PERFORMANCE MEASUREMENT REPORT

TEB1113 – Algorithm and Data Structure

Sept. 2024

Submitted by

24000182	NURUL HANIIZATI BINTI HAZLI
24000574	SARA EUDORA BINTI SAID
22006373	DANIA ADRIANA BINTI MOHD FAIZAL
24000918	ZULAIKHA BINTI MOHD AZHAR
24000227	NUR FATIHAH BINTI MOHD NOOR

Submitted to

Dr M Nordin B Zakaria

CIS Department
Universiti Teknologi PETRONAS, Malaysia

Introduction

Drones are increasingly used in various fields, including logistics, agriculture, and search-and-rescue operations. Effective communication within a swarm of drones is essential for tasks like coordination, message relaying, and obstacle avoidance. This project focuses on simulating a drone swarm communication network, represented as a complete graph within two partitions, where drones can relay messages, search for other drones, and compute shortest communication paths.

The simulation leverages Unity for visualization and interaction, incorporating graph traversal algorithms and user interface elements for an intuitive experience. Performance analysis, including frame rate measurements, evaluates the scalability of the simulation under varying drone counts.

Objectives

- Develop a Graph-Based Communication Network: Design a DroneNetworkCommunication class to represent the graph of drones within each partition.
- Enable Drone Searching and Shortest Path Calculation: Implement BFS and DFS for drone search and Dijkstra's algorithm for finding the shortest communication path.
- Create a User-Friendly Interface: Build UI components for drone search, pathfinding, and displaying drone details.
- Enhance Realism in the Simulation: Include additional functionalities, such as drone self-destruction, dynamic partitioning, and obstacle avoidance.
- Analyze Performance: Measure framerate and algorithm runtime under different numbers of drones to evaluate scalability.

System Setup

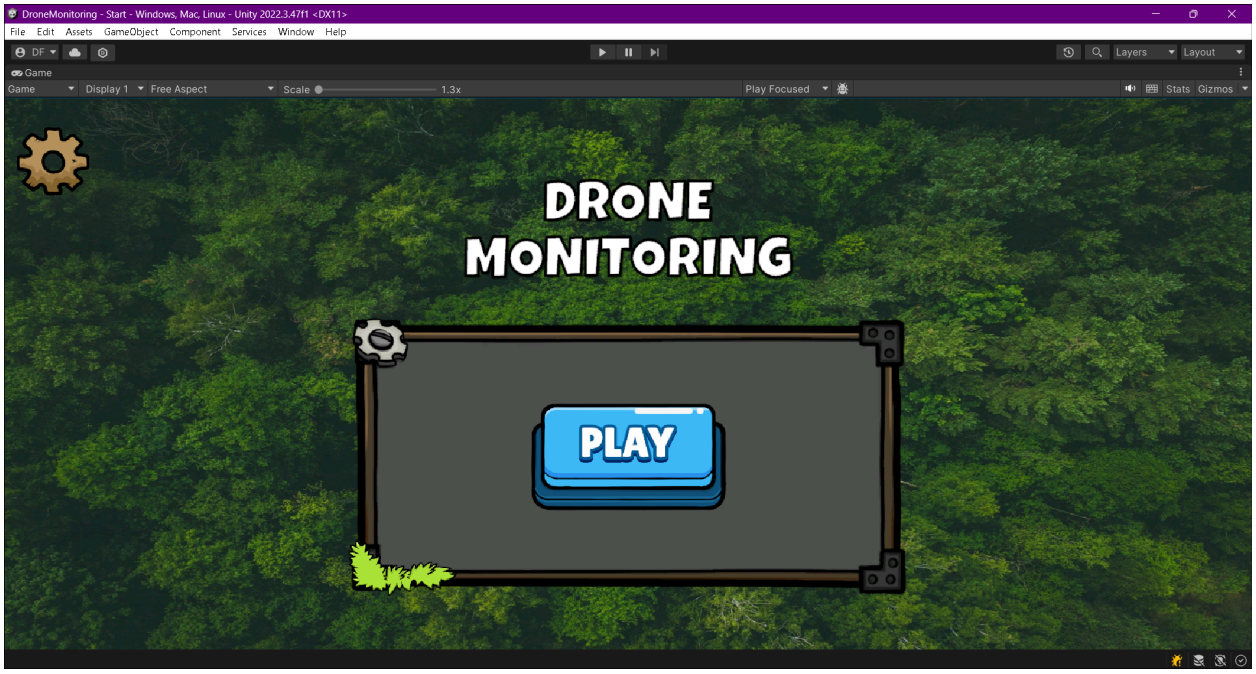
Hardware: CPU = Intel(R) Core(TM) i7-10870H CPU @ 2.20GHz

RAM = 16GB

GPU = NVIDIA GeForce RTX 3060 Laptop GPU.

