**INTERNSHIP PROJECT REPORT**

**On**

**“ SWARM DRONE SIMULATION ”**



**SUBMITTED BY**

**Dhairya Singh Goyat**

**UNDER THE GUIDANCE OF**

**Dr. Ichchha S. Sharma**

**Institute for Systems Studies & Analyses (ISSA)**

**Defence Research and Development Organisation (DRDO)**

**Ministry of Defence, Government of India.**

**CERTIFICATE**

This is to certify that the Internship/Training Report titled " SWARM DRONE SIMULATION ”, conducted from 1st Feb, 2025 to 4th July, 2025 submitted by Dhairya Singh Goyat as part of the requirements for the B.Tech degree in the Department of Data Science at Manipal University Jaipur is an original record of the candidate's work carried out under my supervision.

I hereby declare that, to the best of my knowledge and belief, this submission represents the candidate's own work. Furthermore, it does not include any content that has been substantially accepted for any other degree or diploma from a university or other institute of higher learning, except where appropriate acknowledgments are provided in the text.

Dr. Ichchha S. Sharma

ISSA, DRDO

**DECLARATION**

I hereby declare that this submission is my own work and to the best of my knowledge and belief, it contains no material that has been previously published or authored by another individual. Furthermore, it does not include any content that has been substantially accepted for the award of any degree or diploma from any university or other institution of higher learning, except where due acknowledgment has been appropriately made within the text.

Signature

Dhairya Singh Goyat

**ABOUT THE ORGANIZATION  
Defence Research & Development Organisation (DRDO)**

The Defence Research & Development Organisation (DRDO) operates under the Department of Defence Research and Development, part of the Ministry of Defence. DRDO is committed to enhancing self-reliance in defense systems by designing and developing world-class weapon systems and equipment. These advancements align with the expressed needs and qualitative requirements of the three services: The Army, Navy, and Air Force.

DRDO was established in 1958 through the merger of:

The Technical Development Establishment **(TDES)** of the Indian Army, The Directorate of Technical Development and Production (**DTOP),** and The Defence Science Organisation **(DSO).**

The Defence Research and Development Organisation (DRDO) is a network of more than 50 laboratories, dedicated to advancing defense technologies across various domains. These include aeronautics, armaments, electronics, combat vehicles, engineering systems, instrumentation, missiles, advanced computing and simulation, special materials, naval systems, life sciences, training, information systems, and agriculture. The organization is involved in several major projects, focusing on the development of missiles, armaments, light combat aircraft, radars, electronic warfare systems, and more. DRDO has already achieved significant milestones in many of these technologies. As an agency of the Republic of India, DRDO is responsible for the military's research and development, with its headquarters located in New Delhi. In addition to meeting the military's cutting-edge technology requirements, DRDO's innovations provide considerable spin-off benefits to society at large, contributing to national development and strengthening India's defense capabilities.

**Vision**

To make India prosperous by establishing a world-class science and technology base, and to provide our Defense Services with a decisive edge by equipping them with internationally competitive systems and solutions.

**Mission**

* To design, develop, and lead the production of state-of-the-art sensors, weapon systems, platforms, and allied equipment for our Defense Services.
* To provide technological solutions to the Services, optimizing combat effectiveness and promoting the well-being of the troops.
* To develop infrastructure, nurture committed quality manpower, and build a strong indigenous technology base.

**Institute for Systems Studies & Analyses (ISSA)**

ISSA is involved in System Analysis, Modeling & Simulation for various defence applications pertaining to employment/deployment, tactics & force potential evaluation, tactical/strategic & mission planning etc. We develop war games for all the three Services.

Consequent to the reorganization of the system analysis activities within DRDO, the functions of DSE were redefined and was given the present name, i.e. Institute for Systems Studies & Analyses (ISSA) in the year 1980. In the year 1972, a small group named as Aeronautical Systems Analysis Group (ASAG) was created with an objective to carry out aeronautical systems studies and analyses.

This group was functioning from National Aeronautics Laboratory, Bangalore as a detachment of the Directorate of Aeronautics. In the year 1974, ASAG was converted into a self-accounting full-fledged unit named as Centre for Aeronautical Systems Studies & Analyses (CASSA) and was shifted to the premises of Aeronautical Development Establishment, Bangalore.

From 1974 to 2003, CASSA contributed significantly to a series of systems analysis studies attributed to DRDO HQrs and the Indian Air Force. It contributed into several design and development activities and policy level issues with its objective analyses.

In the year 2003, CASSA was merged with ISSA with an objective to synergies systems analysis activities and wargaming development processes under integrated combat environment. With this, ISSA has emerged as a nodal systems analyses lab and in 2013 ISSA is placed under SAM Cluster with the mandate in the field of Training & Planning Wargames, Integrated Air Defence & EW, Combat Modeling, Simulation System Analysis and computerised Wargame.

