Carleton University

Department of Systems and Computer Engineering

SYSC 3303A - RealTime Concurrent Systems - Winter 2025

Project Specification: Firefighting Drone Swarm

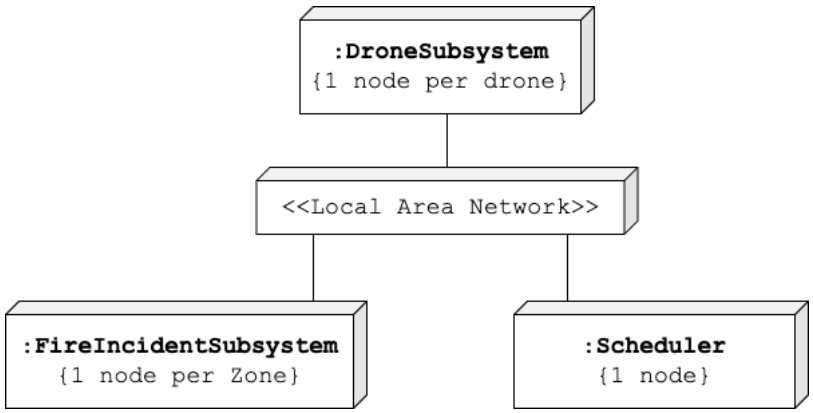
Teams will design and implement a control system and simulator for a firefighting drone swarm. The system will include:

1. A **Scheduler** (controlling and coordinating the drones)
2. A **Drone Subsystem** (representing the drones themselves, including motors/propellers, water/foam tanks, and sensors)
3. A **Fire Incident Subsystem** (simulating one or more "zones" where fires can break out, along with sensors and user requests for assistance)

Each of these three components should run as a separate process (i.e., separate Java programs) and communicate using **DatagramSocket** objects (UDP). Eventually, they will be deployed on separate computers in the lab, or on a single machine with multiple Java instances.

Your simulation must be **configurable** with respect to the number of drones, the number of fire zones, the time it takes for drones to drop firefighting agents, how long it takes to travel between zones, and so on. Your system is expected to operate in "**real-time**," meaning that it should simulate the passage of time for drone movements, water/foam drops, nozzle operations, and so on.

The code will be written in **Java**, using the **IntelliJ IDE**, and must be designed to run within the provided lab environment.



System Components

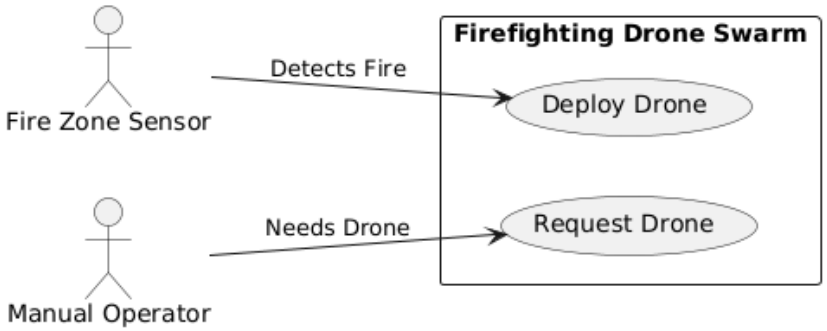
1. **Scheduler**
   - **Core Controller:** Receives fire incident requests from the Fire Incident Subsystem, determines which drone(s) to dispatch.
   - **Fault & Error Handling:** Must detect problems such as a drone getting "stuck" in flight or a nozzle failing to open.
   - **Dispatch & Coordination:** Sends commands to drones (take off, drop agent, return to base, etc.) and monitors their status.
   - **Runs as its own process.**
2. **Drone Subsystem**
   - **Water/Foam Tanks:** The drone carries a limited firefighting agent supply, which can be "dropped" on a fire.
   - **Motors/Propellers:** Control flight between incident zones
   - **Bay Doors / Nozzle:** Opens or activates to drop water/foam
   - **Sensors & Indicators:**
     - **Arrival Sensor:** Detects when the drone has arrived at or is above a target fire zone.
     - **Agent Level Sensor:** Indicates remaining water/foam capacity.
     - **Status Lights:** e.g., "flying," "dropping agent," "refilling," etc.

