Firefighting Drone System
Final Report

SYSC3303 A2 Team 3

Members:

Hanyoung Chung
Nick Fuda
Thomas Imbert
Elton Kong
Bryson Kushner

Table of Contents

1.0 Breakdown of Responsibilities

At the beginning of the project, it was determined that the team would delegate tasks for each iteration. This would allow for the team to adapt to the strengths and weaknesses of members, while also allowing each member to have experience working with each other on different components and apply the course material.

1.1 Iteration 1

For iteration 1, the team met in the first lab and discussed the requirements for the iteration and how the tasks would be split. The attending team members were Thomas Imbert, Hanyoung Chung, Elton Kong, and Nick Fuda. The responsibilities for the first iteration were as follows:

Table 1: Iteration 1 Responsibilities

Team Member	Contributions
Thomas Imbert	- Sequence Diagram - Code review
Hanyoung Chung	- Unit tests and test classes - Code debugging and review
Elton Kong	- Class UML Diagram - Code review
Nick Fuda	- DroneSubsystem class - FireIncidentSubsystem class - Scheduler class - FireEvent class

In this iteration, it was deemed easiest for one member to write the majority of the code, as they were required to work together and worked using similar procedures to assignment 1. This also allowed for other members who did the diagrams to review the code and create the diagrams for the system. One member dedicated to unit tests and debugging allowed for issues and oversights to be found. This approach allowed each of the members to understand the system and work together collaboratively to create the foundational first iteration of the project. The classes were implemented to specification, and a class called FireEvent was designed to hold event objects and their information.

1.2 Iteration 2

For iteration 2, the team met in the lab to demo iteration 1 and discussed the requirements for the iteration and how the tasks would be split. The attending team members were Thomas Imbert, Hanyoung Chung, Elton Kong, Bryson Kushner and Nick Fuda. The responsibilities for the second iteration were as follows:

Table 2: Iteration 2 Responsibilities

Team Member	Contributions
Thomas Imbert	- Scheduling algorithm - Simulation class - Code review
Hanyoung Chung	- Unit tests and test classes - Helping integrate simulation class - Code debugging and review
Elton Kong	- Sequence diagram - Code review
Nick Fuda	- DroneSubsystem state machine - Drone helper class - Code debugging and review
Bryson Kushner	- UML class and state diagrams

In iteration 2, a scheduling algorithm was created to sort incoming events by severity using a queueing algorithm. Medium and Low severity events were added to the back of the queue on a first come first serve basis, while high priority events pushed to the front of the queue on arrival. A simulation class was also made to simulate real time, which allowed for better testing and visualization of the system. In this iteration, a state machine was also implemented using a helper class to represent drones in the drone subsystem. A typical state machine for this iteration is a closed loop; idle → en route → deploying agent → returning → idle. The team worked on individual portions of the assignment while in close contact with each other to ensure cohesiveness between classes.

1.3 Iteration 3

For iteration 3, the team met in the lab to demo iteration 2 and discussed the requirements for the iteration and how the tasks would be split. The attending team members were Thomas Imbert, Hanyoung Chung, Elton Kong, Bryson Kushner and Nick Fuda. The responsibilities for the third iteration were as follows:

Table 3: Iteration 3 Responsibilities

Team Member	Contributions
Thomas Imbert	- Simulation code and integration - Sequence Diagrams - State Machine Diagrams
Hanyoung Chung	- Unit tests and test classes - Code debugging and review
Elton Kong	- Sequence diagram - Documentation - Code review
Nick Fuda	- UDP implementation - Multithreading for Drones - Multithreading for scheduler messaging - Code debugging and review
Bryson Kushner	- UML class diagram

Iteration 3 required implementation of UDP to allow the programs to run on different computers and work together. This was implemented using multithreading for the scheduler messaging, so that the scheduler could send and receive messages on separate ports, preventing deadlocking. Multiple drones were also added to each operate on their own thread within the drone subsystem. Rerouting implementation was started to interrupt travel and minimize response times. The simulation code was updated to function with UDP and unit tests were added to ensure ports functioned correctly. These changes required updates for the class and sequence diagrams.

1.4 Iteration 4

For iteration 4, the team met in the lab to demo iteration 3 and discussed the requirements for the iteration and how the tasks would be split. The attending team members were Thomas Imbert, Hanyoung Chung, Elton Kong, Bryson Kushner and Nick Fuda. The responsibilities for the fourth iteration were as follows:

Table 4: Iteration 4 Responsibilities

Team Member	Contributions
Thomas Imbert	- Simulation code and integration - Fault handling - State Machine Diagrams
Hanyoung Chung	- Unit tests and test classes - Code debugging and review
Elton Kong	- Sequence diagram - Drone location/Tracking implementation - Code review and documentation
Nick Fuda	- Documentation - Help with designing drone tracking - Fault handling tests - Code debugging and review
Bryson Kushner	- Fault Handling - UML class diagram

Iteration 4 required fault simulation and handling. Faults caused by packet loss or corrupt messages were to be handled by the scheduler, while drone faults were to be handled by the subsystem. Drones that encountered faults, such as nozzle jamming and getting stuck, had alternative routes in their state machines for handling them, including returning to base, disabling, and relinquishing control of an event back to the subsystem to be handled by a different drone. Drone location tracking was implemented in this iteration to help with the GUI in iteration 5, and tests were created to ensure that each fault was handled in a predictable, controlled manner. Timing diagrams were not included in this iteration as an oversight by the team.