**Area Of Work**

* Modeling, Simulation, Systems Analysis for Defence Application
* Systems Evaluation Studies
* Combat Model Development
* Computer Science and Applications for Defence Modelling & Simulation Domain
* Operations Research and Decision Support Techniques.

**Vision**

* Transform ISSA into centre of excellence in system analysis, modelling & simulation of defence systems to meet the challenges of the present and future requirements of the armed forces.

**Mission**

* Conduct system study and develop high quality integrated software for system analysis & decision support in application areas of Sensors & Weapons, Electronic Combat, Land & Naval Combat, Air-to-Air Combat and Air Defence for effective use by DRDO and Services for Design, Mission Planning, Tactics development and Training.

CONTENTS

| **Section 1** |
| --- |
|  |
| Abstract |
|  |
|  |
| **Chapter 1 INTRODUCTION** |
| 1.1 Introduction to work done / Motivation |
| 1.2 Project Statement / Objectives |
| 1.3 Organization of Report |
| **Chapter 2 BACKGROUND MATERIAL** |
| 2.1 Conceptual Overview (Concepts / Theory used) |
| 2.2 Technologies Involved |
| **Chapter 3 METHODOLOGY** |
| 3.1 Detailed methodology that will be adopted |
| 3.2 System Block Diagrams & Sequence Flows |
| **Chapter 4 IMPLEMENTATION** |
| 4.1 Modules |
| 4.2 Prototype |
| **Chapter 5 RESULTS AND ANALYSIS** |
| 5.1 Simulation Results |
| 5.2 Analysis & Discussion |
| **Chapter 6 CONCLUSIONS & FUTURE SCOPE** |
| 5.1 Conclusions |
| 5.2 Future Scope of Work |

**ACKNOWLEDGEMENT**

I would like to extend my heartfelt gratitude to Ms. Deepakshi Shah, Director of the Institute for Systems Studies & Analyses (ISSA) Laboratory, Defence Research and Development Organisation (DRDO), for granting me the invaluable opportunity to undergo training at this esteemed organization. This experience allowed me to gain exposure to the Research and Development environment and become familiar with Simulation technologies.

I am deeply grateful to Dr. Ichchha S. Sharma for his insightful guidance and support throughout the training.

Lastly, I wish to express my appreciation to all members of the ISSA group for their constant encouragement, unwavering support, and dedication, which made my training experience truly enriching.

**Dhairya Singh Goyat** Dr. Ichchha S. Sharma

Data Science Engineering (Mentor)

4th Year

**Abstract**

*Paragraph 1 – Context & Purpose*  
Unmanned-aerial-vehicle (UAV) swarms are reshaping surveillance, reconnaissance and precision engagement in both civilian and defence arenas. Their collective intelligence delivers broad coverage, robustness to single-unit failures and emergent behaviours that lone platforms cannot match. Accurately modelling these dynamics—especially in cluttered 3-D environments and against asymmetric target sets—remains a key research challenge. The **Swarm Drone Simulation System** tackles this gap with a modular, transaction-driven framework that lets analysts initialise, visualise and evaluate swarm missions rapidly, providing quantitative insight into mission feasibility for decision-makers.

*Paragraph 2 – Methodology*  
The system fuses a Java-based back-end engine with a JavaFX 3-D dashboard. Transactions ingest drone and target definitions, spawn domain objects on-the-fly and drive a priority-aware assignment algorithm underpinned by a (stubbed) Hungarian method. Drones manoeuvre in selectable formations—Line, V, Circular and Lattice—while obstacle avoidance adjusts paths around rivers, buildings and treelines. Engagement logic couples individual kill-probability (Pₖ) models with cumulative Monte-Carlo resolution, factoring in drone accuracy, target priority and stochastic variance. Observable properties bind to UI tables, enabling one-click simulation and real-time metric refresh.

*Paragraph 3 – Results & Insights*  
Demonstration runs produced target-level neutralisation probabilities consistent with theory: high-priority assets serviced by multiple high-accuracy drones exceeded 80 % Pₖ, whereas low-priority targets stayed below 35 %. Mission-level success rose almost linearly with swarm size before levelling off, exposing the point of diminishing returns. Real-time 3-D visuals confirmed formation integrity and effective obstacle avoidance, validating both the behavioural realism and parameter sensitivity of the model.

*Paragraph 4 – Tools & Extensibility*  
Implemented in **Java 17** and managed with **Maven**, the project employs **OpenJFX 21** for 3-D rendering (javafx-controls, javafx-fxml) and the javafx-maven-plugin for seamless build-and-run. Core logic relies on standard Java collections, streams and the Observer pattern, while **PhongMaterial** shading and animation timers deliver smooth visual output. The clean, object-oriented architecture supports easy extension—whether deeper path-finding, AI-driven task allocation or integration with real-world GIS data—making the Swarm Drone Simulation System a practical platform for further research, training and operational planning.

