# SYSC 3303 A2 - Final Project Report

Group 2
Farhan Mahamud - 101147861
William Forrest - 100803271
Jacob Hovey - 101163798
Subear Jama - 101154626

# Breakdown of Responsibilities

For all of the iterations of this project, we split the project into four major components, with each group member taking ownership of a single component and its test cases. The code and tests were split into the following components:

- The floor module was “owned” by Subear
- The scheduler module was “owned” by Jacob
- The elevator module was “owned” by Farhan
- The simulation module, which we added to read and understand input files, as well as manage things such as realtime tracking and our GUI, was “owned” by Will

Although our approach to coding was to take ownership of our designated modules, we put heavy emphasis on collaboration and understanding of the entire system by every team member. We had several in-person and discord meetings to discuss our updates and code together over the course of the iterations to ensure our modules would work together correctly, for example being able to send and read update packets between the modules. In these meetings, we also often did pair programming or shared advice to debug issues together. While we had ownership of the specified classes, over the course of the iteration we all collaborated and added code to various different parts of the system. Particularly, we often created test cases for other modules in order to check each other's work, as well as sometimes adding functions or sections to other modules to help with implementing certain features whose scope crossed over between all modules.

Beyond coding, we also had various other responsibilities in documentation and diagrams that we had to keep up to date every iteration. These requirements include:

- A readme providing information on our code
- This work breakdown
- Instructions on running and testing our system
- Class, sequence, state and timing diagrams
- For this final iteration only, this report

With the external requirements, we split these various responsibilities each iteration in a way that seemed as fair and even as possible. As the requirements of the iteration change, we also adapted perfectly to meet the requirements making changes when needed to. For example, in iteration 3, there were significant changes to be made in our program to adapt the UDP network communication model. With that in mind, each of us did a class, sequence and state diagram relevant to our modules as we had the most knowledge of those sections. Our approach was to complete the portion that we had

taken ownership of, and then get the rest of the team to review it. Everyone went over every section, and if there were any issues or anything the creator needed to include, we shared suggestions and worked to finalize all of the documents in a way that we were all satisfied with.

Ultimately, our main goal was to ensure that we split the work each iteration in a way that we felt was as fair and even as possible. We worked to foster as much collaboration and communication as possible, to help each other out and all work together to create the best system we could as a team.

# Diagrams

## Class Diagrams

Higher quality images are available in the diagrams folder.