1.5 Iteration 5

For iteration 5, the team met in the lab to demo iteration 4 and discussed the requirements for the iteration and how the tasks would be split. The attending team members were Thomas Imbert, Hanyoung Chung, Elton Kong, Bryson Kushner and Nick Fuda. The responsibilities for the fifth iteration were as follows:

Table 5: Iteration 5 Responsibilities

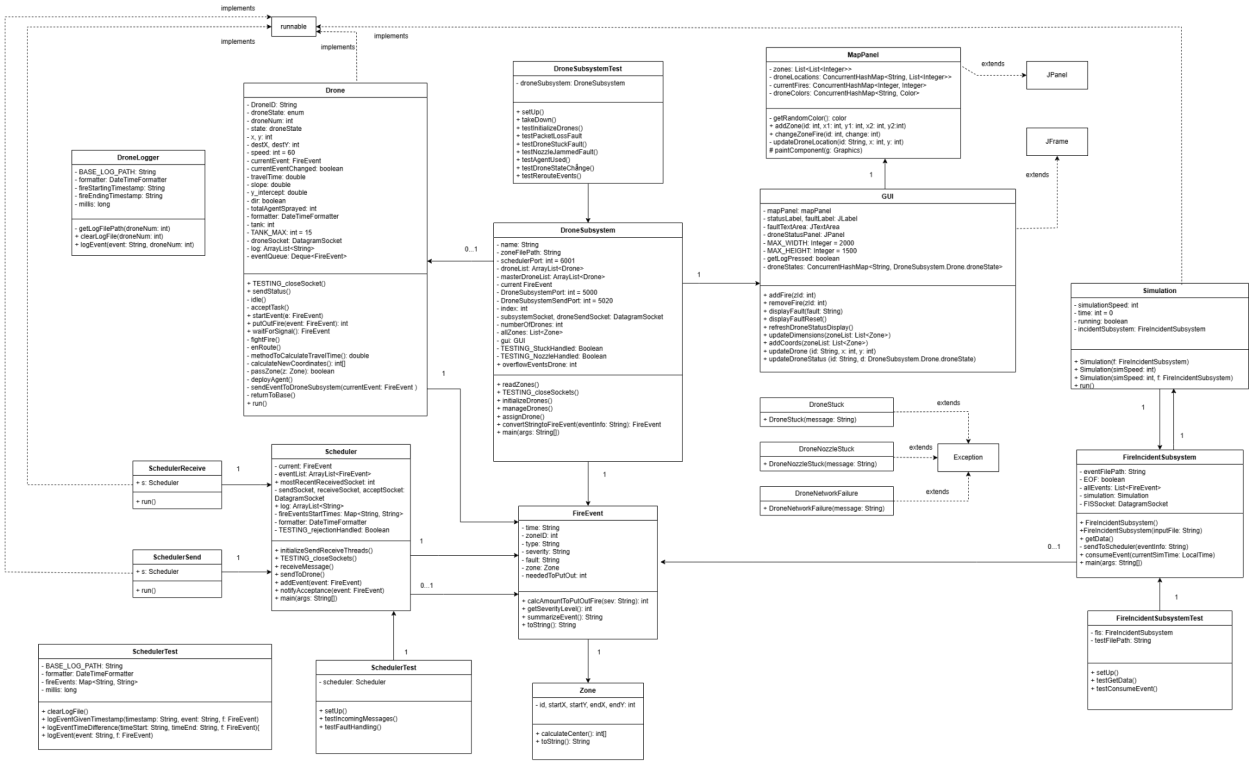
Team Member	Contributions
Thomas Imbert	- GUI and Tests - Logging and Metrics - Agent Tanks Capacity Limits
Hanyoung Chung	- Unit tests and test classes - Code debugging and review
Elton Kong	- Sequence diagram - Interrupt method and GUI - Code review and documentation
Nick Fuda	- Final Report - Timing Diagrams - Code debugging and review
Bryson Kushner	- Interrupt Event Handling - Debug and code review

For the final iteration before the 15 minute demo, GUI was implemented with metrics and fault logging. Ten drones were required for this iteration, which required multithreading to simulate concurrency. The implementation of extinguishing agent tanks and interruption for handling local events was fully realized. The team worked closely to ensure the GUI would properly represent the running simulation, and that the final specifications were met to complete the project. Final test cases were added, and timing diagrams were included. To avoid the issues of “too many cooks in the kitchen”, the report was delegated to one person and the rest of the team worked in pairs to complete the code.

2.0 Diagrams and Analysis

The documentation throughout the production of the system has given useful design information. The designs have allowed for each iteration to have pre-designated changes without needing to rebuild the entire project. These also allowed for proper tracking of expected behaviour and simulation.

