Elevator Management System

CPS 406 - Final Document

Prof. Abhari

Group 9

Section 9

Group Members: Tyler Diep, Youssef Mezher, Juan Pires,

Griffin Ashby, Joshua Siraj

Joshua Siraj	Backend/Lead Developer	20%
Tyler Diep	Frontend Developer	20%
Youssef Mezher	Frontend Developer	20%
Juan Pires	Backend Developer	20%
Griffen Ashby	Frontend Deveoper	20%

TABLE OF CONTENTS

Initial Software Management Plan

Initiation of the System:

The first task, with high motivation, was to choose a name for this team. We believed that anything ending with an “X” resonated with efficiency and confidence, such as ElevatorX and AscendX. After much thought, we decided to stay with the elevator management system, and continued to plan out the major components.

The initial plan of the front end was displaying an UI that shows different floors, then pressing a button will move the elevators. A list with the types of buttons to be implemented are as follows:

- Call buttons (Up and Down)
- Floor Number (1,2,3,4,5, etc.)
- Emergency Stop (Stop elevator at its spot)

These buttons were general to a real elevator, and ones that are the most essential when it comes to taking a lift. Elevators also involved directions, wait times, priorities in terms of providing the service, etc. The logic for all these functions required some back end technology such as:

- Deciding which floor the elevators will go to
- Input the number of floors and elevators, along with the UI and logic adapting accordingly
- When the elevator reaches the floor, it opens and then closes, then a prompt to call the floor number appears and the elevator moves accordingly
- An emergency stop can be called to halt its position and stop before reaching the requested floor

We decided that the spiral model with an incremental approach would fit best for our project since it would take a lot of trial and error how this elevator would function, especially the

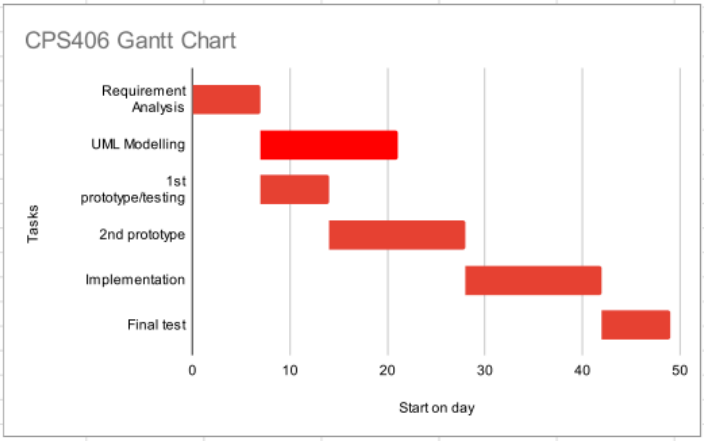
initial functionality to add extra floors/inputs was implemented (an action that we did not decide to implement in the final prototype).

Responsibilities alongside Gantt Chart

The Gantt Chart represents the task division throughout the semester. Consisting of a spiral model approach, it follows the small incremental approaches after every week generated progress, as we stayed on track, as well as catching errors quickly before it was too late. Although the Gantt chart lays out the schedule for lab tasks, the team had to dedicate time outside of class to understand underlying complexities to an elevator system, and make sure that the proper resources were prepared before the next team meeting. Multiple meetings were held to discuss the progress of our application, and as a team, responsibilities were carried out evenly to work as a single unit. Implementing the prototypes took the longest time, because it had to be managed properly to align with our requirements.

Task	Start date	End date
Requirement Analysis	Feb-6	Feb 13
UML modelling	Feb 13	Feb 27
1st prototype/test 1	Feb 13	Feb 20
2nd protoype	Feb 20	Mar 6
Implementation	Mar 6	Mar 20
Final test	Mar 20	Apr 3

Task	Start on day	Duration
Requirement Analysis	0	7
UML Modelling	7	14
1st prototype/testing	7	7
2nd prototype	14	14
Implementation	28	14
Final test	42	7



Requirements Analysis

The elevator management system aims to provide demonstration of a real-life application to how it's used between different people. Every person has a role when it comes to

an elevator, such as a passenger, a person in waiting etc, so the system aims to represent how the elevator provides its service to everyone. The objectives of the elevator consisted of:

- A limited simulation developed in JavaScript.
- A simulation that is meant to create an elevator management system that can be adapted to real elevator systems.
- A structure consists of the UI with buttons, elevator movement, etc.
- A behavior is caused by inputs like buttons being pressed to cause elevator movement.