   Each drone subsystem instance communicates with the Scheduler to report status and receive commands.
3. **Fire Incident Subsystem**

   Simulates zones where fires occur.
   - **Fire Request Mechanism:** When a fire starts (or flares up), it sends a request for drone assistance.
   - **Incident Lamps/Indicators:** Indicate that a drone is inbound or currently fighting the fire.
   - **Simulated "Users":** Could be sensors or operators who report the fire. They also confirm when the fire is extinguished.

   Generates arrival notifications, logs, or other relevant data to verify correct drone service.

Use Cases



Deploy Drone Use Case

Actors: Fire Zone Sensor (primary), Arrival Sensor

Precondition: A fire is detected at a given zone.

Description:

1. The fire zone sensor reports a fire to the Scheduler, providing its location and severity.
2. The fire incident sensor detects a fire and sends a "fire request" to the Scheduler.
3. The Scheduler chooses an available drone (or drones) and instructs it/them to proceed to the affected zone.
4. If the drone is on standby (idle), the system transitions the drone to "flight" mode (instructs it to depart immediately). Otherwise, the request is queued and awaits scheduling.
5. The Scheduler adds this request to its queue of missions.
6. The drone receives flight instructions, takes off, and travels toward the fire zone.
7. As the drone approaches (position is tracked in real-time), the arrival sensor signals the Scheduler.
8. The Scheduler commands the drone to initiate firefighting (drop water/foam).
9. The drone reduces its water/foam capacity during the drop. If more water/foam is needed, it may need to leave and refill.
10. Once the fire is controlled (is extinguished), the zone sensor notifies the Scheduler that the fire is out.
11. The drone either returns to base or proceeds to the next assignment.

Alternatives:

- If the drone cannot deploy its water/foam nozzle (fault), the Scheduler must **handle this error** and possibly dispatch another drone. The drone must be **returned back to base**.
- If the drone is out of water/foam, it must first **refill** (return back to base).

Postcondition: The zone is either fully serviced or awaiting further assistance from another drone.

Request Drone Use Case

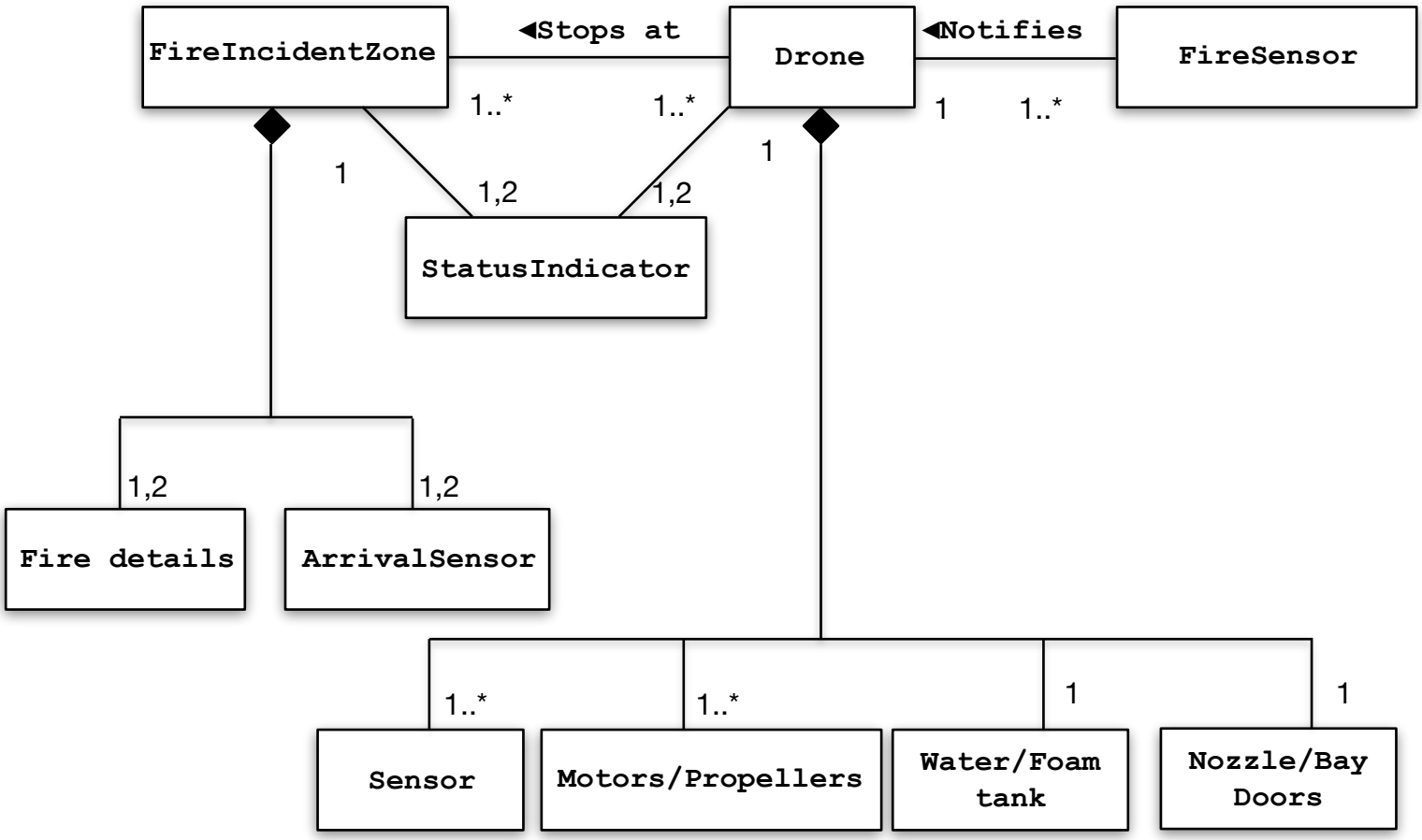
Actors: Manual Operator (primary), Arrival Sensor