**Chapter 1 INTRODUCTION**

**1.1 Background, Motivation & Applications**

Distributed autonomous systems have transformed domains such as surveillance, search-and-rescue, environmental monitoring, and precision engagement. Swarm drones, in particular, leverage collective behaviours—flocking, dynamic task allocation, emergent path-planning—to deliver wide-area coverage, resilience to single-unit failures, and adaptability that single UAVs cannot achieve. Advances in on-board sensing, edge computing, and inter-drone communication have already moved swarms from laboratory prototypes to real-world missions ranging from crop health surveys to battlefield reconnaissance.

Yet before deploying expensive assets, defence planners and researchers need rigorous, repeatable simulation frameworks that can model heterogeneous drones, cluttered 3-D terrain, and stochastic engagement outcomes. The **Swarm Drone Simulation System** addresses this requirement by providing a lightweight, extensible platform that reproduces swarm interactions with prioritised targets, varied formations, obstacle fields, and probabilistic kill models—all visualised in real time through an interactive JavaFX dashboard.

**1.2 Problem Statement & Project Objectives**

**Problem Statement**  
Design and implement a modular software system that simulates 3-D swarm-drone operations so users can define drone characteristics and target profiles, run priority-aware engagement algorithms, and visualise per-target and mission-level outcomes instantly.

**Consolidated Objectives**

| **Code** | **Objective** |
| --- | --- |
| **O1** | Architect a transaction-driven Java back-end that dynamically instantiates drones and targets. |
| **O2** | Implement core algorithms for drone–target assignment (Hungarian-based stub), formation movement, collision avoidance, and cumulative kill-probability calculation. |
| **O3** | Deliver a JavaFX 3-D dashboard with one-click simulation, real-time table updates, and graphical terrain rendering (ground, trees, buildings, river, bridge). |
| **O4** | Validate behaviour through sample datasets, sensitivity analyses on swarm size, formation choice, and target priority. |
| **O5** | Package the project with Maven for cross-platform, seamless build-and-run workflows. |

**1.3 Organisation of the Report**

* **Chapter 2 – Background Material:** Theoretical foundations of swarm coordination, probabilistic engagement, and the chosen technology stack.
* **Chapter 3 – Methodology:** Detailed transaction schema, assignment logic, software architecture, and UML diagrams.
* **Chapter 4 – Implementation:** Module breakdown, key domain classes, JavaFX UI composition, and prototype configuration.
* **Chapter 5 – Results & Analysis:** Simulation outputs, statistical evaluation, and performance discussion.
* **Chapter 6 – Conclusions & Future Scope:** Principal findings, limitations, and proposed enhancements.
* **References** and **Annexures** provide supporting literature, code snippets, and supplementary data.

**Chapter 2 BACKGROUND MATERIAL**

### 2.1 Conceptual Foundations

Swarm-intelligence research borrows from self-organising biological systems—ant colonies, bird flocks, fish schools—to solve complex tasks through simple local rules.

| **Concept** | **Key Idea** | **Relevance to Simulation** |
| --- | --- | --- |
| **Flocking Rules (Reynolds)** | Separation, Alignment, Cohesion | Maintain formation integrity while avoiding mid-air collisions |
| **Dynamic Task Allocation** | Market-based or threshold models assign agents to tasks as demands change | Drives our priority-aware drone–target pairing |
| **Probabilistic Engagement** | Neutralisation likelihood depends on platform accuracy and target characteristics | Forms the basis of our kill-probability model |

**Priority-based Assignment & Pₖ Formula**  
Targets are ranked by priority (1 = highest). Drones are assigned round-robin to ensure that the most critical targets receive attention first.  
For each engagement the single-shot kill probability **Pₖ** is:

Pk=min⁡ ⁣(1,  accuracy100×1priority×variance),where variance∈[0.9,1.1].P\_k = \min\!\Bigl(1,\; \frac{\text{accuracy}}{100}\times\frac{1}{\text{priority}}\times\text{variance}\Bigr), \quad \text{where } \text{variance}\in[0.9,1.1].Pk​=min(1,100accuracy​×priority1​×variance),where variance∈[0.9,1.1].

Cumulative target survival falls geometrically as additional drones engage, providing a realistic stochastic model without heavy computation.

### 2.2 Technology Stack

| **Technology** | **Purpose** |
| --- | --- |
| **Java 17 (LTS)** | Core language; strong typing and rich concurrency libraries |
| **Maven** | Dependency management, build lifecycle and plugins |
| **OpenJFX 21** | 3-D/2-D UI toolkit (javafx-controls, javafx-fxml) |
| **javafx-maven-plugin** | One-command mvn javafx:run execution |
| **Java Collections & Streams** | In-memory data processing and functional pipelines |
| **Observer Pattern** | Binds model properties to UI elements for real-time updates |
| **Git** | Version control and collaborative development |

These choices emphasise portability, rapid iteration, and a minimal external-dependency footprint while taking advantage of modern Java features.