In reflection to these requirements, it was made possible using JavaScript, since we could implement classes and handy functions connecting to the back-end and front-end. For example, the inputs for each button worked in motion with JavaScript “onclick” functions, allowing us to diversify the actions by specifying the “onclick” parameters.

The system components represented how a real-life elevator would work, so roles of actors were presented so that a general understanding of the system could be perceived.

Table of Elevator Required Roles

Major Components	Major Responsibilities and Roles
Elevator	(EV) Process elevator request, process floor request, open/close doors, indicate a moving direction, and process the emergency alarm.
Elevator Interface	(EI) provides up/down, close/open, emergency alarm/stop, floor choice buttons

Elevator Admin	(EA) Keeps the system up to date and assures the system is fully functional and safe to use.

As a team, the elevator admin represented us as a team, through many prototypes and managing the project as is, in order to keep the elevator up to date. In addition to our requirements, we decided to cut the emergency alarm out of the picture, focusing more on the general functionality of elevator lift for travel. As well as amenities, such as radio, dynamic floor indicator, to focus on the main representation of an elevator.

Functional Requirements

The main requirements that were functional, which are used to interact with the user, consisted of the buttons and doors for achieving the human interaction of the elevator. Here is a list of functional requirements that were needed in order to attain a reasonable amount of interaction between users:

Inputs:

- Number elevators and floors.
- Up and down floor buttons.
- Emergency buttons.
- Open and close buttons.

Outputs:

- UI with floor and elevators.
- Elevator movement.
- Doors open/close.

Computations:

- Creating the UI based on floor number and elevator number.

- Computing which elevators to choose to send to the floor.

Timing/Sync:

- Elevator movement computation is done every time the button is pressed.
- Door movement is done every time the button is pressed on the elevator.

A table representing the actions of these requirements listed what should be done on the call of these buttons. We decided to cut computing the number of elevators and floors on user input since it would have been a nice addition to the simulation, but we believed that it did not serve a relevant purpose when it comes to the overall functionality of a real-life elevator

Table of Functional Requirements (FR):

ID	Description: FRs for EV
FR1	User processes floor choice
FR2	User requests going up
FR3	User requests going down
FR6	User processes door close
FR7	User processes door open

Table of Non Functional Requirements (NFR):

ID	Description
NFR1	The EV provides a weight limit
NFR2	The EV provides maximum passenger capacity indicator
NFR3	The EV provides a floor indicator

NFR4	The EV provides direction indicator
NFR5	The EV provides a clock
NFR6	The EV rings a bell on emergency calls
NFR7	The EV provides an intercom

Assumptions/Responsibilities

A list of assumptions were in the event of unforeseen circumstances that us programmers would not be able to control. Most of the conditions relate to the real-life application of an elevator, in the hope that it would not jeopardize the simulation of our elevator management system.

Table of Assumptions (A):

ID	Description
A01	The EV is always functional
A02	The EV cannot move in both directions at once
A03	Elevator is always going up, down, or is stationary
A04	User will always a choose a floor
A05	Door must always close before EV moves
A06	Door must always open when EV reaches destination
A07	EV will always stop on emergency request

A list of responsibilities were also laid out to indicate a specific event for the elevator to partake. These responsibilities include overall functionality that is provided by the elevator, and reflects how the code should also be laid out.