Precondition: A user at a fire zone calls for drone assistance (manually or due to heightened risk).

Description:

1. A user or system operator at the zone triggers a "request drone" event.
2. The Fire Incident Subsystem sends the request to the Scheduler with details about the zone's needs.
3. The Scheduler receives this request and identifies which drone is best suited (e.g., based on location, available water/foam, battery).
4. Upon receiving instructions, the drone navigates to the fire zone.
5. When the drone arrives (detected by an arrival sensor), the Scheduler is notified.
6. The drone drops water/foam. ~~or conducts a reconnaissance pass if instructed~~.
7. If multiple zones need attention, the drone can be routed accordingly.

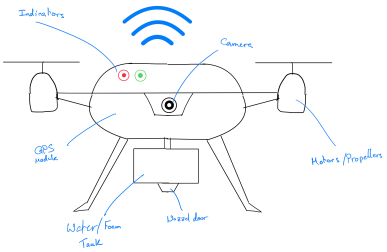
Postcondition: The drone has been successfully deployed to the requesting zone for immediate or pending firefighting action.



Static Model of the Problem Domain

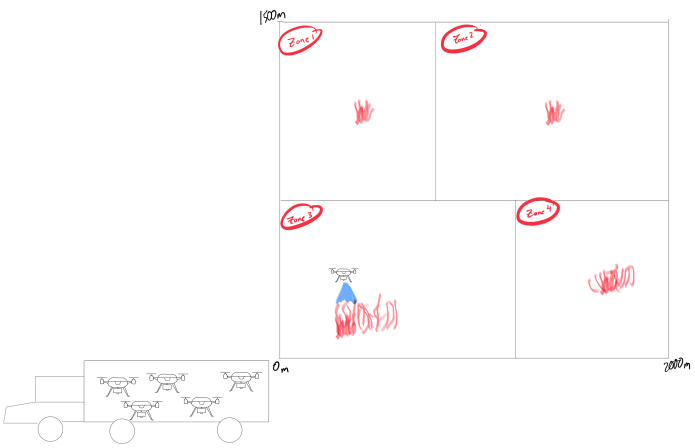
Each drone has:

- **Water/Foam tank:** limited capacity of firefighting agent.
- **Motors/Propellers:** allow movement between zones.
- **Nozzle/Bay Doors:** open/close mechanism for firefighting agent release.
- **Status indicators:** flight, approach, dropping agent, etc.
- **Sensors:** track arrival, measure water/foam levels, detect faults.



