# 🛰️ Project Status Report: Anti-Aircraft Radar Simulation

Project Team: Muhammad Umar Aslam, Muhammad Awab Abdullah, Abdul Rehman Asif

Date: October 2025

Post Phase 1 Review

## 1. Executive Summary

The Anti-Aircraft Radar Software Simulation aims to develop a real-time system capable of **detecting, tracking, identifying, and providing firing solutions** for airborne targets. Development is structured around an **11-Phase Agile Roadmap** to ensure systematic complexity management and a modular final product.

The foundation is strong: **Phase 1: Mathematical Foundation** has been successfully completed. This crucial milestone establishes the core geometric and trigonometric capabilities required for all subsequent movement and fire-control calculations. Remaining efforts will focus on integrating real-time kinematics, I/O, advanced data structure implementation, and final system validation.

## 2. Core System Capabilities and Functional Breakdown

The final application will be structured around four primary, decoupled modules, defining the system's core defensive functionality.

### 2.1. Detection 📡

* **Goal:** Identify unknown objects entering the radar's designated range.
* **Principle:** Uses spatial algorithms (e.g., **linear search**) to manage detection events, logging when a target enters or leaves the defined radius.

### 2.2. Tracking 🏃

* **Goal:** Calculate and update real-time kinematics (**speed, acceleration**) of the target object.
* **Principle:** Employs the **Finite Difference Method** ($\Delta x/\Delta t$, $\Delta v/\Delta t$) to compute movement based on current and historical state data, relying on accurate, real-time mouse input (Phase 7).

### 2.3. Fire Control 💥

* **Goal:** Compute the precise trajectory required for the ground-based anti-aircraft gun to achieve an interception.
* **Principle:** Employs advanced **Trigonometric Firing Solutions** to determine the required elevation angle and the azimuth direction.

### 2.4. Identification (IFF System) 🤝

* **Goal:** Differentiate between friendly and hostile/unknown targets.
* **Principle:** Utilizes highly efficient **Hash Table Lookups** ($O(1)$ complexity) to check an object's ID against a list of known friendly signatures, minimizing latency in mission-critical decision-making.

## 3. Project Roadmap Status (11 Phases)

The project follows a linear progression, ensuring mathematical and logical components are solidified before interface integration begins.

| **Phase** | **Milestone Summary** | **Core Deliverable** | **Status** |
| --- | --- | --- | --- |
| **Phase 1** | Mathematical Foundation | Distance, direction, and angle conversions. | ✅ **COMPLETE** |
| **Phase 2** | Kinematics Reporting | Single object correctly updates speed/acceleration. | ⏳ Pending |
| **Phase 3** | Range Detection | Console logs show target entering/leaving range. | ⏳ Pending |
| **Phase 4** | Firing Solution | Gun module outputs directional results (e.g., "$30^\circ$ NE"). | ⏳ Pending |
| **Phase 5** | Real-time Terminal UI | Text-only real-time output display. | ⏳ Pending |
| **Phase 6** | Live ASCII Radar | Live ASCII radar scope updates every frame. | ⏳ Pending |
| **Phase 7** | Mouse Input | Mouse movement updates target position/kinematics. | ⏳ Pending |
| **Phase 8** | IFF System | Radar differentiates friendlies from unknowns. | ⏳ Pending |
| **Phase 9** | Modular Core | Clean core logic separation (GUI-ready architecture). | ⏳ Pending |
| **Phase 10** | Unit Testing | All code sections pass defined unit tests. | ⏳ Pending |
| **Phase 11** | Final Build | One-command build and run script implemented. | ⏳ Pending |

## 4. Phase 1 Technical Implementation Details

This section details the specific structures and concepts used to achieve the Phase 1 milestone.

### 4.1. Milestone Achievement:

* **You can calculate distance, direction, and convert angles with confidence.**

### 4.2. Key Data Structures Used:

| **Data Structure** | **Implementation Details** | **Purpose** |
| --- | --- | --- |
| **Structs/Classes** | Basic definition of the Vector2D and Target classes (likely within include/radar/Target.hpp and src/radar/Target.cpp). | Used as Abstract Data Types (ADTs) to encapsulate the position and state of an object. |
| **Arrays (Coordinates)** | Use of simple C-style arrays (or basic language arrays/vectors) to hold 2D coordinates (e.g., [x, y] or [horizontal\_dist, height]). | Stores the positional components (coordinates) of the radar/gun and the target. |

### 4.3. Core Algorithms and Concepts:

* **Position and Distance Calculation:** Achieved using the **Euclidean Distance Formula**:
* **Direction and Angle Conversion:** Achieved using **Trigonometric Angle Conversion** via the **arctan function** to correctly determine azimuth across all four quadrants.
* **Memory Management:** **Pointer Arithmetic** is used internally within the Target and Vector2D implementations for efficient access and manipulation of coordinate components stored in memory.

## 5. Architectural and Technical Strategy

### 5.1. Key Data Structures Plan

The strategic choice of data structure is driven by performance requirements, prioritizing **O(1) lookup speed** and **real-time state management**.

| **Data Structure** | **Primary Application** | **Phase(s)** | **Performance Benefit** |
| --- | --- | --- | --- |
| **Arrays / Structs** | Coordinate storage, Vector definitions, Target state | All | Foundational, high-speed access to fixed data. |
| **Hash Tables** | IFF System (Friendly/Unknown classification) | 8 | $O(1)$ average time complexity for lookups. |
| **Queues & Stacks** | Managing detection events, screen refresh operations | 3, 6 | Efficient **FIFO/LIFO** control flow for sequential processing. |
| **Linked Lists** | Storing historical target states for kinematics | 2 | Efficient dynamic **insertion/deletion** of state history. |

### 5.2. Modular Architecture (Phase 9 Readiness)

The project adheres to a strict modular design, ensuring the core logic is completely decoupled from the interface layer. This is essential for achieving Phase 9 and simplifying future migration to a GUI.

| **Module Directory** | **Responsibility** | **Status/Goal** |
| --- | --- | --- |
| src/radar/ | **Simulation Core:** Pure logic for Radar, Target, and Gun classes. | MUST contain zero printing or I/O calls. |
| src/ui/ | **Interface Layer:** Handles all terminal I/O (ConsoleUI) and system interaction (MouseInput). | Completely separate from radar/ logic. |
| include/ | **Header Mirror:** Provides public interface headers for all modules. | Ensures clean linking and compilation. |
| gui/ | **Future GUI:** Placeholder for GUI libraries (e.g., Dear ImGui). | Allows easy integration post-Phase 9. |
| tests/ | **Quality Control:** Contains all unit tests (test\_target, test\_gun). | Validates correctness for Phase 10 milestone. |