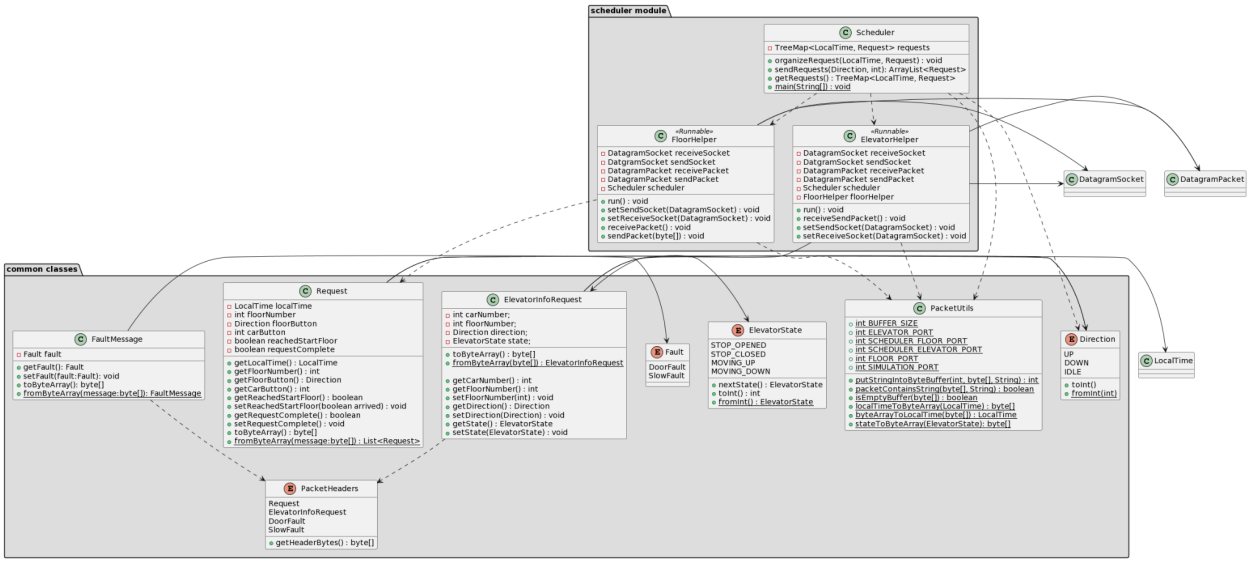



Figure 1. Scheduler Class Diagram

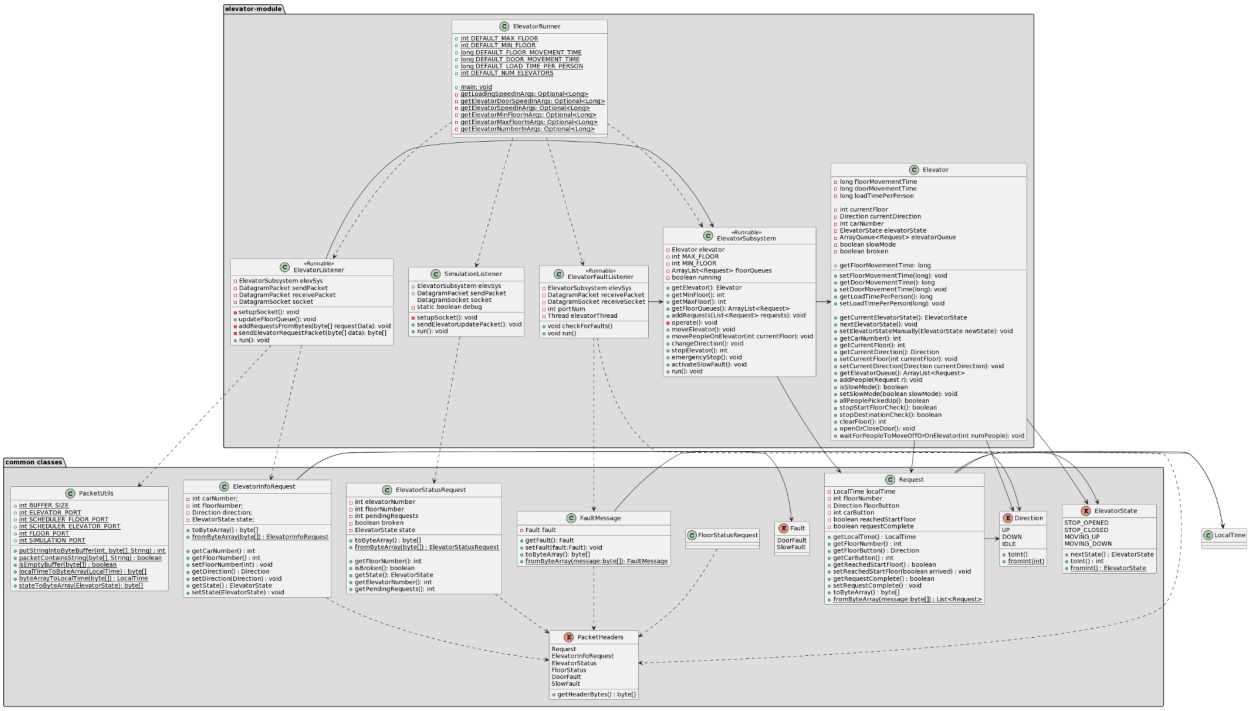



Figure 2. Elevator Subsystem Class Diagram

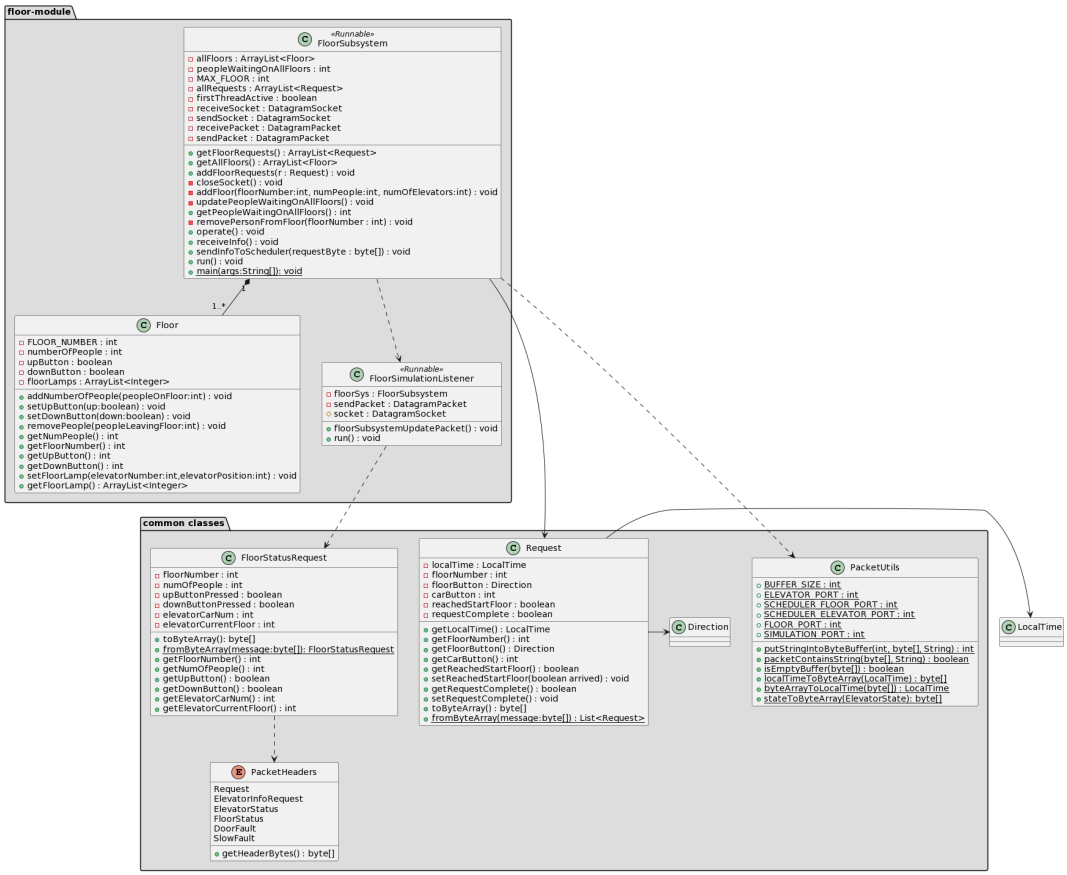



Figure 3. Floor Subsystem Class Diagram

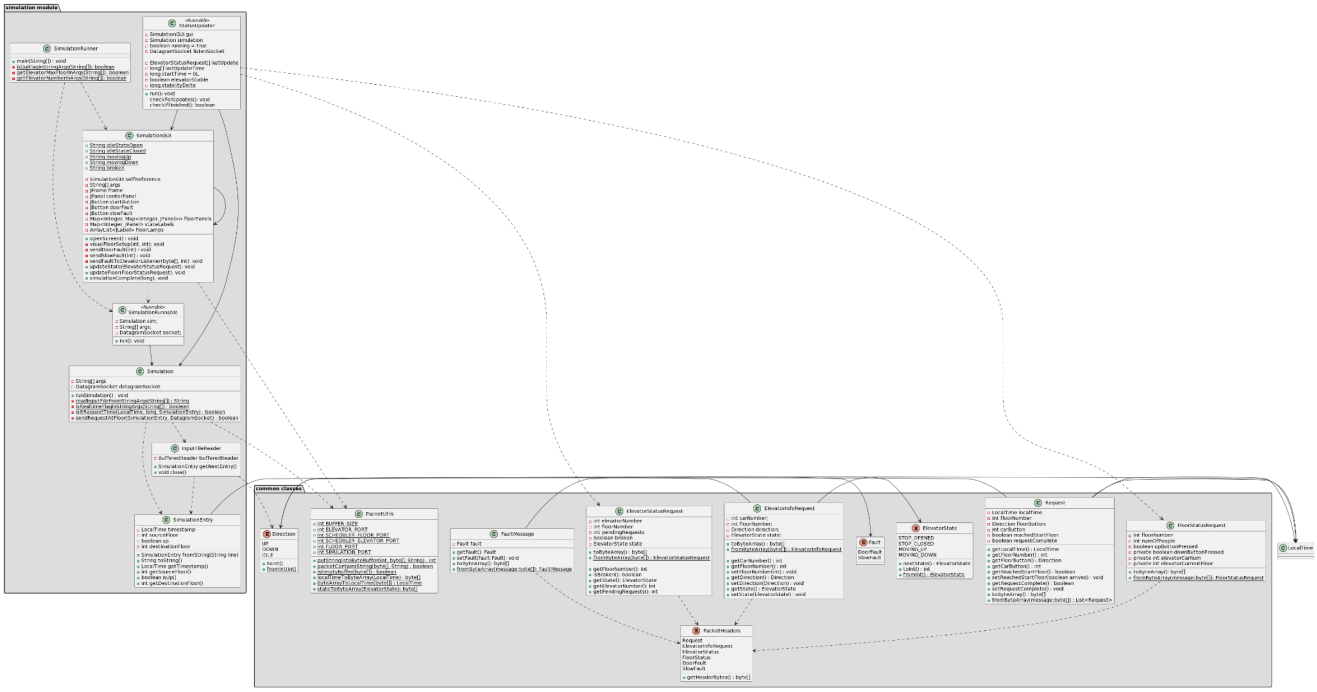



Figure 4. Simulation Class Diagram

# Sequence Diagrams

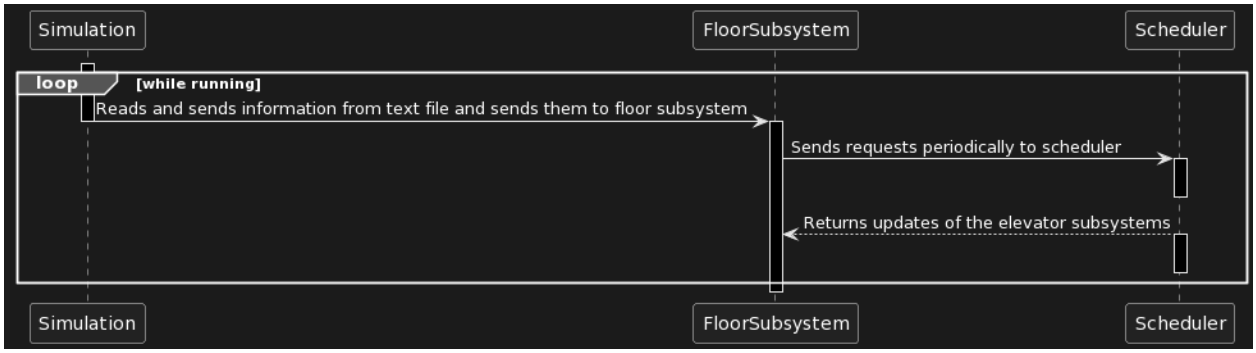