Each fire incident zone has:

- **Fire sensors** or **manual triggers** (Request buttons): to request a drone.
- **Indicators:** show that a drone is inbound, has arrived or actively dropping agent.
- **Arrival sensors:** detect drone presence.
- **Fire details:** severity, spread rate, etc.



Real-Time Behavior & Configurability

- You will simulate the time it takes for drones to fly between zones, drop water/foam, and possibly land or hover for extended operations.
- The system must accept configuration parameters such as:
  - **Number of drones**
  - **Number of zones (fire sites)**
  - **Travel time** between zones
  - **Agent drop time** (time required to release water/foam)
  - **Drone capacity** (amount of agent each drone can carry)

Fire Incident Subsystem Simulation

The fire incident subsystem simulates fire outbreaks and potential user/automated requests. It should also test for the proper operation of the drone, i.e., a fire is detected and a request was sent to the drone, waits for the arrival of a drone to the zone, then extinguish the fire (release firefighting agent).

Your fire incident subsystem should read an input file that drives events, e.g., time stamps, zone IDs, severity levels, or repeated calls for assistance. The file should use the format shown below:

Time	Zone ID	Event type	Severity
hh:mm:ss.mmm	n	FIRE_DETECTED/DRONE_REQUEST	High/Moderate/Low
14:03:15	3	FIRE_DETECTED	High
14:10:00	7	DRONE_REQUEST	Moderate

i.e., a time stamp, white space, an integer representing the zone that the fire is detected or drone was requested in, white space, a string consisting of either FIRE_DETECTED or DRONE_REQUEST, a white space, a string consisting of either High, Moderate or Low. Use the following amounts (in litres) of water/foam needed for each event severity:

Severity	Amount of water/foam (L)
Low	10
Moderate	20
High	30

For example, in the table above, a fire is detected at 2:03:15 pm in zone 3 with high severity. It is strongly advisable to maintain separate queues for each zone and severity since fires might be at different locations before a certain drone reaches its destination. The subsystem must handle multiple simultaneous fire requests (in case fires erupt in different zones). It must confirm correct drone action: a fire zone receives assistance after a drone arrives and uses its agent.

Also assume that all zones are rectangular shapes and their coordinates (in meters) must be read from a different file that follows the format shown below:

Zone ID	Zone Start	Zone End
n	(x1, y1)	(x2, y2)
1	(0, 0)	(700,600)
2	(0, 600)	(650,1500)

Scheduler Specification

- Accepts fire reports (or requests) from the Fire Incident Subsystem and dispatches drones to respond.
- Manages mission queues to minimize response times or prevent any zone from being ignored.
- Tracks and updates each drone's status: location, remaining firefighting agent, faults, etc.
- Must handle communication faults (lost packets) or drone hardware faults (nozzle failure, drone stuck in flight).

Drone Subsystem Specification

- Implements the drone state machine: idle, en route, dropping agent, refilling, faulted, etc.
- Sends notifications to the Scheduler upon arrival at a zone.
- Opens/closes nozzle or bay doors when instructed by the Scheduler (or upon local state changes).
- Tracks agent capacity and simulates depletion during each firefighting operation. If empty, the drone must refill.

User Interface

- **Minimal** UI is required for early iterations; eventually, show a **map/console** of zones and drones.
- Must allow loading of the **incident input file** and system configuration:
  - Number of drones, number of zones, agent capacity, travel times, etc.
- The final iteration should display each drone's position/status in real time.

Development Process

An **iterative, incremental process** will be used to develop the system. Each iteration produces an executable subset of the final solution. During lab sessions, you will present the working features of your **previous** iteration's code.

Iteration 0 – Data Collection

Collect or estimate real-world drone parameters:

- Travel time between zones
- Time to open/close nozzle doors
- Rate of dropping water/foam
- Acceleration/deceleration for takeoff/landing

Record this data (e.g., in Excel), and justify how you arrived at your chosen values. Add references to these values. Submit a PDF with analysis.