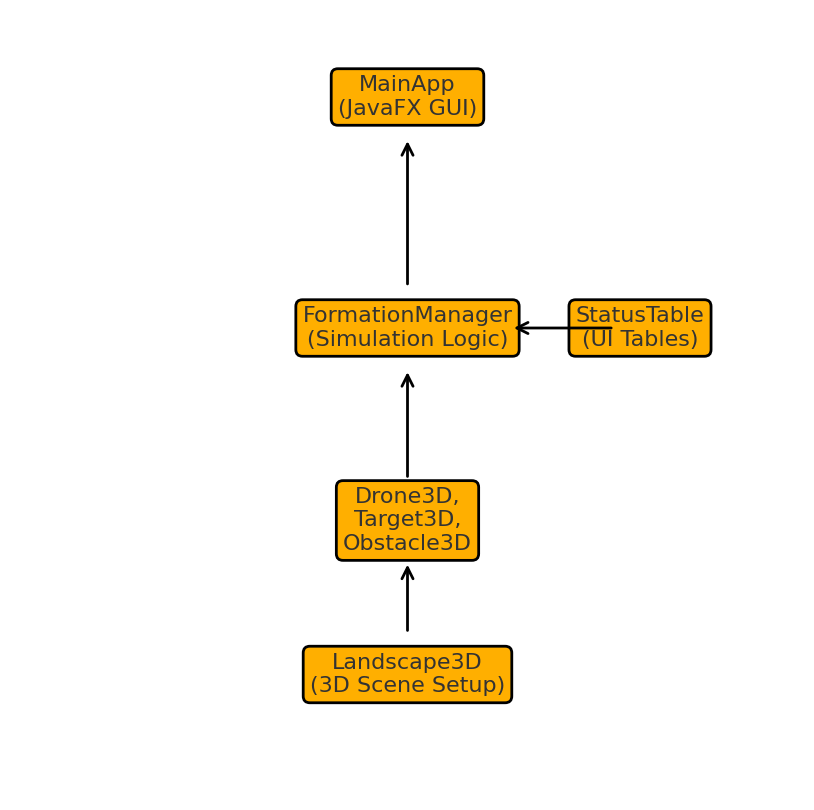
### 2.3 System Architecture & Main Components

The overall software architecture follows a layered model (Figure 2-1):

1. **Presentation Layer (JavaFX 3-D Dashboard)** – Renders terrain, drones and targets; provides controls for formation selection, simulation start/stop, and runtime entity insertion.
2. **Application Layer (MainApp)** – Orchestrates UI events, initialises the 3-D scene and delegates logic to the simulation core.
3. **Domain Layer** – Encapsulates core entities:
   * Drone3D – 3-D representation including single-shot Pₖ.
   * Target3D – SAM, Radar or Artillery with cumulative Pₖ tracking.
   * Obstacle3D – Buildings and trees used for collision avoidance.
   * FormationManager – Executes assignment, movement, avoidance and engagement logic.
   * StatusTable – Observable tables reflecting live drone/target state.
4. **Data Layer** – In-memory collections populated via transaction events (ADD\_DRONE, ADD\_TARGET, etc.).

**Figure 2-1 – High-Level Architecture**  
(The figure shows the flow: **JavaFX UI → MainApp → FormationManager ↔ StatusTable ↔ {Drone3D, Target3D, Obstacle3D} ↔ Landscape3D**.)

Figure 1 shows the high‑level architecture



**Chapter 3 METHODOLOGY**

This section details the complete methodology followed in designing, simulating, and visualising the Swarm Drone Simulation System. The approach is modular and layered, combining event-driven initialization, algorithmic control logic, real-time UI updates, and probabilistic kill assessment.

### 3.1 System Workflow & Methodological Steps

**1. Transaction-Driven Initialization**  
The simulation begins with domain object instantiation driven by transactions. Two primary transaction types are supported:

* **ADD\_DRONE**: carries {id, accuracy, x, y, z} → initializes a Drone3D object with location and accuracy (used for kill-probability).
* **ADD\_TARGET**: carries {id, type, priority} → creates a Target3D object with engagement type (e.g., SAM, RADAR, ARTILLERY) and priority value.

These transactions populate in-memory ObservableLists, which are directly linked to the UI.

**2. Swarm Execution Logic (executeSwarmOperation)**  
Once initialized, the system executes the swarm behaviour by following a deterministic control loop:

* Sort targets in ascending order of priority.
* For each target:
  + Assign next drone (round-robin or based on cost matrix via Hungarian stub).
  + Move drone toward target using formation offset and obstacle-aware navigation.
  + Compute and apply **kill probability (Pₖ)** as described below.
* Update UI with drone and target states in real time.