Figure 6. Sequence diagram for the call elevator action.

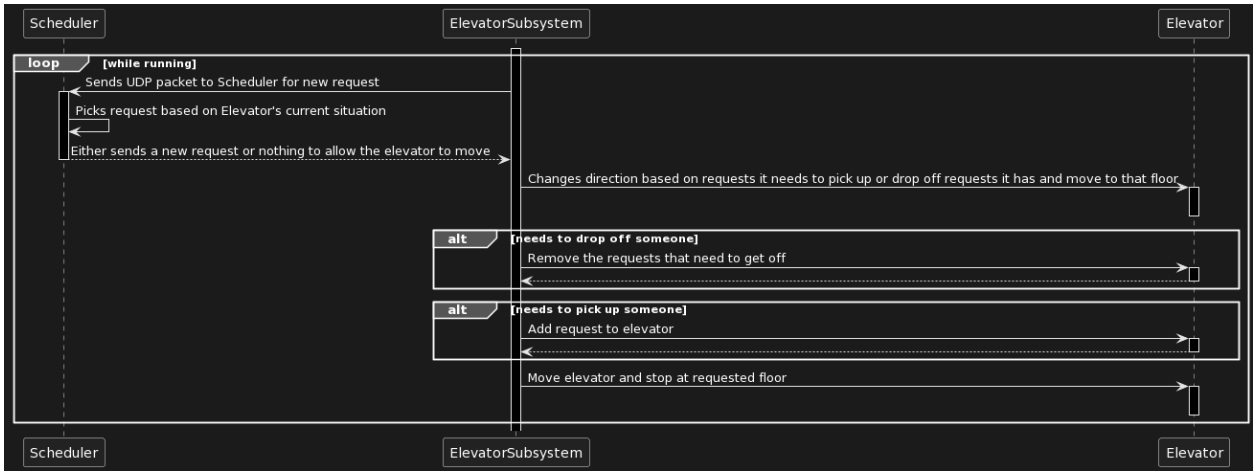



Figure 7. Sequence diagram for the elevator arrived action.

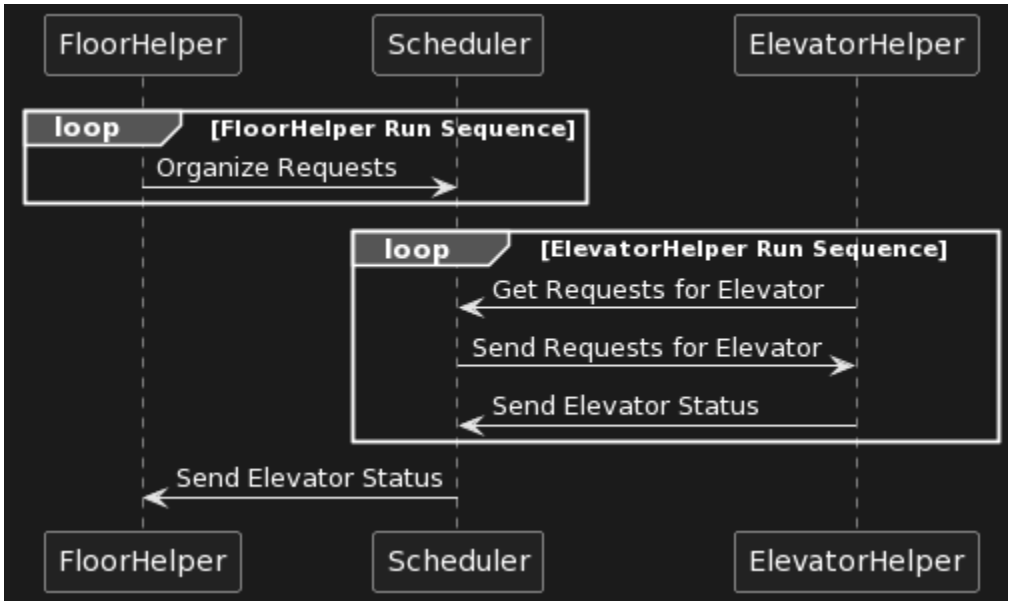



Figure 8. Sequence diagram for the internal scheduler threads.

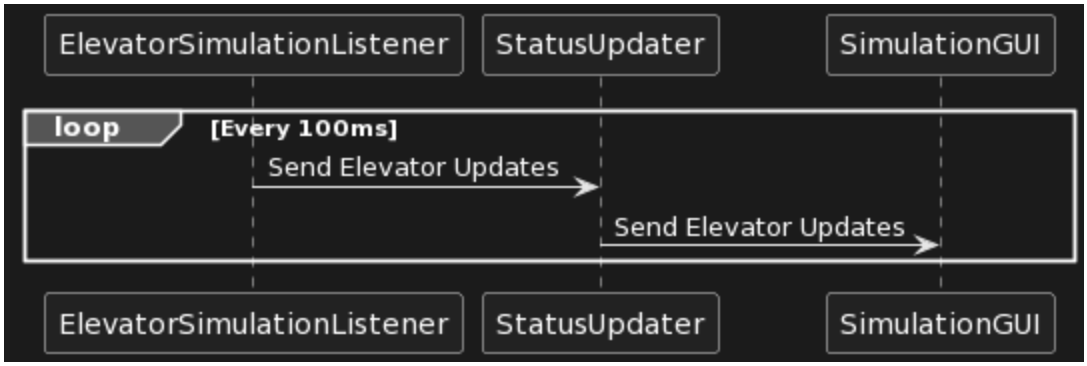



Figure 9. Sequence diagram for the elevator subsystem to send status updates to the simulation.