Table of Responsibilities (R):

ID	Description
	Responsibilities of the Elevator Operator (OP)
R01	OP must maintain consistent surveillance
R02	OP must assure elevator functionality.
R03	OP must assure elevator conditions are safe
R04	OP must promptly respond to emergency requests
	Responsibilities of the Elevator (EV)
R5	EV must provide the passenger a lift to their destination
R6	EV must complete its requests before direction change
R7	EV must halt at its position on emergency stop
R8	EV must open door upon request
R9	EV must close door upon request
R10	EV must hold door open until suitable to leave
R11	EV must remain closed while performing lift
	Responsibilities of the Elevator Admin (EA)
R13	EA must assure buttons are working

R14	EA must check surveillance system functionality

As admins, they should also take note that the responsibilities made for the elevator is majorly THE ADMINS' responsibility. To maintain the responsibility that the elevator must stay functional, we have cut down the responsibilities to the actual elevator and admin to process a broader understanding for customers' clarity for functionality. This leaves out the possibility of ambiguity and to focus rather on the main function of the elevator.

Use Cases

As part of the development of the EMS, use cases were very handy in outlining the steps whenever such an event occurred. The process required the team to think through many possible scenarios and decide how we could approach it using applied assumptions and responsibilities, which made it ultimately simpler for us to implement the simulation. Although, implementing the front end would be the more difficult part since it would mean a brand new environment to explore, thus extending our schedule from the original Gantt chart. Using use cases, it provided insight on alternative strategies that could make our software much more module-independent. Below is a table of some use cases that are similar to the use case diagrams, which go into more depth to the main functionalities.

Table of Use Cases (UC):

ID	Name	Description
UC01	Request upward lift	User requests for an elevator to perform an upwards lift, from the floor where the user is currently situated.
UC02	Request downward lift	User requests for an elevator to perform a downwards lift, from the

		floor where the user is currently situated.
UC03	One floor requested	System must process the floor choice that has been imputed by the user and apply it.
UC04	Multiple floors requested	Elevator processes two or more floors by prioritizing direction of first request
UC05	Process emergency stop	User requests for an elevator to perform a full stop, remaining stationary at its current position
UC06	Force open from inside	User requests for the elevator door to remain open from the inside.
UC07	Force close from inside	User holds button to force close door on command

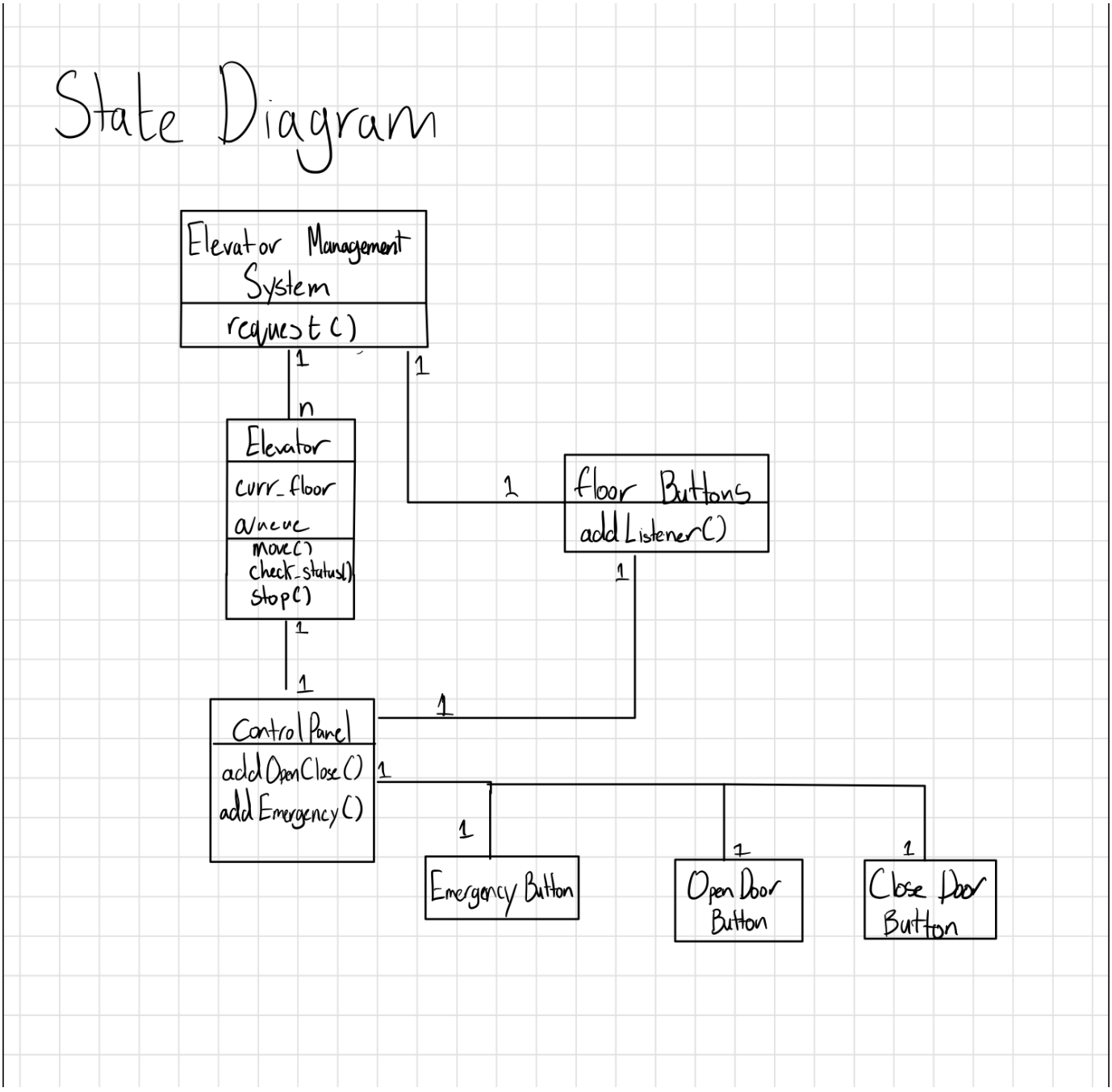
Here below is a snapshot of a use case scenario, taken from our Phase 2 Lab report. In depth diagrams and walkthroughs are provided in later sections of this document.