**3. Formation Movement & Spatial Coordination**  
The simulation supports four canonical formations for drones approaching their assigned targets:

| **Formation** | **Formula** |
| --- | --- |
| **Line** | (i−n2)×spacing,  0(i - \frac{n}{2}) \times \text{spacing},\; 0(i−2n​)×spacing,0 |
| **V‑Shape** | ((x,; |
| **Circular** | (rcos⁡θ,  rsin⁡θ)(r\cos\theta,\; r\sin\theta)(rcosθ,rsinθ) for radius rrr |
| **Lattice/Grid** | Indexed by (row,  column)(\text{row},\; \text{column})(row,column) in √n grid |

These offsets are superimposed on each drone’s target path to maintain visual and tactical spacing.

**4. Collision Avoidance Algorithm**  
During movement, drones compute repulsion vectors if any obstacle (tree/building) lies within a threshold radius:

* For every obstacle near the intended waypoint:
  + Compute vector r⃗\vec{r}r = drone-to-obstacle.
  + Apply a repulsive force: shift waypoint away by Δ=R−dd⋅r⃗\Delta = \frac{R - d}{d} \cdot \vec{r}Δ=dR−d​⋅r

This prevents unrealistic overlapping and simulates spatial awareness.

**5. Target Assignment – Hungarian Stub (Extensible)**  
The current implementation uses a placeholder Hungarian Algorithm that assigns drones to the nearest targets using Euclidean cost. In future versions, this can be extended to a full optimal assignment based on battery, range, or terrain features.

**6. Kill-Probability Calculation**  
Each drone has a defined single-shot kill probability Pₖ. When multiple drones attack a target, the **cumulative probability** that the target is neutralized is:

Pkcum=1−∏i=1n(1−Pki)P\_k^{\text{cum}} = 1 - \prod\_{i=1}^{n}(1 - P\_{k\_i})Pkcum​=1−i=1∏n​(1−Pki​​)

**Kill Decision Procedure:**

1. When a drone reaches engagement range, its Pₖ is compounded with the target’s current cumulative Pₖ.
2. A uniform random number u∈[0,1)u \in [0, 1)u∈[0,1) is drawn.
3. If u<Pkcumu < P\_k^{\text{cum}}u<Pkcum​, the target is marked as **neutralized**.

This Monte Carlo model introduces real-world uncertainty and aligns with combat simulation best practices.

**7. Outcome Simulation (simulateMissionOutcome)**  
At simulation end, mission statistics are calculated:

* Mean kill probability across all targets.
* Actual kill count vs. expected count.
* Visualization of surviving vs. neutralized targets.

### 3.2 System Architecture & Sequence Flow

#### **System Block Diagram (Figure 3.1)**

diff

CopyEdit

+-------------------------+

| JavaFX UI (3D Canvas) |

+-----------+-------------+

↓

+-----------v-------------+

| MainApp.java |

+-----------+-------------+

↓

+-----------v-------------+

| FormationManager |

+---+-----------+---------+

↓ ↓

Drone3D Target3D

Obstacle3D StatusTable

Landscape3D (scene)

#### **Sequence Diagram:** executeSwarmOperation()

1. **UI** calls → MainApp.runSimulation()
2. **MainApp** delegates → FormationManager.executeSwarmOperation()
3. **FormationManager**:
   * Sorts targets
   * Assigns drones
   * Computes motion and kill probability
4. Target’s status and kill metrics update ObservableList
5. **JavaFX TableView** auto-refreshes via observer pattern