Unity Version: Unity 2022.3.47f1

User Interface(UI)



The system's main UI interface consists of a main menu and scene where users can easily interact with drones, visually identifying them and performing actions through buttons like "Play".

By selecting "Play" from the Main menu, the player is sent to the game level where drone generation begins which produces a collection of drones that are subsequently allocated to the scene. Every drone has unique characteristics, like colour and colours are awarded based on this

characteristic. This colour-coding makes it easier for users to easily identify the various drone kinds in the surrounding area.

Search

When you click the **Search** button in the "Drone Monitoring" interface, the system processes the drone identifier entered in the input field (e.g., "RedDrone3") to locate the corresponding drone in the network. Once found, the application may visually highlight the drone on the map, making it easier to distinguish from others. This could involve enlarging the drone's icon, adding a visual marker, or applying a unique animation around it to draw attention. The interface is designed to provide a clear and intuitive way to identify the specific drone among a large number of others.

In addition to visual highlighting, the system updates the information panel displayed at the bottom left of the interface. Details such as the drone's position, speed, partition, and target are refreshed to reflect the searched drone's current attributes. If the drone is part of a specific partition, the application might also display related communication details, such as its role within the linked-list communication network. This ensures that users can quickly access all relevant data for the searched drone, streamlining monitoring and management.

SearchBYkey

In the "Drone Monitoring" interface, the **Search by Key** functionality likely enables users to locate drones based on a unique key or identifier associated with each drone. This key might represent specific metadata or attributes assigned to drones, such as an internal ID, code, or tag distinct from the drone name. By clicking the **Search by Key** button after entering the key in the input field, the system processes the query and locates the corresponding drone.

Once the drone is found, the interface may update by highlighting the drone on the map, similar to the regular search function. The bottom-left information panel would likely display relevant details about the drone, such as its position, speed, target, and partition. This feature is particularly useful in scenarios where users need to search for drones based on technical or system-level identifiers rather than user-friendly names, ensuring precise tracking and monitoring within the network.

FindShortestPath

The **Find Shortest Path** functionality in the "Drone Monitoring" interface appears to calculate the shortest path between two drones based on the provided inputs for the **Path Start** and **Path End** fields. In this example, "RedDrone3" is entered as the starting point, and "BlueDrone5" as the endpoint. When the user clicks the **Find Shortest Path** button, the system computes the shortest path between these two drones.

Once the calculation is completed, the result is displayed below the button, showing the shortest path's distance (e.g., "11.66 units" in this case). Additionally, the interface might visually highlight the computed path on the map, using lines or markers to indicate the connection between the drones. The information panel at the bottom left continues to show the details of the currently selected or relevant drone, along with global simulation metrics like FPS, simulation time, and drone count. This feature is valuable for managing communication routes, optimizing resource use, and analyzing drone movement within the network.

Destruct

The **Destruct** functionality in the "Drone Monitoring" interface allows the user to simulate the removal or self-destruction of a specific drone from the network. To use this feature, the user enters the name or identifier of the target drone (e.g., "RedDrone3") into the input field and clicks the **Destruct** button. This action likely triggers a process where the specified drone is removed from the map and the linked-list communication network, visually and logically.

Once the drone is destroyed, the interface updates to reflect the change. The drone disappears from the map, and its information is removed from the bottom-left display panel. Additionally, the system may recalculate the drone count and update related metrics, such as the simulation status and the network structure of any affected partitions. If the drone was part of a linked-list communication chain, the network might adjust dynamically to bypass the destroyed drone and maintain communication between the remaining drones. This feature is critical for testing fault tolerance, network integrity, and system responses to drone failures.

SearchAfterDestruct

The **Search After Destruct** functionality in the "Drone Monitoring" interface involves attempting to locate a drone after it has been removed or "self-destructed" from the network. If a user enters the identifier of a previously destroyed drone (e.g., "RedDrone3") into the input field and clicks the **Search** button, the system likely responds with an error message or visual feedback indicating that the drone cannot be found.