Iteration 1 – Establish Basic Communication

Using Assignment 1, create three threads, one for each subsystem identified above. The Fire Incident Subsystem and the Drones are the clients in the system; the Scheduler is the server. The Fire Incident subsystem is to read in events using the format shown above: Time, Zone ID, Event Type and Severity. Each line of input is to be sent to the Scheduler. The Drones will make calls to the Scheduler which will then reply when there is work to

be done. The Drone will then send the data back to the Scheduler who will then send it back to the Fire Incident subsystem. For this iteration the Scheduler is only being used as a communication channel from the Fire Incident Subsystem thread to the Drone thread and back again. It is only necessary to create a test case showing that your program can read the input file and pass the data back and forth. Note that in a future iteration, there will be multiple Drones all running independently of each other, and there will be three programs running on separate computers, so be sure that you allow for this configuration when you design your code, or you will have to refactor your design at a later point in time.

You probably don't want to pass text strings from the Fire Incident subsystem to the rest of the system. You should develop a data structure which only passes the necessary information representing the hardware device (Fire Zone ID, Severity, etc.) to the scheduler. If you choose to work ahead, it would be a good idea to tag this "version" in GIT for the purposes of the demo to the TA's and for submission in Brightspace.

Iteration #1 Summary:

- Create three threads (or processes):
  1. **Fire Incident Subsystem** (client)
  2. **Drone Subsystem** (client)
  3. **Scheduler** (server)
- The Fire Incident Subsystem reads input events (time, zone, event type, severity, etc.) and sends them to the Scheduler.
- The Drone Subsystem consults the Scheduler to see if it has tasks to perform (put out fires).
- For this iteration, the Scheduler primarily acts as a pass-through: reading messages from the Fire Incident Subsystem and forwarding them to the Drone Subsystem (and vice versa).
- Focus on communication: verify you can read an input file, pass messages, and get round-trip acknowledgments. Real scheduling logic will be added later.

Work Products for Iteration #1:

- "README.txt" file explaining the names of your files, set up instructions, etc.
- Breakdown of responsibilities of each team member for this iteration
- UML class diagram and sequence diagrams.
- Detailed set up and test instructions, including test files used
- Code (.java files, all required IntelliJ files, etc.)

Iteration 2 – Add Scheduling & Drone State Machines

The goal of this iteration is to add the state machines for the scheduler and drone subsystems assuming that there is only one drone. However, you should bear in mind that for the next iteration, your system is expected to coordinate between the drones in order to maximize the number of zones serviced over time (i.e., the throughput). Note that the drone subsystem is used to notify the scheduler that an drone has reached a zone, so that once an drone has been told to move, the drone subsystem also has to be informed so that it can send out messages back to the scheduler to denote the arrival by an drone. You can either maintain a single event list or have separate tasks for each drone. Perhaps you can think of another way of doing it too.

Iteration #2 Summary:

- Implement core scheduling logic to decide which drone handles a new fire.
- Implement drone state transitions (idle, en route, dropping agent, etc.).

- Start with only one drone to simplify debugging.

Work Products for Iteration #2:

- "README.txt" file explaining the names of your files, set up instructions, etc.
- Breakdown of responsibilities of each team member for this iteration
- UML class diagram and sequence diagrams.
- State machine diagram for the scheduler and drone subsystems.
- Detailed set up and test instructions, including test files used
- Code (.java files, all required IntelliJ files, etc.)

Iteration 3 – Multiple Drones & Distributed Execution

For this iteration, you are to use what you have learned from Assignments 2 and 3 and split your system up into three (or more) separate programs that can run on three separate computers and communicate with each other using UDP. Your task in Assignment 3 was to implement a simple version of a remote procedure call, so you are encouraged to recycle that code here. The Scheduler will now be used to coordinate the movement of drone such that each drone services roughly the same number of zones as all of the others and so that the waiting time for fires to be extinguished in a zone is minimized. Hint: if zone 1 needs to be serviced by a drone and there is a drone on the way to service zone 2 which has the same severity and that drone must pass through zone 1, then zone 1 must be serviced instead. The state machines for each drone should execute independently of each other, but they will all have to share their location with the scheduler. The scheduler will choose which drone will be used to service a given request.