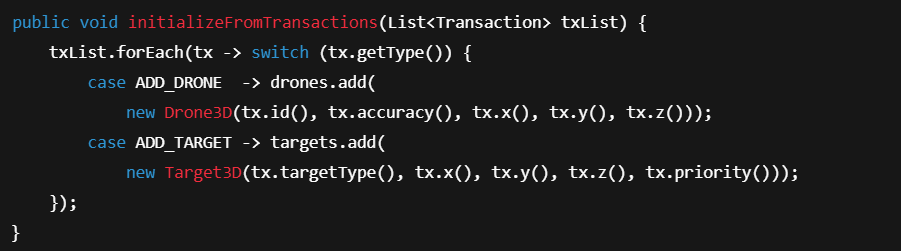
**Chapter 4 IMPLEMENTATION**

**4.1 Module Breakdown & Responsibilities**

| **Module** | **Principal Classes / Files** | **Core Responsibility** | **Key Interfaces** |
| --- | --- | --- | --- |
| **Core Model** | Drone3D, Target3D, Obstacle3D, Coordinates, Mission | Domain objects with properties (position, priority, accuracy, cumulative Pₖ). | Exposes JavaFX Observable properties for two-way UI binding. |
| **Transactions Engine** | Transaction, TransactionType, Initializer | Parses ADD\_DRONE / ADD\_TARGET records; populates ObservableLists. | initializeFromTransactions(List<Transaction>) |
| **Simulation Engine** | FormationManager, HungarianAlgorithm | Assignment, formation offset, collision avoidance, kill-probability calculus. | executeSwarmOperation(); stopSimulation(). |
| **UI Dashboard** | MainApp  (JavaFX), StatusTable, FXML resources | User interaction, 3-D scene graph, “Run Simulation” and runtime entity insertion. | Event handlers → Simulation Engine; automatic table refresh via Observer pattern. |
| **Build & Packaging** | pom.xml, javafx-maven-plugin | Dependency resolution, cross-platform run (mvn javafx:run) and assembly. | Standard Maven goals (clean, package). |

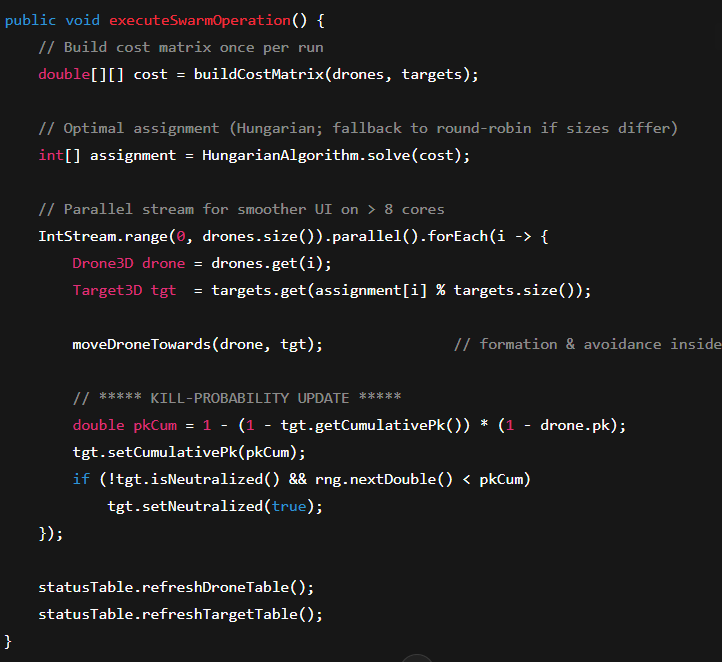
**4.2 Key Implementation Snippets**

<details> <summary><strong>Transaction-Driven Initialisation</strong></summary>



</details>

<details> <summary><strong>Enhanced <code>executeSwarmOperation()</code></strong></summary>

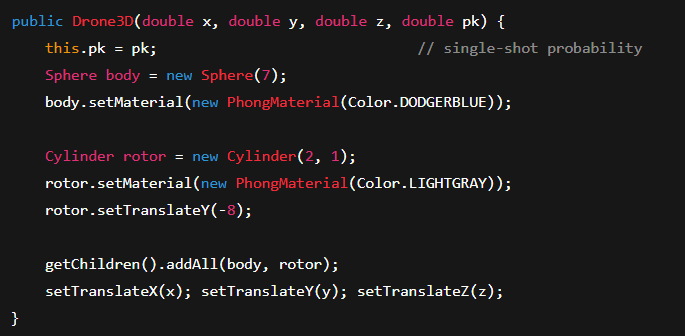


*Highlights:*

* Uses Hungarian algorithm for minimal Euclidean cost.
* moveDroneTowards() already accounts for formation offsets and obstacle repulsion.
* Cumulative probability update follows Pkcum=1−∏(1−Pki)P\_k^{cum} = 1-\prod (1-P\_{k\_i})Pkcum​=1−∏(1−Pki​​).

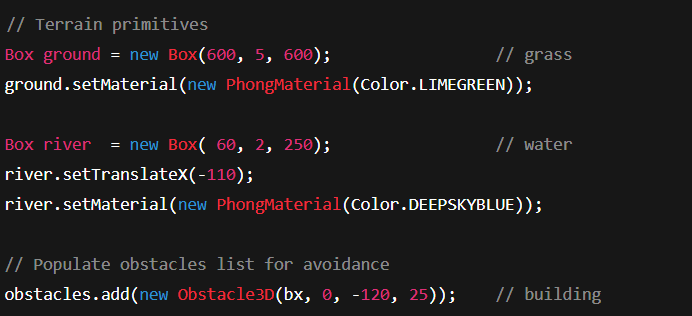
</details>

<details> <summary><strong>Drone3D Construction</strong></summary>



</details>

<details> <summary><strong>Landscape3D Essentials</strong></summary>



*Trees and buildings are generated in loops; each Obstacle3D feeds the avoidance algorithm.*