Example Use Case Scenario Table

STD01: Process down choice
Use Case Name: UC#01
Scope & Description: User requests for an elevator to perform an upwards lift, from the floor where the user is currently situated.
Primary Actors: Users
Pre-condition: Elevator is already in upwards motion, or elevator has no existing passengers
Triggering Event: Pressing button requesting the elevator to go up
Main Scenario Steps: (already defined for this Use Case) 1. (R6) EV will perform upwards lift when available 2. (R8) Open door for user 3. (R9) Close door when user has entered
Post-condition: Elevator proceeds to level of choice

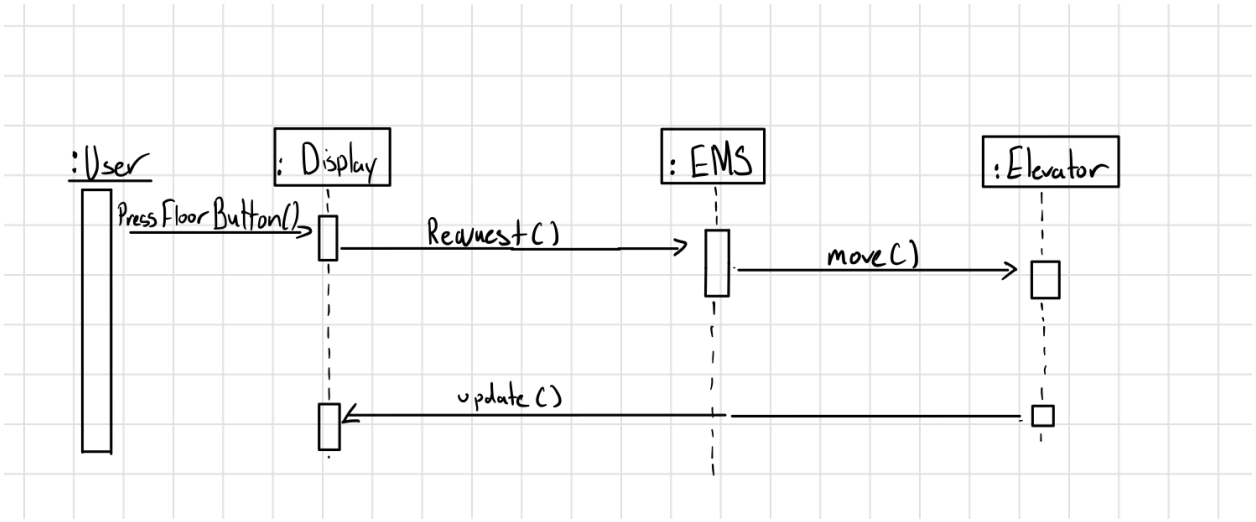
Extensions: Elevators will only proceed upwards until all destinations have been serviced. Elevators will not process lower floors choices if it is going up. Main scenario steps will cycle.
Non-functional Requirements: Direction indicator, floor indicator (incrementing representation since elevator is going upwards)
Frequency of Occurrence: Upon user request for the elevator