Iteration #3 Summary:

- Increase complexity: run separate programs on potentially separate machines.
- The Scheduler coordinates multiple drones, ensuring balanced workloads.
- Each drone operates independently but reports status to the Scheduler.

Work Products for Iteration #3:

- "README.txt" file explaining the names of your files, set up instructions, etc.
- Breakdown of responsibilities of each team member for this and previous iterations
- Any unchanged diagrams from the previous iterations UML class diagram
- Detailed set up and test instructions, including test files used
- Code (.java files, all required IntelliJ files, etc.)

Iteration 4 – Fault Handling

For this iteration, you will be adding code for detecting and handling faults. To this end, you will have to add timing events so that if the timer goes off before a drone reaches a zone, then your system should assume a fault (either, the drone is stuck between zones, nozzle/bay doors are stuck or the arrival sensor at a zone has failed). Similarly, you should detect whether a nozzle/bay doors or not, or is stuck open. In Iteration 5 below, your drone status output should show these faults. A door which has not closed should be regarded as a transient fault, so your system should be able to handle this situation gracefully (unlike the LRT). However, the a nozzle/bay doors fault should be regarded as a hard fault and should shut down the corresponding drone.

You must submit code to enable us to see that your drone scheduler can deal properly with the faults shown above (i.e., you must be able to inject these faults into the system). I suggest that you inject these faults using the input file (so you will have to modify its format and be able to show to us how it works).

Iteration #4 Summary:

- Inject faults:
  - Drone stuck mid-flight
  - Nozzle jammed
  - Packet loss or corrupted messages
- The Scheduler should detect and handle these faults gracefully (e.g., reroute missions, mark a drone as offline).

Work Products for Iteration #4:

- "README.txt" file explaining the names of your files, set up instructions, etc.
- Breakdown of responsibilities of each team member for this and previous iterations
- Any unchanged diagrams from the previous iterations UML class diagram
- Timing diagrams showing the error scenarios for this iteration
- Detailed set up and test instructions, including test files used
- Code (.java files, all required IntelliJ files, etc.)

Iteration 5 – Capacity Limits & UI

The last stage of the project is to add a display console showing where each of the drone is in real time and displaying any faults (if any). The idea is to have output suitable for the fire fighter dispatcher sitting at the desk in headquarters to refer to. This part of the project can be part of the scheduler subsystem if you so choose, or you may want to develop a separate subsystem altogether.

You are also to implement capacity limits (if not done so previously) so that zones waiting to be serviced will have to wait for a drone with a full tank.

You are also to to add instrumentation to your code to record the time it takes to extinguish all fires in the input file and to record the distance required to reach desired zone. In the real world, the goal is to minimize these numbers.

Iteration #5 Summary:

- Enforce agent capacity: if a drone runs out of firefighting agent, it must return to a "refill station."
- Add a user interface showing each drone's location, fire statuses, and any faults.
- Collect and log overall performance metrics (response times, fire extinguish times, etc.).

Work Products for Each Iteration

1. README.txt with instructions and file naming conventions
2. Team Responsibilities breakdown
3. UML Diagrams (class, state machine, sequence diagrams)
4. Detailed Setup & Test Instructions (including input files)
5. Source Code (.java files, IntelliJ project files)

Final Project Presentation

Your team will give a 15 minute presentation to a TA of iteration #5. It must show 10 drones running with 4 fire incident zones. Your demonstration should also show a drone stuck mid-flight and nozzle jammed on a drone. The goal of the presentation is to show us how awesome your team and project are, and to submit your final copy of your work using Brightspace.

Finally, we will have a little contest to see whose Firefighting Drone Swarm system can extinguish all fires from an input file supplied to you in the least amount of time. We shall assume that all the drones are working for the entire period (no faults).

Final Project Presentation Summary:

- A 15-minute demo of your final system.
- Must show multiple drones (10) responding to multiple zones (5 zones).
- Demonstrate at least one fault scenario (e.g., a drone stuck mid-flight) and how your Scheduler manages it.
- The goal is to showcase your system's abilities and to submit all code/documentation on Brightspace.

The winning team earns bragging rights!