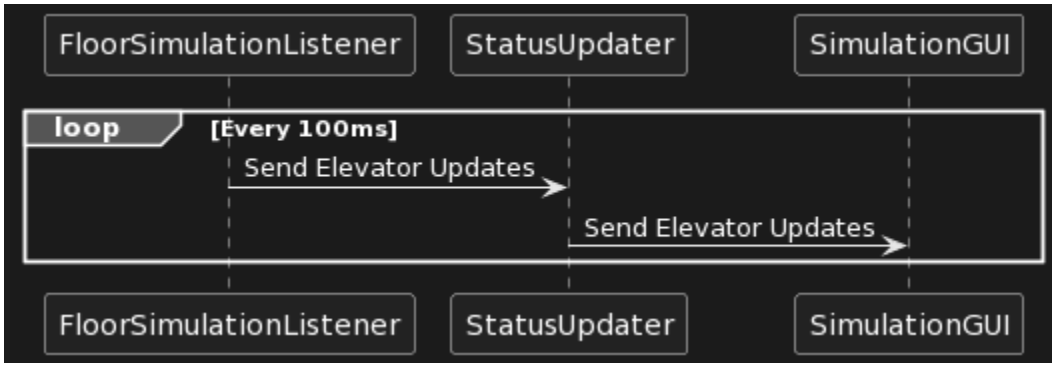



Figure 10. Sequence diagram for the floor subsystem to send status updates to the simulation.

# State Machine Diagrams

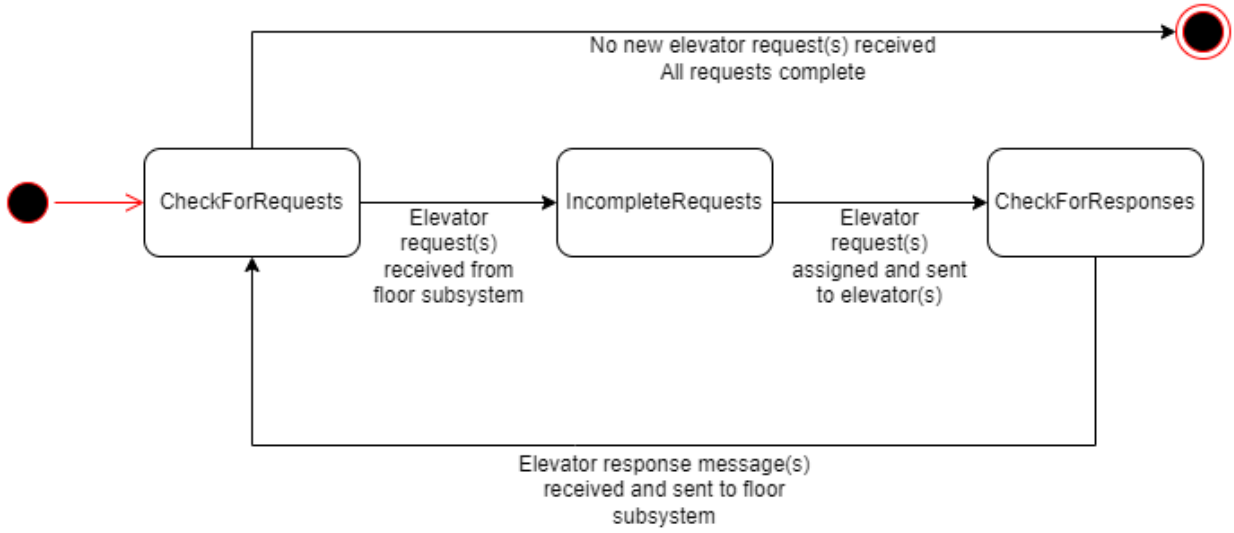



Figure 11. State machine diagram for the scheduler.

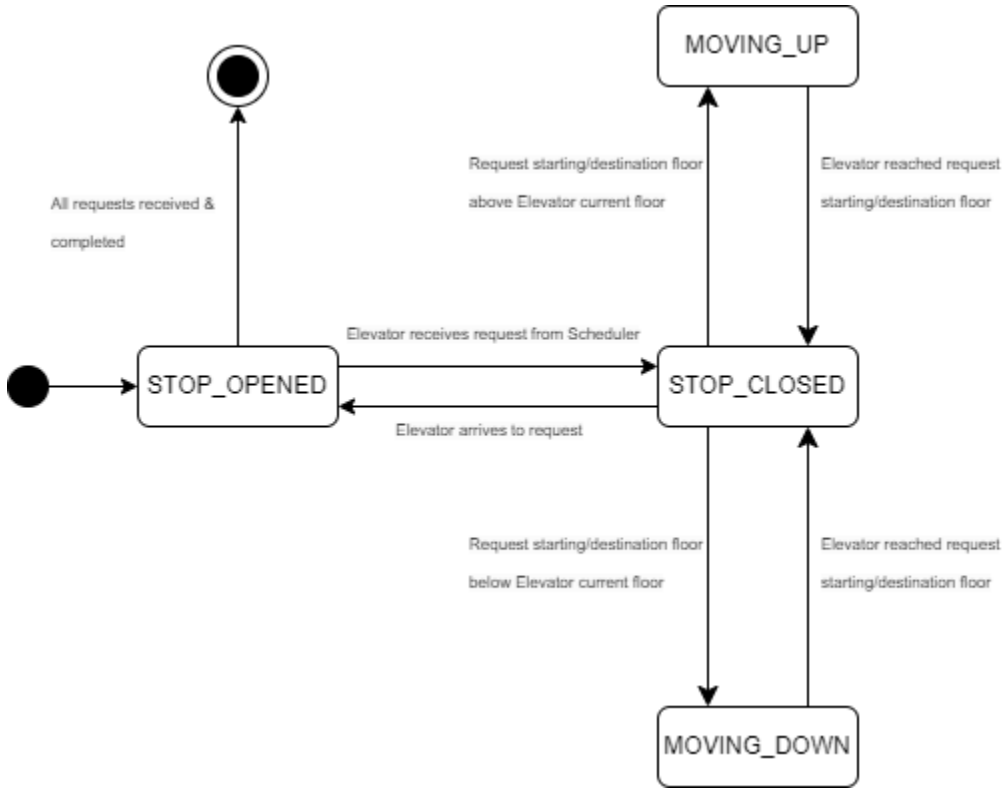



Figure 12. State machine diagram for the elevator subsystem.

## Timing Diagram


Figure 13. Timing diagram for the elevator system.

