda (runtime inference in Unity using the trained ONNX model)

Neural Network Planning

NN Control

- One “Brain Drone” agent controls the swarm at the tactical level
- Other drones run:
    - either simple autopilot “follow this waypoint / formation slot”
    - or small role scripts that obey high-level commands

NN Action Space

High-level movement actions

- swarm_heading (continuous: -1 to 1 → maps to yaw change)
- swarm_speed (continuous: normalize between min/max cruise)
- altitude_bias (continuous: go lower/higher to break LoS)

Role-specific command actions

- decoy_emit_mode (discrete: Off / Low / High “aircraft emulate”)
- target_select (discrete: choose which known emitter / site is highest priority)
- strike_commit (discrete: yes/no)
- which_sacrificial_drone (discrete: select from available drones)

Optional

- Swarm formation
- “split swarm” action: keep together vs divide into 2 groups
- “jamming posture”: conserve vs aggressive

NN observation space

Fixed-size structured vector

Swarm state (friendly side)

- number of drones alive by role (brain/decoy/acquisition/sacrificial)
- average swarm position, velocity, heading
- fuel/energy proxy (optional)
- comms quality metric (if you simulate it)

Threat/environment state

- list of detected emitters (top N)
    - for each: relative bearing, distance, last-seen time, type (radar / SAM / unknown)
- “threat heat” features

    - nearest engagement zone distance
    - whether currently tracked/locked
- terrain LoS proxy
    - “is line-of-sight to nearest radar broken” (boolean-ish numeric)

Decoy effectiveness feedback

- last X seconds: did turning on SAS cause radar activation / tracking?
- how many enemy sensors turned on after decoy emit?

NN Rewards

Primary mission rewards

    - big reward for destroying radar/SAM sites (objective success)
    - medium reward for correctly localizing an emitter (acquisition success)
    - reward for causing enemy radar to activate (decoy success), but only if it leads to detection/localization

Penalty shaping

    - penalty per friendly drone lost (avoid suicide swarm)
    - penalty for being tracked/locked
    - penalty for wasting decoy emit time (optional)
    - penalty for time (encourage efficiency)

Anti-cheese rules

- If decoy just spams emit forever: add penalty for excessive emission time unless it produces useful results
- If agent hovers at edge of range: include time penalty + reward only on actual progress toward mission completion

Degraded Operation and Safety Fallbacks

- If brain drone is destroyed:
    - secondary brain drone assumes control
    - otherwise revert to scripted swarm behavior
- If comms quality drops below threshold:
    - restrict NN action space (no split-swarm, no deep penetration)
- If NN outputs invalid actions:
    - clamp actions
    - revert to last known safe command
- Optional: confidence threshold on NN output before executing irreversible actions (e.g., strike commit)

Performance Evaluation Metrics

- Mission success rate (% of runs with primary objectives achieved)
- Average time to radar localization
- Friendly drone loss ratio
- Cost-effectiveness proxy (objectives achieved per drone lost)
- Generalization score (performance drop between seen vs randomized scenarios)

Training Strategy

Use TensorBoard to observe training

Phase A: Navigation + survival

- enemy radars exist but do not fire
- objective: reach waypoint zones while minimizing detection

Phase B: Decoy / provoke behavior

- enemy reacts (radar on/off) based on decoy emission
- objective: cause radar activation AND keep swarm alive

Phase C: Full SEAD loop

- add acquisition + emitter localization
- add sacrificial strikes
- objective: destroy site(s) efficiently with acceptable losses

Phase D: Robustness training

- domain randomization:
    - site placements
    - radar ranges
    - comms degradation
    - terrain types