2.1 UML Class Diagram

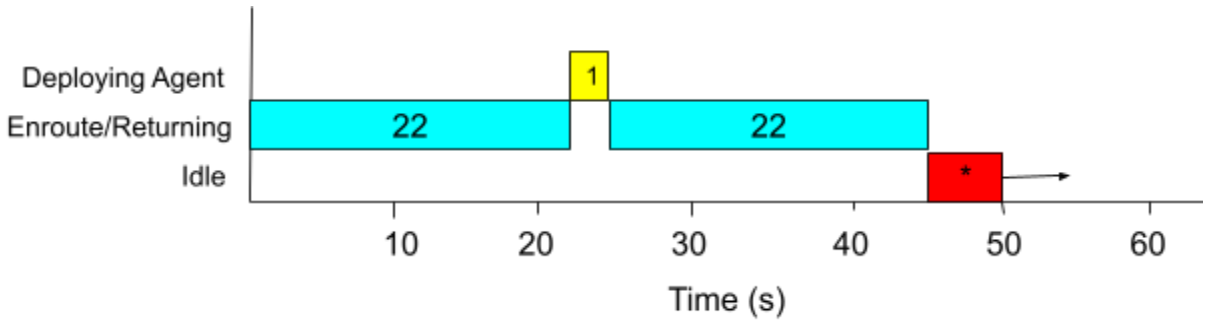


UML Class Diagram: Class diagram representing entire system

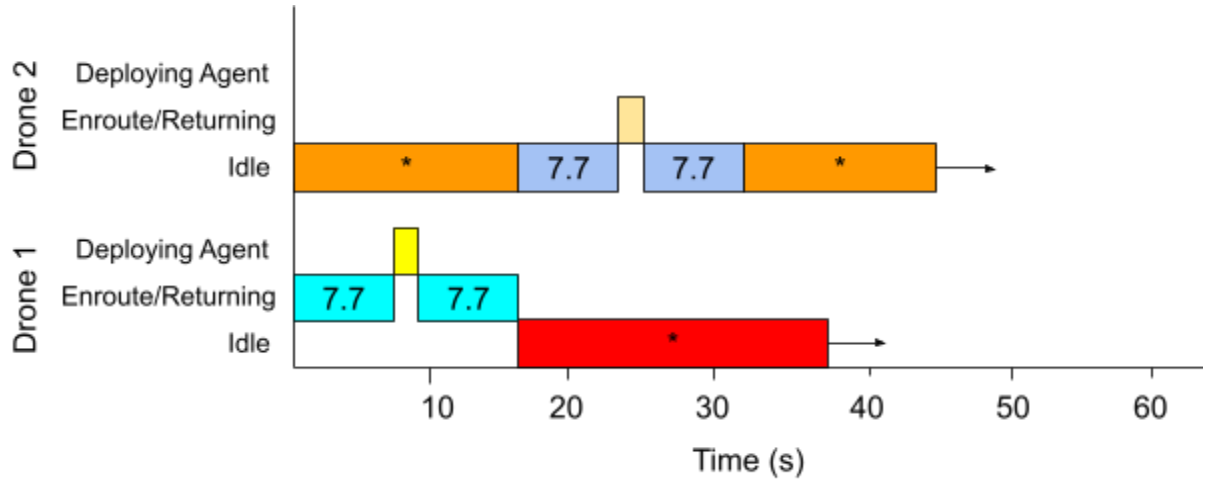
This UML Class diagram covers the entirety of the project. The main classes to model the system are the Scheduler, DroneSubsystem, FireIncidentSubsystem. These classes communicate and perform the functions of the system, including the intake of events from a file, the scheduling of events, the dispatching of drones, logging of events, and the handling of faults. Each of these classes have a test class for unit tests. FireEvent is a class representing a data object for events, and the GUI class handles the view. The simulation class runs the system at a changeable time dilation. Due to the size, this UML is difficult to read in a document, and it is recommended to view the original file either on the Github repository or in the final project submission on brightspace.

2.2 Timing Diagrams

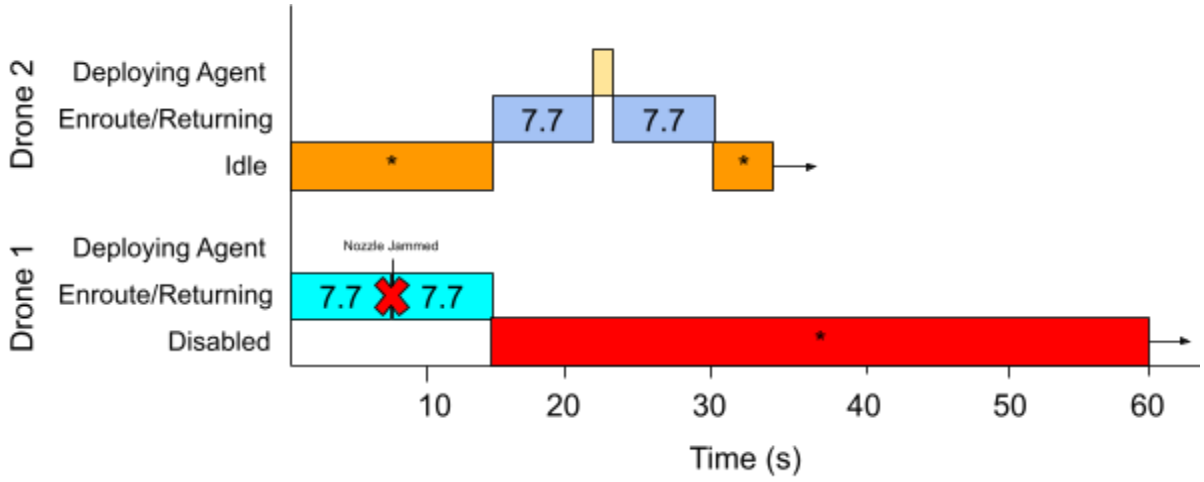
Note that the timing diagrams reflect simulation times at 3.52 times speed using the Sample_zone_file.csv found in the project folder