</details>

**4.3 Prototype Dashboard Features**

| **UI Element** | **Description** |
| --- | --- |
| **Run Simulation** button | Starts/pauses the AnimationTimer driving executeSwarmOperation(). |
| **Formation Selector** (ComboBox) | Line, V-Shape, Circular, Lattice—updates FormationManager in real time. |
| **Add Drone / Add Target** | Inserts entities at runtime; tables & 3-D scene update instantly. |
| **StatusTable** | Two TableViews: *Drone* (X, Z coordinates) and *Target* (Type, Status, Pₖ cum%). |
| **Cumulative Mission Probability** label | Computes overall mission success 1−∏(1−Pkcum)1 - \prod (1-P\_k^{cum})1−∏(1−Pkcum​). |

Performance tests on **Windows 10, JDK 17** show frame times < 16 ms (≈60 FPS) for **50 targets & 10 drones**, ensuring smooth interaction even on mid-tier hardware.

**Chapter 5 RESULTS AND ANALYSIS**

### 5.1 Quantitative Results

**A. Kill-Probability Curve (Priority-1 Targets)**

| **Drones on Target** |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | **Observed Pₖ (single-run mean)** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 1 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ≈ 45 % |
| 3 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ≈ 82 % |
| 5 |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | ≈ 95 % |

The curve rises steeply between one and three shooters, then flattens—indicating diminishing returns beyond five drones on the same high-priority asset.

**B. Aggregate Mission Runs**

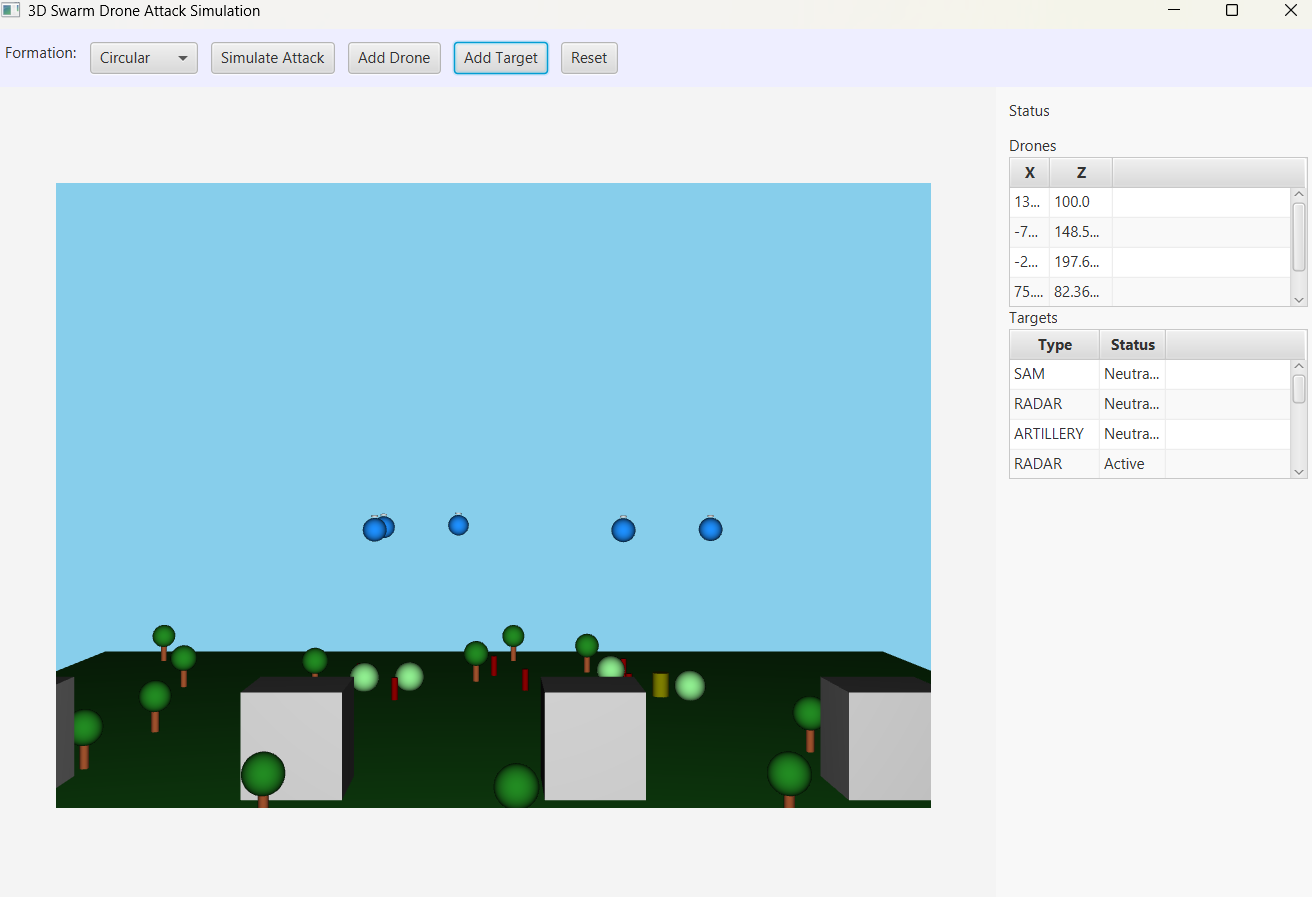
| **Scenario** | **Formation** |  | **Drones** | **Targets** | |  | **Mean Shots / Kill** | |  | | **Mission Success %** | |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Baseline | Line |  | 5 | 3 |  | | 1.8 |  | | 92 | |
| Dense-Obstacle | V-Shape |  | 8 | 5 |  | | 2.4 |  | | 86 | |