Class/State Diagram



Use/Test Cases

Requesting an elevator from the outside floor buttons



When the user presses the floor button the following EventListener in the FloorButtons class is activated:

```
class FloorButtons {

  constructor(doc) {

      // Get HTML for the floor buttons

      this.floorButtons = doc.getElementsByClassName("btn");

  }

  addEventListeners(elevNum=null) {

      // Add Event Listener for Floor Buttons

      Array.prototype.forEach.call (this.floorButtons, (button) => {

      button.addEventListener("click", () => {

      if (elevNum == null) {

      ems.request(button.innerText);
```

```
      } else {
            ems.request(button.innerText, elevNum);
      }
      })
      })
  }
}
```

The message request is sent to the Elevator Management System(EMS) class with an argument of floor number.

```
request(floor, elevN = null) {

      if (floor < 1 || floor > this.floors) {
      console.log(`Invalid floor. Please enter a number between 1 and
${this.floors}.`);
      }

      var flag = false;
      // Checks if this is a custom request
      if (elevN == null){
      // If not a custom request, check if an elevator is already on that
floor
      for (var i = 0; i < this.numElev; i++) {
      if (this.elevators[i].getFloor() == floor) {
```

```
            flag = true;
            break;
        }
        }
        }
        // Checks if already on floor
        if (flag) {
        console.log(`Elevator already on ${floor} or on the way`);
        } else {
        console.log(`You have requested to go to floor ${floor}.`);
        console.log(elevN);
        // Find closest elevator unless it's specified
        let closestElev;
        if (elevN == null) {
        closestElev = this.find_closest(floor);
        } else {
        closestElev = this.elevators[elevN];
        }
        closestElev.queue.push(floor);
        if (!closestElev.isMoving && !closestElev.isDoorOpen){
        closestElev.move(closestElev.queue.shift());
        }
        }
}
```

Firstly, it checks if the floor number requested is valid. Then if the request is not custom to a specific elevator it checks if there already is an elevator on that floor. If there is not an elevator already there EMS evokes an inner function to find the closest elevator.

```
find_closest(floor) {
      var closestElev = this.elevators[0];
      var currMin = Math.abs(closestElev.getFloor() - floor);
      var check = 0;
      for (var i = 1; i < this.numElev; i++) {
      check = Math.abs(this.elevators[i].getFloor() - floor);
      if (check < currMin) {
      closestElev = this.elevators[i];
      currMin = check;
      }
      }
      console.log(`Closest elev is ${closestElev.elevNum}`);
      return closestElev;
  }
```