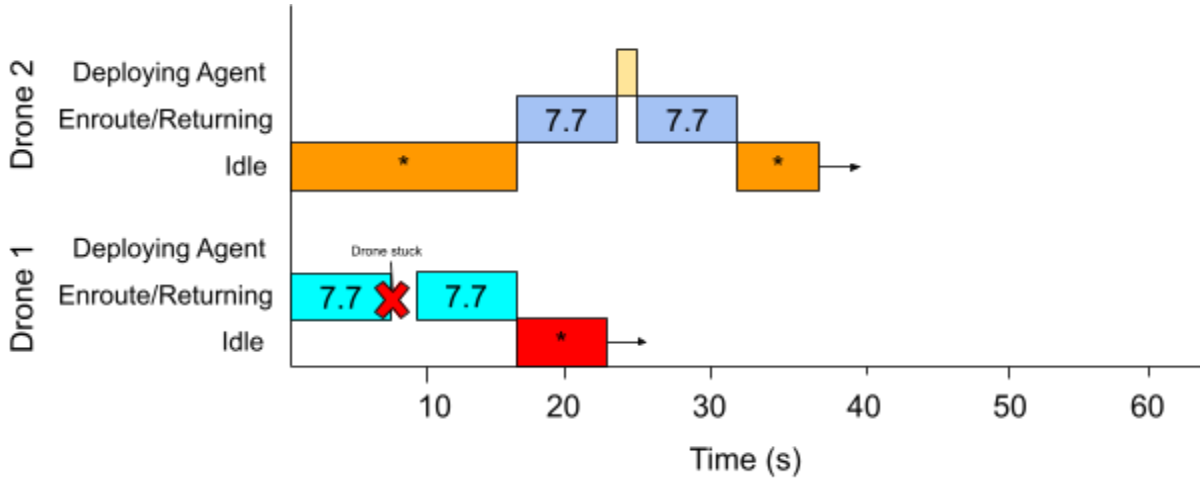
Timing Diagram 1: Zone 3 Low Severity Fire From Base at T=0
At 60 units per second, a single drone travels 22 seconds to zone 3. A low severity fire can be extinguished with a single drone in 1 second (10L/s). Note that the drone remains in the idle state until the next scheduled event.



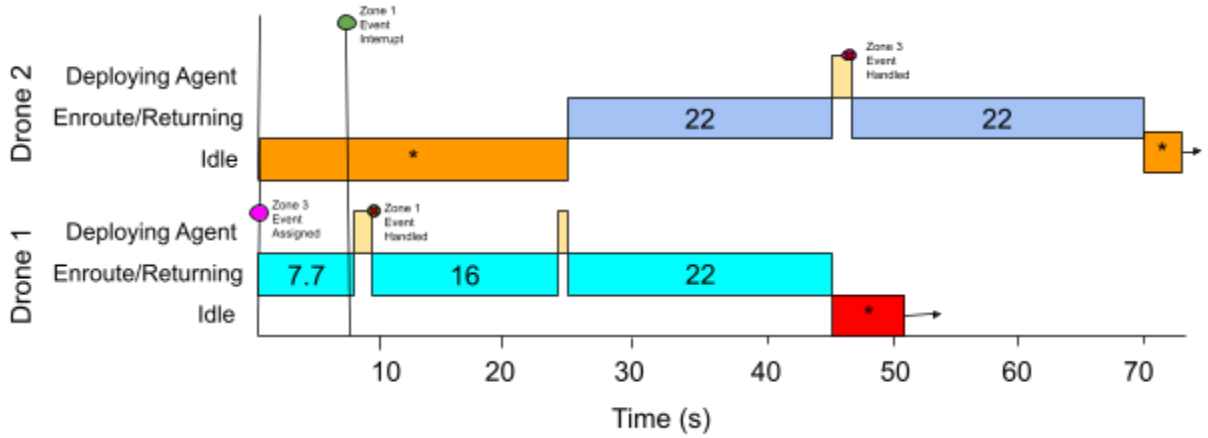
Timing Diagram 2: Zone 1 High Severity Fire, 2 Drones from Base at T=0
At 60 meters per second, it takes the drone 7.7 seconds to travel to zone 1. A high severity fire requires 30 L of agent to extinguish, which is the full capacity of 2 drones. Once the first drone deploys its agent, the second drone is signalled to finish the extinguishing to avoid the refill time of the first drone.