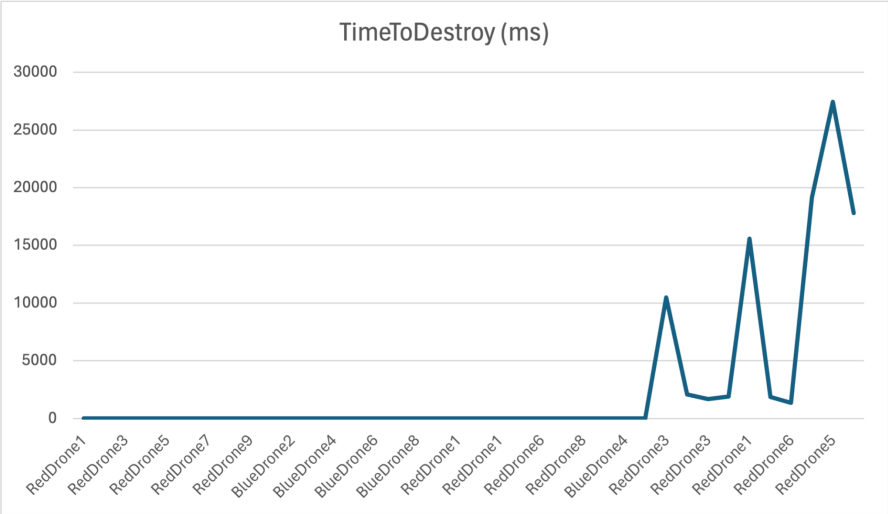
In this scenario, the map would not highlight the drone, and the bottom-left information panel would not display any details about it. Instead, the system might update with a message such as "Drone not found or inactive" to inform the user. This behavior ensures that users are aware of the drone's absence from the network, reinforcing the consequences of the destruction action and preventing unnecessary searches for removed entities. This functionality provides clarity and consistency in the system's response, ensuring that it accurately reflects the current state of the network.

FindShortestPath for DestructedDrone

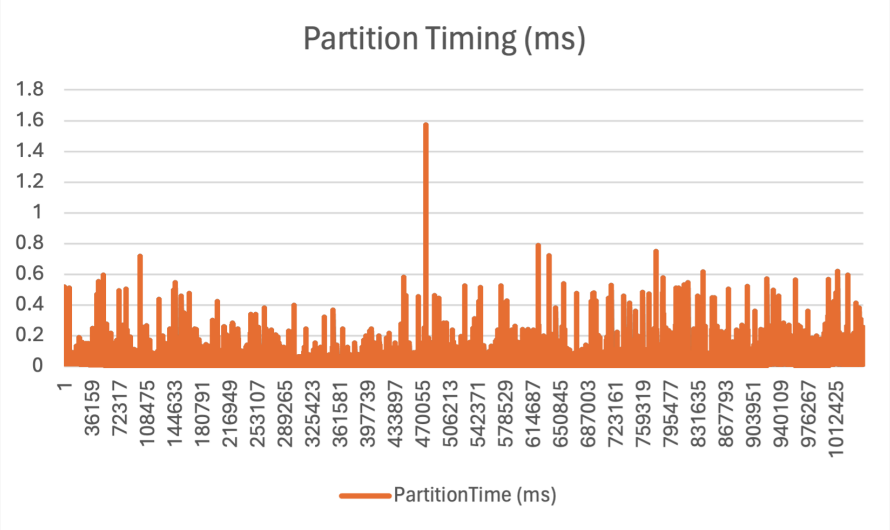
When using the **Find Shortest Path for Destructed Drone** functionality, selecting a destructed drone, such as **RedDrone3**, as the starting point would result in an invalid path calculation. Since **RedDrone3** no longer exists in the network, the system cannot compute a valid route. If the user enters **RedDrone3** as the starting drone and **BlueDrone5** as the target, the interface would likely display an error message, such as "Drone is not found or inactive" . This ensures the user is immediately informed that the chosen drone is no longer available for pathfinding.

In response to the error, the system would also clear any previous path highlights from the map and prevent any visual representation of a route. No path would be drawn, and the interface might prompt the user to select an active drone as the new starting point. This ensures that the pathfinding feature only operates with valid, active drones, maintaining clarity and consistency in the system's operation. The error handling guides users to correct their input, ensuring that the pathfinding process can proceed smoothly once a valid drone is selected.

Analysis



The "Time to Destroy" graph shows how long it takes for drones to self-destruct during the simulation. At first, most RedDrones and BlueDrones destroy themselves quickly, with times near zero, showing the system works efficiently. But for certain drones like RedDrone6 and RedDrone5, the destruction time jumps to over 25,000 ms. This might mean there are problems with the graph algorithms or the way drones are connected. The big spikes suggest that fixing the partitions or improving the algorithms could make things faster and more reliable.



The "Partition Timing" graph demonstrates that the algorithm used for dividing drones into two partitions is generally efficient, with most timings between 0.3 ms and 0.6 ms, ensuring real-time performance for communication within the swarm. However, the noticeable spike around 1.6 ms suggests potential inefficiencies or scalability limits when handling specific graph structures or larger drone configurations. This outlier could lead to delays in critical operations like message relaying or coordination, impacting the responsiveness of the simulation in dynamic scenarios. Overall, while the partitioning process is stable and suitable for most cases, optimizing the algorithm to handle edge cases and scaling efficiently with larger swarms would enhance performance and ensure consistent low-latency communication in the drone network.

Appendix

Link to demonstration video: https://youtu.be/f7WMXfMX4F8?si=umNr8t9OpmAp6dwu