Then it pushes the request to the queue of that elevator and only calls move on that elevator if and only if the elevator is not currently moving and the elevator's doors are closed. This is an important functionality which can be explained by looking at the move, open and close door methods in the Elevator class.

```
// move method, will simulate up or down using translate
  move(floor) {
```

```
      this.isMoving = true;
      this.movingTo = floor;
      console.log(`Elevator ${this.elevNum} moving from floor
${this.currentFloor} to floor ${floor}...`);
      setTimeout(() => {
      this.elevDoc.style.transform = `translateY(${(floor-1) * -100}px)`;
      this.isMoving = false;
      this.currentFloor = floor;
      console.log(`Elevator ${this.elevNum} has arrived at floor
${this.currentFloor}.`);
      this.openDoor();
      }, this.speed);

  }
  // change to width to simulate opening door
  openDoor() {
      this.isDoorOpen = true;
      console.log(`Elevator ${this.elevNum} door opening...`);
      setTimeout(() => {
      this.leftDoor.style.transform = `scale(0,1)`;
      this.rightDoor.style.transform = `scale(0,1)`;
      this.closeDoor();
      }, this.doorTime);
```

```
  }
// change to width to simulate closing of door
  closeDoor() {
      this.isDoorOpen = false;
      console.log(`Elevator ${this.elevNum} door closing...`);

      setTimeout(() => {
      this.leftDoor.style.transform = `scale(1,1)`;
      this.rightDoor.style.transform = `scale(1,1)`;

      if (this.queue.length > 0) {
      this.move(this.queue.shift());
      }
      }, this.doorTime);
  }
```

After the elevator moves to a floor by adding a transform to the style sheet of the elevator, it calls the open door method. After the door has been open for a set amount of time that method calls the close door method. Here after closing the door it checks if there is a request in the queue, if there is it calls move on that floor request. This assures that all floor requests are fulfilled back to back. This is also why move only needs to be called by the EMS when the elevator is not moving and the doors are closed.

Unit Test: Elevator Floor Buttons Request

Test Case 1

Instructions:

1. Press floor six button
2. Wait for elevator to reach floor six
3. Press floor five button
4. Press floor two button

Expected Results:

1. Either elevator moves to floor six since they are at the same distance. In this case elevator one.
2. When floor five button is pressed elevator one moves to it since it's the closest, already at floor six.
3. When floor two button is pressed elevator two moves to it since it's the closest, already at floor one.

Test Case 2

Instructions:

1. Press floor five button
2. Press floor four button
3. Press floor two button

Expected Results:

1. Either elevator moves to floor five since they are at the same distance. In this case elevator one.
2. Then elevator one will then move to floor four.
3. When floor two button is pressed elevator one moves to it since it's the closest, already at floor one.

Test Case 3:

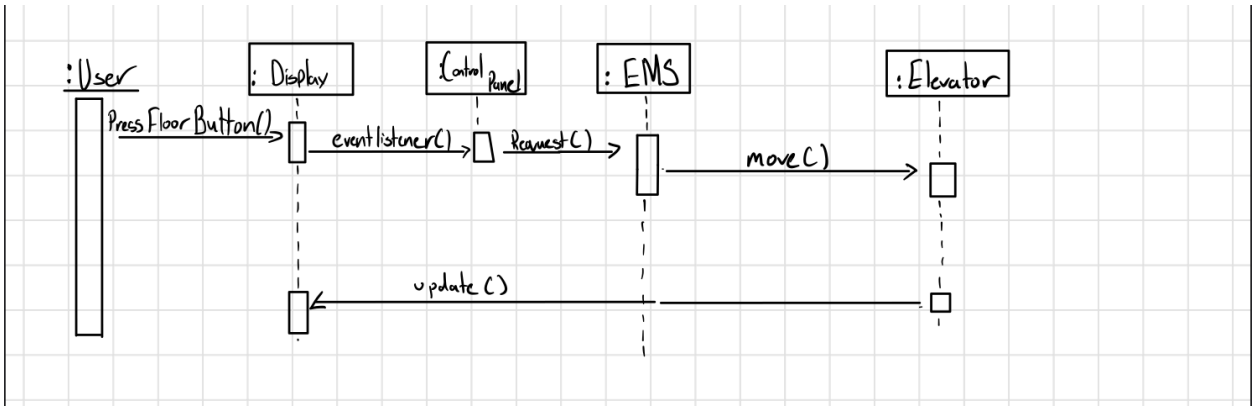
Instructions:

1. Press floor five button
2. Wait for elevator to move to floor five
3. Press floor two button
4. Wait for elevator to move to floor two
5. Press floor six button
6. Press floor one button

Expected Results:

1. Either elevator moves to floor five since they are at the same distance. In this case elevator one.
2. When floor two button is pressed elevator two moves to it since it's the closest, already at floor one.
3. When floor six button is pressed elevator one moves to it since it's the closest, already at floor five.
4. When floor one button is pressed elevator two moves to it since it's the closest, already at floor one.