Timing Diagram 3: Zone 1 Low Severity at T=0 With Fault: Nozzle Jammed
At 60 units per second, it takes the drone 7.7 seconds to travel to zone 1 from base. A low severity fire only requires 10 L, or 1 drone. Drone 1 in this diagram has a fault, with its nozzle jamming at 15 s. This causes it to relinquish control of its event and return it to the queue, where the next drone is given the event. Drone 2 has the same travel time as drone 1, spends 1 second deploying its agent, then spends 7.7 seconds returning to base. Drone 1 remains disabled until it can be serviced (outside the scope of the program).



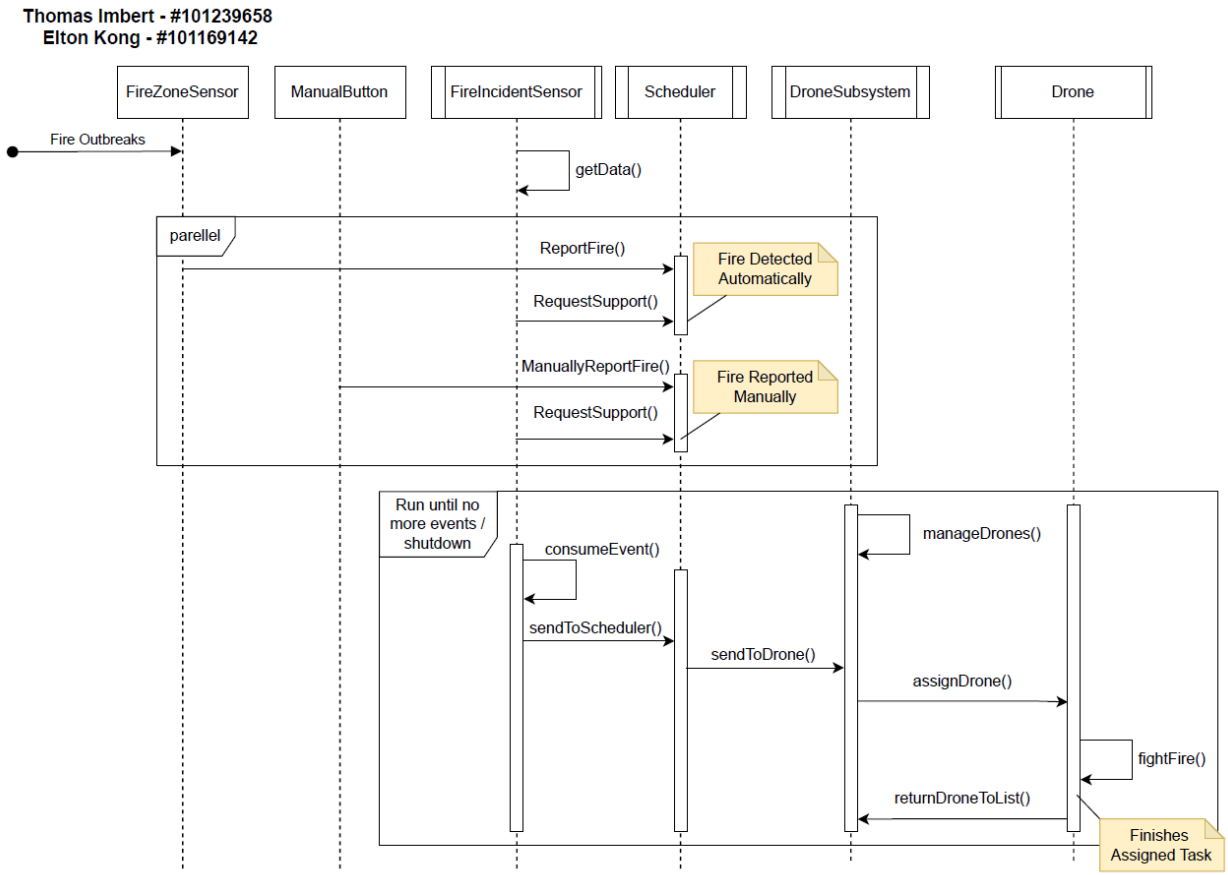
Timing Diagram 4: Zone 1 Low Severity at T=0 With Fault: Drone Stuck
At 60 units per second, it takes the drone 15 seconds to travel to zone 1 from base. A low severity fire only requires 10 L, or 1 drone. Drone 1 in this diagram has a fault, getting stuck in a tree. This causes it to relinquish control of its event and return it to the queue, where the next drone is given the event. Drone 2 has the same travel time as drone 1, spends 1 second deploying its agent, then spends 7.7 seconds returning to base. Drone 1 returns to base and is idle until it receives an event.



Timing Diagram 5: Zone 1 Low Severity (T=7.5) Interrupts Zone 3 Medium Severity (T=0)