# Set Up and Test Instructions

Requirements

- Eclipse 2022 (any edition should be fine)

- Java 17 or higher

How to run GUI

- Right click SimulationRunner.java

- Click “Run As” -> “Run Configurations”

- Click the “Arguments” tab on the right panel

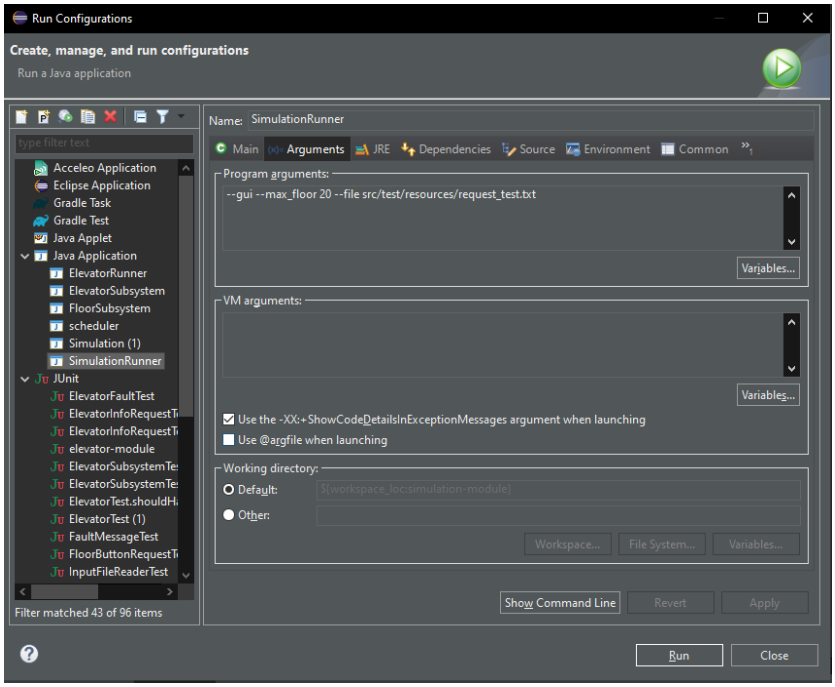
- In “Program arguments”, type “--gui”

- Also type “--file src/test/resources/x.txt”, where x is the name of the input file you want to run, which MUST be in the src/test/resources folder. There is a default file that will be run if you do not enter any specific file

- Click “Apply”

- Run Scheduler.java, ElevatorRunner.java, and FloorSubsystem.java in any order

- Finally, run SimulationRunner.java and the GUI should open



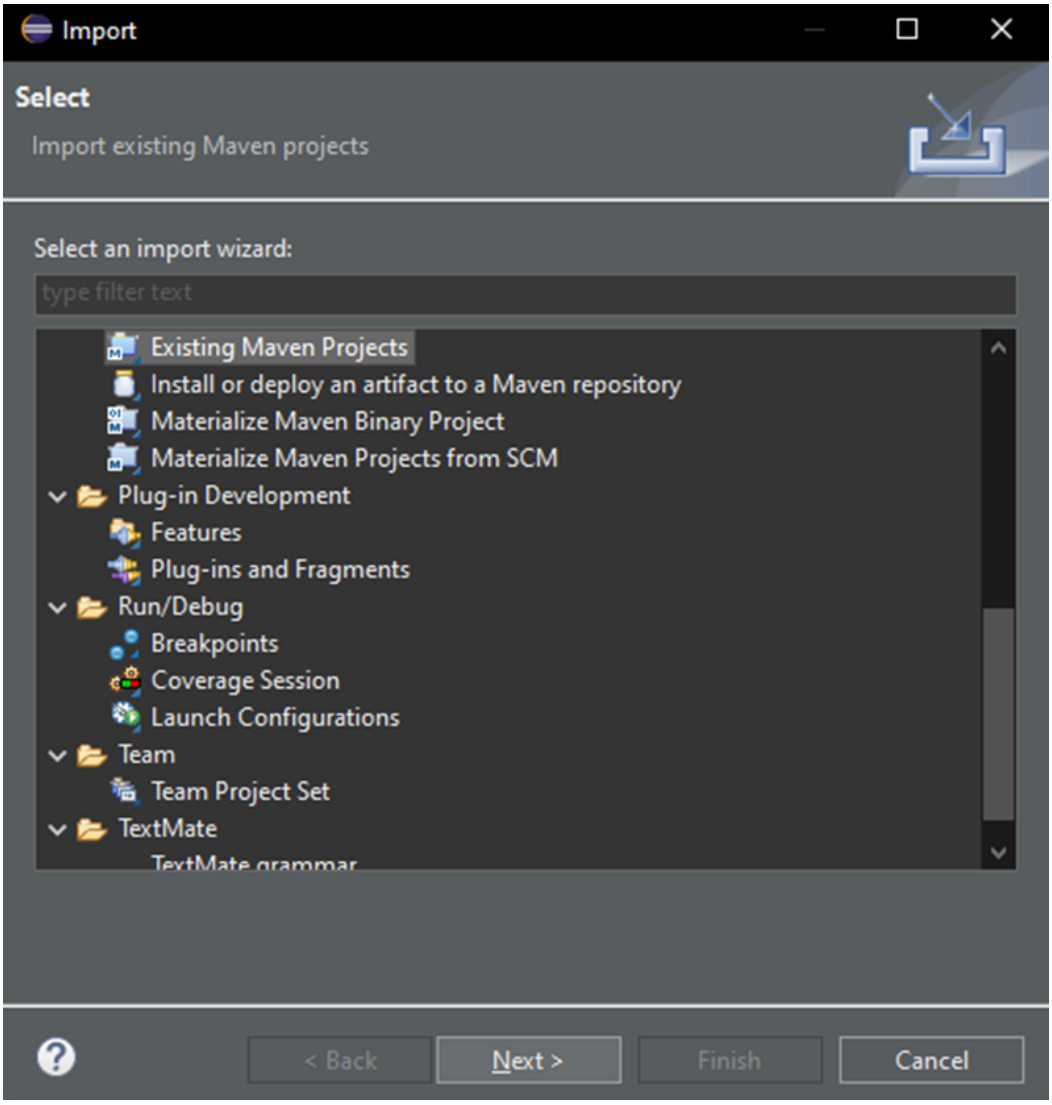

How to send faults

- A slow fault can be sent at any time by pressing the button. It will cause the specified elevator to run at half its regular speed. The sensors will then catch this irregular behaviour and stop that elevator entirely, marking it as “broken”.