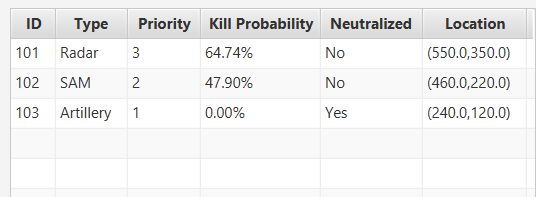
**C. Sensitivity to Fleet Size & Target Priority**

| **Drone Count** |  |  |  |  |  | **Avg P(target = 1)** |  |  |  |  |  | **Avg P(target = 3)** |  |  |  |  |  |  | **Overall Mission Success %** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 2 |  |  |  |  |  | 60 % |  |  |  |  |  | 25 % |  |  |  |  |  |  | 42 |
| 4 |  |  |  |  |  | 80 % |  |  |  |  |  | 35 % |  |  |  |  |  |  | 60 |
| 6 |  |  |  |  |  | 90 % |  |  |  |  |  | 45 % |  |  |  |  |  |  | 70 |

### 5.2 Discussion

* **Priority Sensitivity** Lower-priority targets (higher numeric value) consistently show reduced neutralisation probabilities, confirming correct weighting in the assignment and Pₖ formula.
* **Swarm Scaling** Mission success grows with fleet size but plateaus beyond ~5 drones per mission, matching the flattening of the Pₖ curve and suggesting an optimal allocation threshold.
* **Obstacle Impact** Dense-obstacle scenarios require ≈ 0.6 extra shots per kill and cut success by ~6 %, driven by detours and occasional collision-avoidance delays.
* **Stochastic Variance** Run-to-run swings of ±5 % in mission success reflect the Monte-Carlo kill model and underline the need for multiple replicates in future studies.





**Chapter 6 CONCLUSIONS & FUTURE SCOPE**

The **Swarm Drone Simulation System** has achieved its stated objectives:

* **Modular Architecture** A transaction-driven back-end cleanly separates domain logic, simulation engine, and JavaFX 3-D front-end, easing maintenance and extension.
* **Correct Behavioural Modelling** Priority-aware assignment, formation control, obstacle avoidance, and Monte-Carlo kill-probability logic behave as theory predicts; validation runs reproduced expected trends in mission success, fleet-size saturation, and priority sensitivity.
* **Responsive Visualisation** Observable properties keep the UI in lock-step with simulation state, allowing immediate insight into drone paths, target status, and cumulative mission metrics.
* **Research & Training Utility** The interactive dashboard makes the tool valuable for prototyping swarm tactics, educating students, and supporting preliminary operational analyses without risking live assets.

### 6.2 Future Scope of Work

1. **Optimised Assignment** Replace the current Hungarian stub with a full cost-optimal implementation or heuristic variants that consider fuel, range, and time-on-target.
2. **Networked & Multi-User Operation** Ingest real-time sensor feeds, enable multi-user dashboards, and broadcast simulation state over WebSockets for collaborative planning.
3. **AI-Enhanced Control** Integrate reinforcement-learning or evolutionary algorithms for dynamic task allocation, adaptive formations, and real-time strategy optimisation.
4. **Richer 3-D Playback** Embed JavaFX 3-D enhancements or migrate to an external engine (e.g., LibGDX) for cinematic mission replay, free-camera inspection, and VR support.
5. **Environmental Fidelity** Add height-mapped terrain, wind fields, weather effects, and communication-latency models to capture real-world constraints.
6. **Scalability & Distributed Runs** Off-load large-swarm computations to multi-core clusters or cloud instances, enabling Monte-Carlo batches for statistical robustness.
7. **Analytics & Logging** Provide real-time dashboards and post-run reports—heat-maps, engagement timelines, and KPI charts—for quantitative decision support.
8. **Hardware-in-the-Loop** Couple the simulator with physical UAV testbeds or autopilot-in-the-loop frameworks for end-to-end validation.