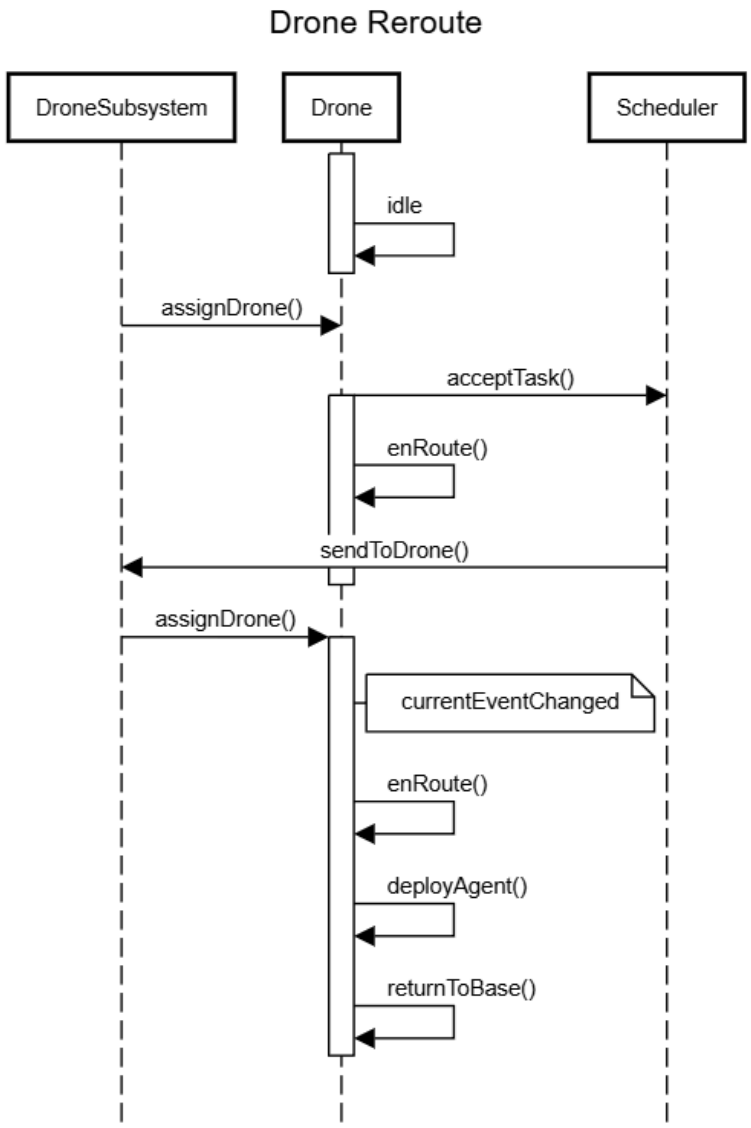
In this diagram, a medium severity fire in Zone 3 is assigned to Drone 1, then a low severity fire in Zone 1 is assigned while Drone 1 is en route to Zone 3. Drone 1 stops to handle the zone 1 event with 10 L of agent, then continues to zone 3. After deploying 5 L of extinguishing agent in zone 3, another drone is assigned the remaining event and uses 15 L to complete the event in zone 3.

2.3 Sequence Diagrams



Sequence Diagram 1: System Sequence Diagram

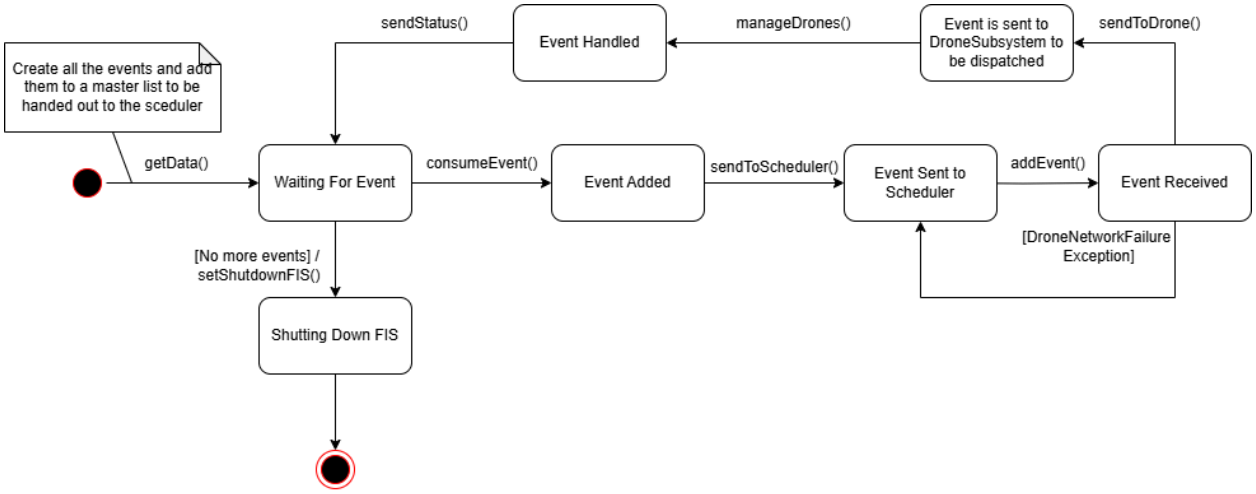
The sequence diagram above represents the entire fire fighting system. It models the particular events with the preceding entities to the designed software, with focus on the subsystems developed. The Fire Incident Subsystem and manual button function in parallel to request support from the scheduler. The Fire Incident Subsystem first gets data from the event file, then reports the events using ReportFire(), requesting support. Once requested, it consumes the event and sends it to the scheduler, which schedules it on arrival, and forwards the first event in its queue to the drone subsystem. An already running Drone Subsystem contains all the drones in the system and assigns events to the available drones using assignDrone(). After being assigned an event, an individual drone uses its fightFire() method, which causes it to progress from the idle state through its state machine, including travelling to the fire, deploying its agent, returning, and handling faults during the process.