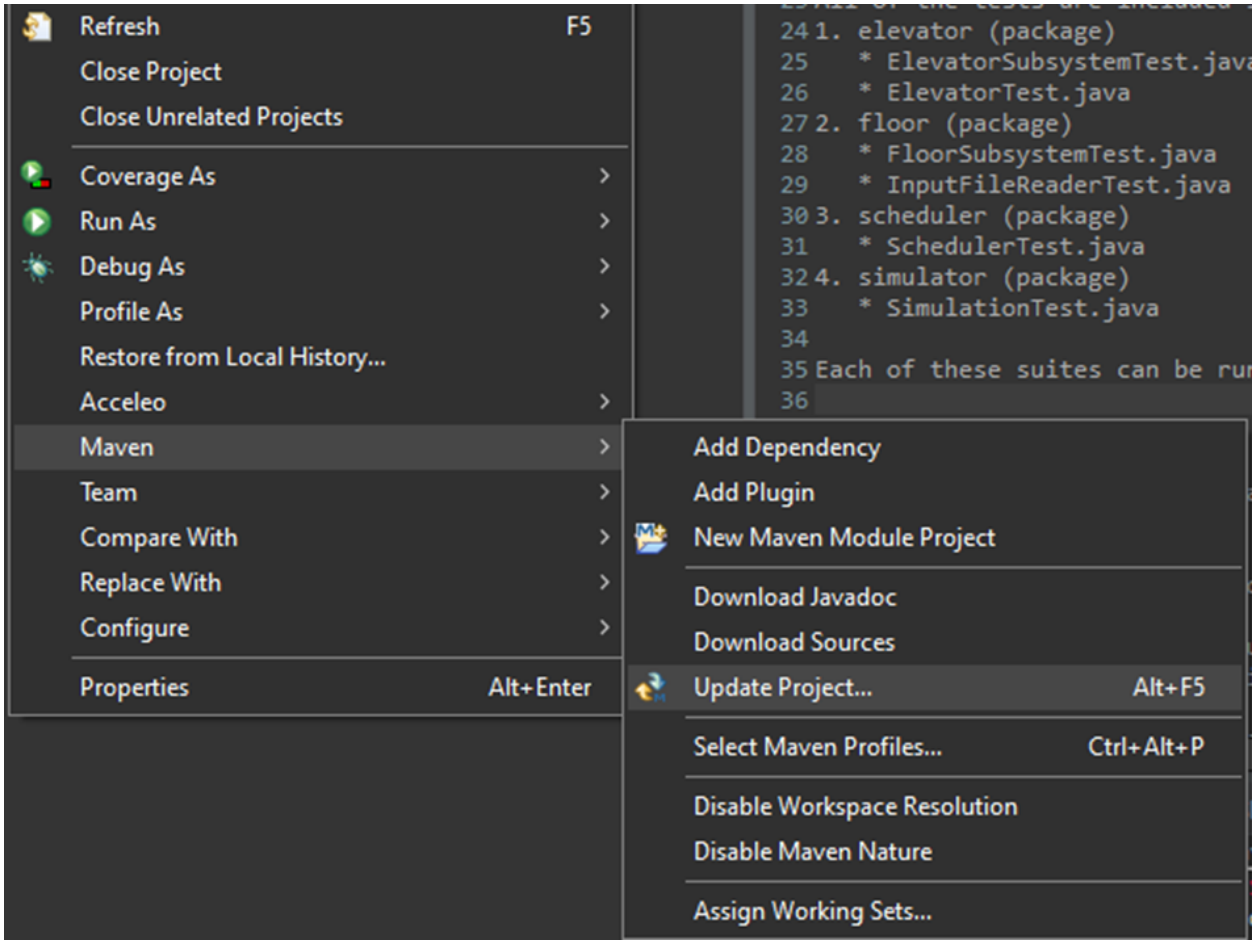
- A door fault can be sent at any time, but will only work if sent when a door is closed and transitioning to open or open and transitioning to closed. It will effectively reset the closing/opening time, as the action was interrupted and has to restart.

How to run tests

- Import the java project into eclipse using the "Import maven project" option in the eclipse workspace

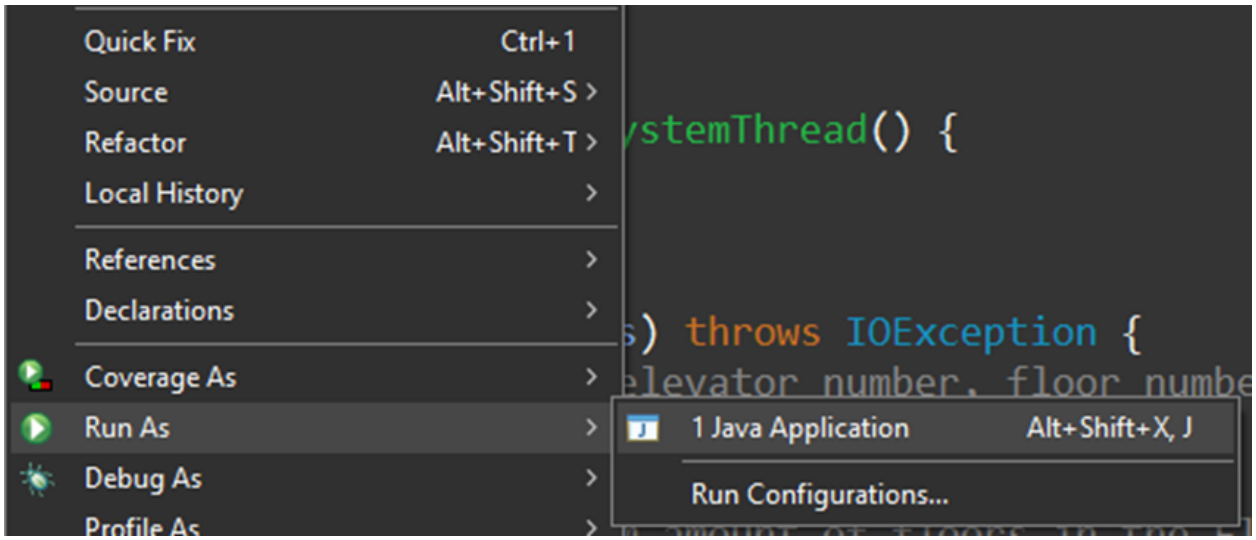



- Right click the project in the project explorer, select “update maven project” to download any dependencies for testing (Or hit alt-f5 if your using default keybindings)




- Navigate to the "Simulation.java" file in the simulation package.

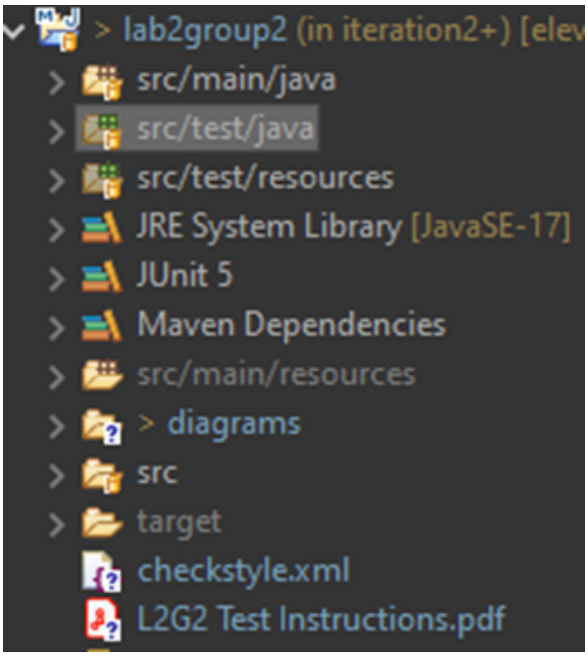
- Right click this file, and tell eclipse to run as java.