Final Deliverables

You must submit a **report** containing:

1. A report consisting of:
   - Team number and team members
   - Table of contents
   - Breakdown of responsibilities of each team member for each iteration
   - All diagrams:
     - UML class diagrams for the three components.
     - A State Machine diagram for the scheduler.
     - Sequence diagrams showing all the error scenarios.
     - Timing diagrams for the scheduler.
   - Detailed set up and test instructions.
   - Results from your measurements.
   - Reflection on your design - what parts do you like and what parts should be redone.
2. Test files for all iterations.
3. Code (.java files, all required IntelliJ files, etc.)

Submission and Grading

Please refer to Brightspace for all submission dates.

Work products for each iteration are to be submitted using Brightspace. Submit using your project team's number, (one submission per team) by the date/time specified. You can include any number of files in your submission. Ensure that all your files are accepted. Please submit PDF instead of Word, LibreOffice, Pages, etc. documents.

The project is worth 30% of your final grade. Iterations #1 through #4 are worth 2.5 marks each. The final demo is worth 5 marks. 15 project marks are for the final deliverables. These 30 marks are for the entire team.

However, based on the participation of each member, each individual's mark could be higher, the same, or lower than the team mark. The highest marks will be awarded to teams where each member participates and contributes roughly equally. Note that a team member who dominates the group is just as likely to lose marks as an individual who goofs off. Mentoring and team-programming is strongly encouraged. Please make sure that all criticism is constructive. I too have written some pretty crappy code in my day and have appreciated constructive feedback.

Submission and Grading summary

- Submit via Brightspace by the deadlines given.
- One submission per team; ensure all files are accepted.
- Please submit PDFs for documentation.
- The project is **30%** of your final grade, split as follows:
  - Iterations #1–#4: 2.5 marks each (10 total)
  - Final demo: 5 marks
  - Final deliverables: 15 marks
- Individual marks can vary based on participation. Balanced contributions typically earn the highest marks.

Bi-Weekly Meetings

Each team will have an approximately biweekly 15 minute meeting with a TA to discuss their progress. The meetings are mandatory, and will be held during the labs. The TA's will expect a demo of each iteration during the meeting. This will be complicated with remote learning, but we will muddle along as best we can.

Good luck and have fun implementing your Firefighting Drone Swarm!

Graduate Attrirbute		Performance Level	Level 1	Level 2	Level 3	Level 4
		Level Descriptor	**Beginning**	**Developing**	**Accomplished**	**Exemplary**
6	**Individual and team work**	6.1 Personal and group time management	*Deadlines often missed. Several projects or assignments appear hurried or inadequately addressed. No evidence of time management planning.*	*Some deadlines missed. Some tasks or assignments appear hurried or inadequately addressed. Some evidence of time management planning.*	*No important deadlines missed. Few tasks or assignments appear hurried or inadequately addressed. Good time management planning.*	*No deadlines missed. Tasks and assignments always excellently prepared and presented. Tasks occasionally completed ahead of schedule. Leadership in time planning.*
		6.2 Group culture, group dynamics	*Not dependable. Little or no trust from teammates. Dishonest or evasive interactions with teammates. Poor conflict resolution.*	*Usually dependable and trusted by teammates. Mostly ethical behavior towards project and teammates. Minor conflicts.*	*Dependable and trusted by teammates. Ethical open interactions with teammates. Good conflict resolution.*	*Dependable and trusted by team mates. Ethical and open interactions with teammates. Leads team in ethical and responsive behavior. Leadership in conflict resolution.*
		6.3 Leadership: initiative and mentoring, areas of expertise, and interdisciplinary teams	*No initiative. No effort to mentor group members or take ownership of an area of expertise.*	*Some initiative. Some effort to mentor group members and take ownership of an area of expertise.*	*Good initiative and self-motivation. Regular effort to mentor group members. Clear command of an area of expertise.*	*Self-motivated, with regular demonstration of initiative. Constructively directs other team members and helps them to improve at their tasks. High degree of knowledge in an area of expertise.*

Change Log

1.0: Initial version.

1.1: Revise input table format

1.2: Revise zone coordinates as input from file

1.3: Added, amounts of water/foam needed for each event severity level. Also, added diagrams showing drone components and sample grid layout.