Sequence Diagram 2: Drone reroute

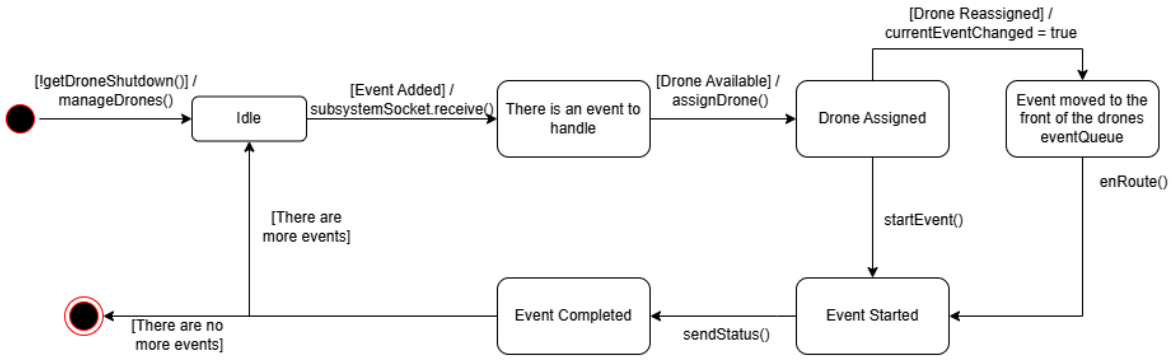
This sequence diagram represents a drone rerouting to deploy the extinguishing agent at an event in the zone it is passing through. It starts functioning as a normal routine, but receives an event from the DroneSubsystem through assignDrone() to change the current event. This brings the drone back to the enroute state to accommodate the new event, which then follows the regular state machine and returns to base.

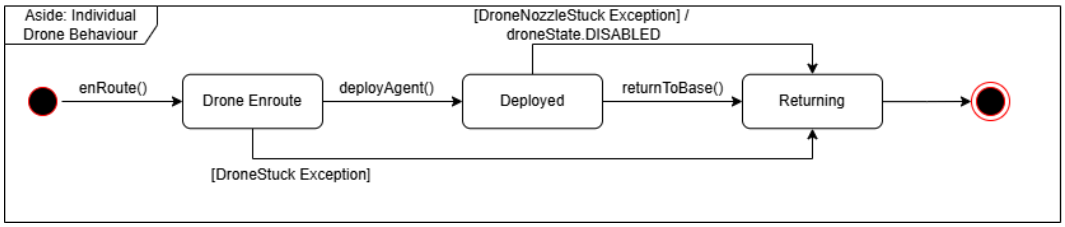
2.4 State Machine Diagram



State Machine Diagram 1: Events State Machine

This state machine diagram represents the finite state machine the events are designed around. The Fire Incident Subsystem (FIS) first creates a master list of events through getData() from a csv file. This transitions to the Waiting for Event state, which then uses consumeEvent() to add the event to the simulation and transition to the EventAdded state. The event is then sent to the scheduler using sendToScheduler(), and the scheduler's addEvent() method notifies the FIS that the event was received. If there is a fault, an exception is thrown and the FIS reverts to the event sent state. If no fault occurs, the event is sent from the scheduler to the drone subsystem, which assigns a drone to the event. If the drone accepts, it sends its status to the scheduler. Once all events in the FIS are consumed and sent to the scheduler, the FIS shuts down.





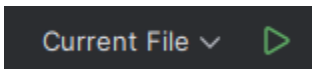
State Machine Diagram 2: Drone Subsystem State Machine

This state machine diagram represents the Drone Subsystem state machines, including the Drone helper class state machine. The Drone Subsystem waits in an idle state once initialized. Once an event is added, the subsystem assigns an available drone to the event, checking to see if there is a drone in the zone of the event. Once the event is assigned to a drone, the drone enters the En route state, and the subsystem lists the event as started. If the drone becomes stuck while enroute, it transitions to the returning state, and relinquishes control of the event, throwing a DroneStuck Exception. Once the drone reaches the zone of the event, it enters the deploy state. If the drone's nozzle jams, it disables, relinquishes control of the event, and returns to base for repairs. If the drone successfully deploys its agent it returns to base, entering the idle state and refilling upon return.

3.0 Instructions to Run and Test the System

To run the system, the use of an IDE such as IntelliJ is required.
To begin:

1. Open the project in IntelliJ as an IntelliJ IDEA Project.
2. In the src folder, locate three java class files:
    a. DroneSubsystem.java
    b. Scheduler.java
    c. FireIncidentSubsystem.java
3. Run the three classes in this order by setting the configuration to “Current File” and clicking the run button for each as shown below

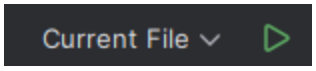
    

    1st: DroneSubsystem.java
    2nd: Scheduler.java
    3rd: FireIncidentSubsystem.java
4. The GUI should open after running the first file, and the simulation will begin when the FireIncidentSubsystem starts reading the event file.