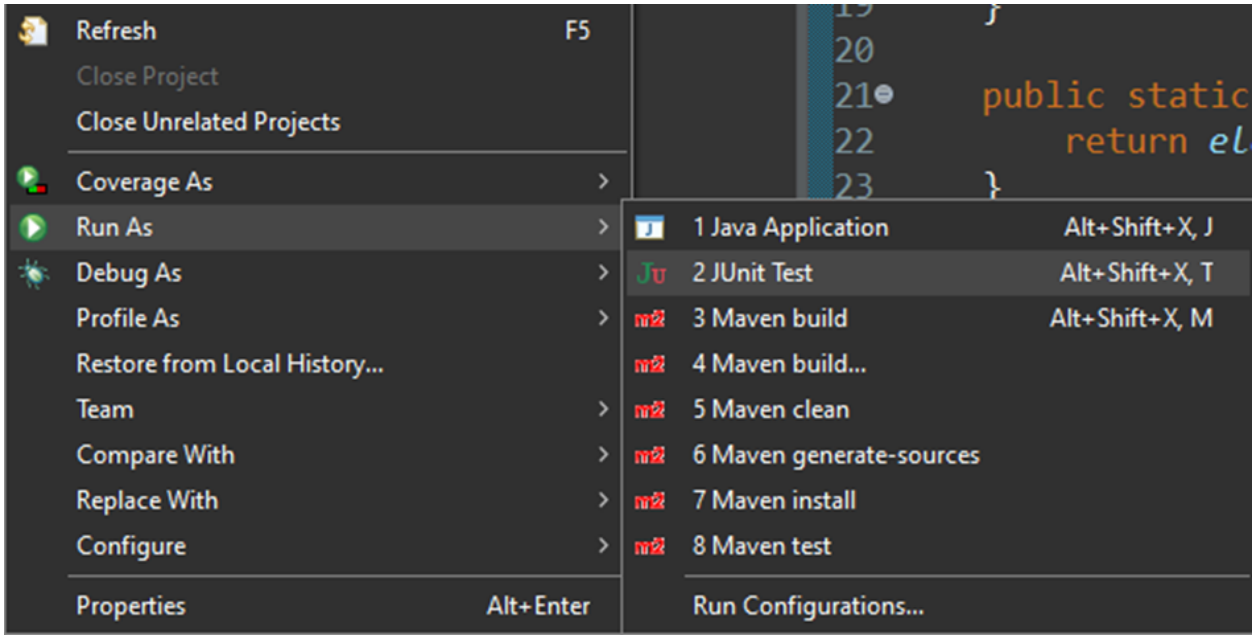

- No command line arguments are required at this time

How to run tests in eclipse

- Right click the "tests" folder in the project explorer for any submodule (The parent module has no tests to run!).



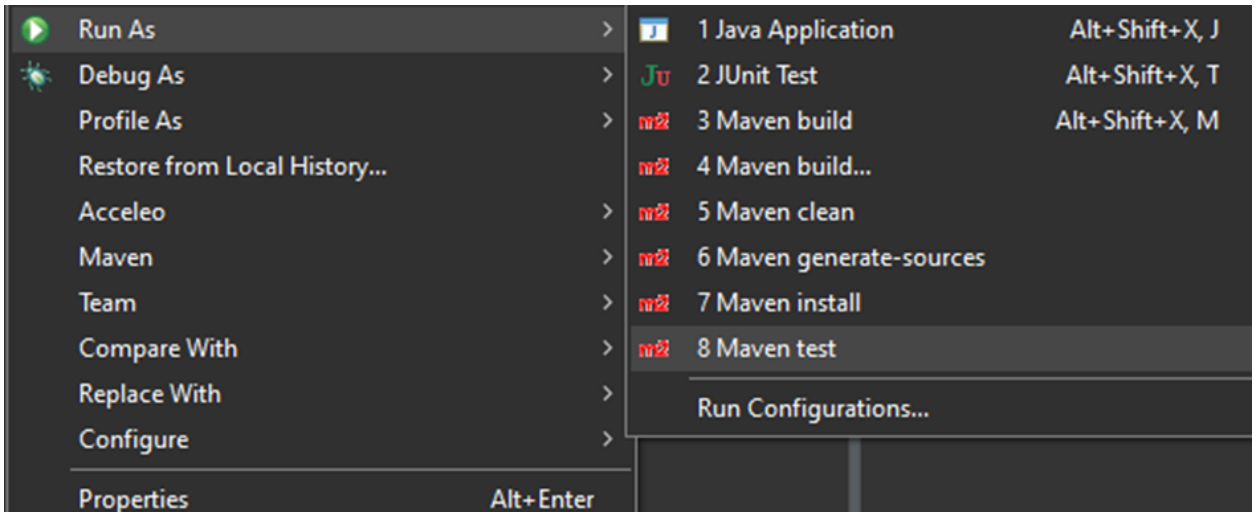

- Select "run as JUnit tests"



Running all tests with maven:

Because there are now separate sub-modules, running all of them at once is easier to perform using maven.

Select “Maven test” on any module in the project, in the project explorer panel.




# Measurement Results

Below is the raw data and calculations for our measurements taken on a real world elevator system in iteration 0:

Floor height: 4m
Deceleration time (7->1) 4.5s (Going down)
Acceleration time (1->7) 3.5s (Going up)

Floor timings:

| Floors | Number of floors changed | Direction | Time | Average Speed | Average Acceleration |
|---|---|---|---|---|---|
| 1->2 | 1 | Up | 8.34s | 0.47m/s | 0.056 m/s^2 |
| 2->1 | 1 | Down | 8.01s | 0.49m/s | 0.061 m/s^2 |
| 4->5 | 1 | Up | 9.55s | 0.42m/s | 0.044 m/s^2 |
| 5->1 | 4 | Down | 14.53s | 1.10m/s | 0.076 m/s^2 |
| 1->7 | 6 | Up | 23.53s | 1.02m/s | 0.043 m/s^2 |
| 7->1 | 6 | Down | 21.65s | 1.10m/s | 0.051 m/s^2 |

| | | | | | |
|---|---|---|---|---|---|
| 2>3 | 1 | Up | 9.34s | 0.42m/s | 0.045 m/s^2 |
| 3->2 | 1 | Down | 8.48s | 0.47m/s | 0.055 m/s^2 |