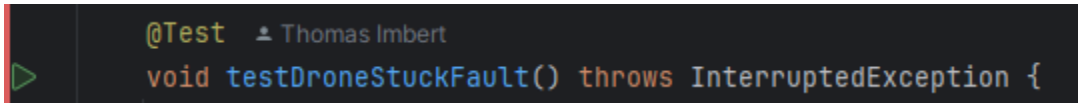
To test the system, the use of an IDE such as IntelliJ is required.
To begin tests: (~3m 16s total)

5. In the src folder, locate three java class files:
    a. DroneSubsystemTest.java
    b. SchedulerTest.java
    c. FireIncidentSubsystemTest.java

6. Run any of the three classes individually by setting the configuration to “Current File” and clicking the run button for each as shown below

7. Alternatively, individual tests from each class can be run by clicking the run icon next to their names, e.g.

4.0 Measurement Results

Table 6: Travel Times (s) for Scaled Simulation and Real Time Conversion

Drone Travel Times		
Zones	**Scaled Time (s)**	**Real Time (s)**
Zone 1	19.8	70.6
Zone 2	20.5	73.2
Zone 3	34.2	122.1
Zone 4	34.3	122.5
Zone 5	42.9	153.2

This table shows the travel times for drones in both the simulation (Scaled Time) and real time. The drone simulation runs at 3.52 times the speed of real time, as the simulation uses 60 m per second, and the real time speed of 60 km/h is 16.78 m/s.

4.1 Scaled Simulation Results

Table 7: Scaled Response Time Results and Averages (seconds)

	Zone 1	**Zone 2**	**Zone 3**	**Zone 4**	**Zone 5**
	34.450	23.917	38.622	39.876	50.865
	32.413	24.677	38.708	38.065	27.578
	25.562	12.783	37.147	37.123	45.539
	17.989	12.736	42.222	44.843	49.678
	26.813	22.227	41.024	40.818	48.395
	27.850	17.053	23.001	34.561	27.274
		25.837	22.839	26.534	48.749
				25.304	
Average	**27.51**	**19.89**	**34.79**	**35.89**	**42.58**

This table shows the data (in seconds) for response times to all events in each zone utilizing the provided final event file (Final_event_file.csv). Zone 2 had the shortest overall response time as one of the two closest zones to base. Zone 5 had the longest response times, as it is the farthest from base and does not benefit from drones

re-routing while passing through it. The total average response time was calculated to be 32.13 seconds, which means that events required an average of 32 seconds to be handled in the simulation.

4.2 Real Time Simulation Results

Table 8: Real Time Simulation Results (seconds)

	Zone 1	Zone 2	Zone 3	Zone 4	Zone 5
	123.772	85.929	138.762	143.267	182.749
	116.454	88.660	139.071	136.760	99.083
	91.840	45.927	133.462	133.376	163.613
	64.631	45.758	151.696	161.113	178.484
	96.334	79.857	147.392	146.651	173.874
	100.060	61.268	82.638	124.171	97.990
		92.828	82.056	95.332	175.146
				90.913	
Average	**98.85**	**71.46**	**125.01**	**128.95**	**152.99**

Table 8 above depicts the scaled time values from Table 7 converted to real time values. This was done by multiplying each value by the scaling factor. This gives a new average of 115.45 seconds for an event to be handled in real time.

4.3 Total Simulation Time and Throughput

The run time for the simulation was 8 minutes and 5 seconds, or 8.083 minutes. This is approximately 1 minute per hour of events. The start of the simulation is at 12:40 a.m., which cuts 40 minutes from the simulation time to save startup time prior to events happening. The throughput of the simulation was 35 processes in 8.083 minutes, giving

Eq 1. Throughput Calculation

$$T = I/\Delta t = 35\ events/8.083\ min = 4.33\ events/min$$

5.0 Conclusions

The drone project was a relatively interesting project to help reinforce concepts learned in the course. It gave an opportunity for the group to work in a realistic workforce team environment on a large scale iterative project, applying both the theory of designing a real-time concurrent system and the simulation of one.

The beginning of the project was slower than anticipated, as a former group member left the course, leading to a greater responsibility for each individual group member. The decisions to let each member work to their strengths at this point allowed us to get to know each other and communicate better before more difficult features needed to be implemented. Initially, the system used thread synchronization between components, which is why it was best implemented by fewer people at this point. In hindsight, this was the optimal method of operation at that point.

Once the foundation was laid and the group began to know each other better, we began to work on more of the project collaboratively. Suggestions, design decisions, and brainstorming allowed for every member to discuss idea implementations and ultimately decide on the final implementations of different methods and algorithms. By iteration 2, the team introduced ideas such as a simulation control class, a drone helper class, and event scheduling algorithms.

Upon reaching the final iteration of the project, each group member now has a solid understanding of the system developed, has a reinforced understanding of the concepts covered in the course, and has practical experience in the application of course material.

The two hurdles that were had in the group were missing a full implementation of drones rerouting to deal with a new event in a zone they were passing through, and missing timing diagrams. The group dealt with these by giving them special attention for the next iteration and affirming the completion to ensure the next features were properly implemented.

Overall, Team 3 is proud of our project and had a good experience working as a group and creating a new project for our portfolios. We endorse this project over traditional labs to learn how to create a real-time concurrent system and the different methods that can be used in its operation.