Fastest time between floors: 2.04s (This was obtained by lapping between each floor while moving floor 1->7 and then taking the shortest time between floors, in this case it was floor 3->4)

Doors:
Opening: 3.00s
Closing: 2.41s

Loading:

| People | Time | Time - doors |
|---|---|---|
| 1 | 8.57s | 2.57s |
| 1 | 6.95s | 1.45s |
| 3 | 10.06 | 4.56s |

Loading time = opening + closing + people factor
Loading time = 5.5s + people * 1.5s = 5.5 + 1.5p

**CALCULATIONS:**

**Max Speed**

$Max\ Speed\ = \frac{Floor\ height}{Fastest\ time\ between\ floors} = \frac{4m}{2.04s}$

$Max\ Speed\ =$ 1.96m/s

**Acceleration (Maximum distance travelled in elevator)**

$acceleration\ = \frac{Max\ speed}{acceleration\ time} = \frac{1.96m/s}{3.5s}$

$acceleration\ =$ 0.56m/s$^2$

**Deceleration (Maximum distance travelled in elevator)**

$deceleration\ = \frac{Max\ speed}{acceleration\ time} = \frac{1.96m/s}{4.5s}$

$deceleration\ =$ 0.43m/s$^2$

**FOR ALL FLOORS (1->7 & 7->1) At 95% confidence**

Velocity Mean = $\frac{\Delta d}{\Delta t}$ = (1.02 + 1.10) / 2 = 1.06 m/s

$$Variance = S = \sqrt{\frac{\sum_{i=1}^{n}(xi-\overline{X})^2}{n-1}} = \sqrt{\frac{(1.02-1.06)^2+(1.10-1.06)^2}{2-1}} = 0.0565m/s$$

Calculating the 95% Confidence Interval for Average Velocity

Formula: $\overline{X} - t_{\alpha/2n-1} * \frac{S}{\sqrt{n}} < \mu < \overline{X} + t_{\alpha/2n-1} * \frac{S}{\sqrt{n}}$

$\mu_{velocity} = 1.06 \pm \frac{0.0565}{\sqrt{2}} \times 12.706$

$\therefore \mu_{velocity} = 1.06 \pm 0.5076$ or $0.5524m/s < \mu_{velocity} < 1.5676m/s$

Assume that acceleration is constant going up and down

**FOR EACH FLOOR At 95% confidence**

Velocity Mean = $\frac{\Delta d}{\Delta t}$ = $\frac{0.47+0.49+0.42+0.42+0.47}{5} = 0.454m/s$

$$Variance = S = \sqrt{\frac{\sum_{i=1}^{n}(xi-\overline{X})^2}{n-1}}$$

$$S = \sqrt{\frac{(0.47-0.454)^2+(0.49-0.454)^2+(0.42-0.454)^2+(0.42-0.454)^2+(0.47-0.454)^2}{5-1}}$$

$$S = \sqrt{\frac{(0.000256+0.001296+0.001156+0.001156+0.000256)}{4}}$$

$S = 0.032m/s$

Calculating the 95% Confidence Interval for Average Velocity

Formula: $\overline{X} - t_{\alpha/2n-1} * \frac{S}{\sqrt{n}} < \mu < \overline{X} + t_{\alpha/2n-1} * \frac{S}{\sqrt{n}}$

$\mu_{velocity} = 0.454 \pm \frac{0.032}{\sqrt{5}} \times 2.571$

$\therefore \mu_{velocity} = 0.448 \pm 0.0368$ or $0.4112m/s < \mu_{velocity} < 0.4848m/s$

Calculating the 95% Confidence Interval for Average Acceleration

Acceleration Mean = $\frac{\Delta d}{\Delta t}$ = $\frac{0.056+0.061+0.044+0.045+0.055}{5} = 0.0522m/s$

$$Variance = S = \sqrt{\frac{\sum_{i=1}^{n}(xi-\overline{X})^2}{n-1}}$$

$$S = \sqrt{\frac{(0.056-0.0522)^2+(0.061-0.0522)^2+(0.044-0.0522)^2+(0.045-0.0522)^2+(0.055-0.0522)^2}{5-1}}$$

$$S = \sqrt{\frac{(0.000014444+0.00007744+0.00006724+0.00005184+0.00000784)}{4}}$$

$S = 0.0074m/s^2$

$\mu_{acceleration} = 0.0522 \pm \frac{0.0074}{\sqrt{5}} \times 2.571$

$\therefore \mu_{velocity} = 0.0522 \pm 0.0085$ or $0.0437m/s < \mu_{velocity} < 0.0607m/s$

**ELEVATOR LOAD Times Calculating Average, Variance & 95% Confidence**

| People | Time | Time - doors |
|---|---|---|
| 1 | 8.57s | 2.57s |
| 1 | 6.95s | 1.45s |
| 3 | 10.06 | 4.56s |

$average\ load\ time\ = \frac{8.57 + 6.95 + 10.06}{3} = 8.53s$

Variance load = $\sqrt{\frac{\sum_{i=1}^{n}(xi - \overline{X})^2}{n-1}} = \sqrt{\frac{(8.57-8.53)^2 + (6.95-8.53)^2 + (10.06-8.53)^2}{3-1}} = 1.55s$

Confidence = $\mu_{load} = \overline{X} - t_{\alpha/2n-1} * \frac{S}{\sqrt{n}} < \mu < \overline{X} + t_{\alpha/2n-1} * \frac{S}{\sqrt{n}}$

$\therefore \mu_{load} = 8.53 \pm 3.8504$

# Reflections

Overall, we are very pleased with the results of our elevator design. We are very happy with the scheduler and elevator optimization logic that we implemented, which allows our system to run smoothly, handling and carrying multiple requests at a time and splitting requests between elevators intelligently based on position and state. We liked the full coverage of our listener threads, and the useful communication between all modules of the project. We liked the addition of the extra module, the simulation, which helped us keep our project very clean and organized. We also liked the clear and useful print statements we implemented into the logs for each of our modules, and we are very proud of how the GUI provides such a useful and clear visual representation of the state of all elevators and the system as a whole. Finally, we liked the ability to rerun our system again after it has completed, from any position with elevators on any floors, seamlessly without issue or failure.

There are, however, some aspects of our system that we feel we could improve upon or change if we were to begin again with the knowledge we have accumulated. The biggest change we would make would be to break up our elevator operate loop into smaller and cleaner sections, to better follow the single responsibility principle. We would also try to build our scheduler differently so that it would send a request to the closest elevator if multiple are idle and looking for jobs. As it is with our current system where it responds to packets from the elevator with prioritized requests, it can sometimes send to a farther elevator if it pings first. Some other things we think could be interesting to add if we had more time or began again would be automatic network discovery to run the modules easily on multiple systems, removing sleeps in

the elevator to replace them with an active motor thread, and breaking up the GUI into more than one class so it is more customizable and follows the single responsibility principle.