Requesting an movement from the Control Panel of an Elevator



Here the only thing that changes is that request is sent to EMS with an optional argument indicating which elevator to move regardless if there is an elevator already on that floor or if another elevator is closer.

```
// Class for control panel of each elevator
class ControlPanel {
  constructor(elevNum) {
      this.controlPanel =
document.getElementsByClassName("control")[elevNum];
      this.floorButtons = new FloorButtons(this.controlPanel);
      this.floorButtons.addEventListeners(elevNum);
      this.emergencyButton =
this.controlPanel.getElementsByClassName("emergencyBtn")[0];
      // Add emergency button listener
      this.addEmergencyListener();
  }
```

Here in the constructor for the Control Panel class the Event Listener in the Floor Buttons class is given an optional elevator number argument, which can be seen being used

below.

```
addEventListeners(elevNum=null) {

      // Add Event Listener for Floor Buttons

      Array.prototype.forEach.call (this.floorButtons, (button) => {

      button.addEventListener("click", () => {

      if (elevNum == null) {

      ems.request(button.innerText);

      } else {

            ems.request(button.innerText, elevNum);

      }

      })

      })
```

Unit Test: Request Elevator from Control Panel

Test Case 1

Instructions:

1. Elevator one control panel button for floor 4 is pressed
2. Elevator two control panel button for floor 2 is pressed

Expected Results:

1. Elevator one will move to floor 4
2. Elevator two will move to floor 2

Test Case 2

Instructions:

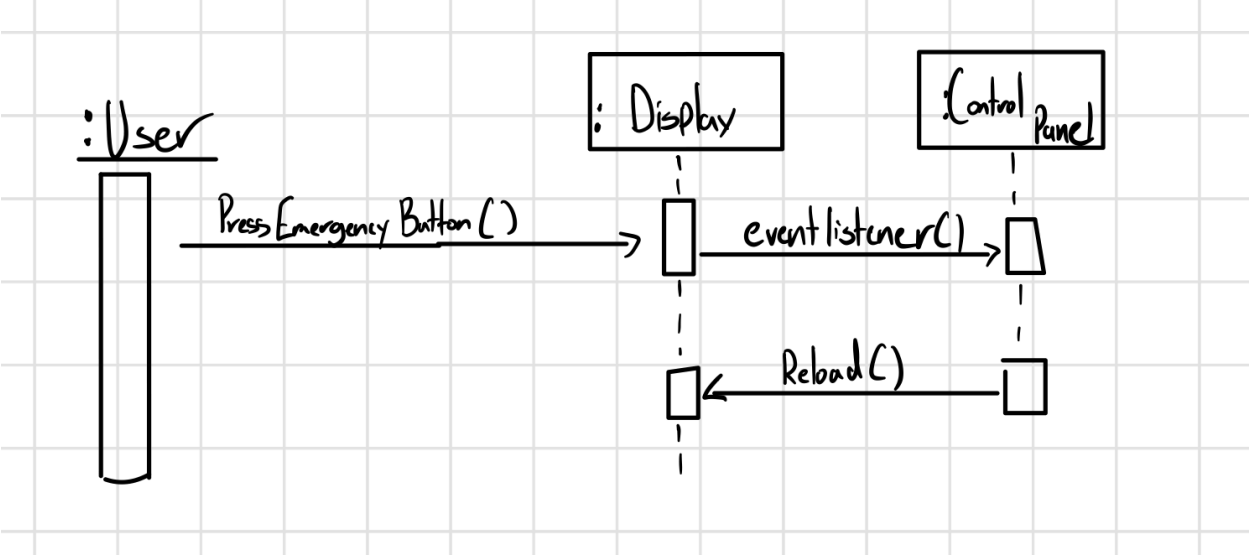
1. Elevator one control panel button for floor 4 is pressed
2. Elevator two control panel button for floor 4 is pressed

3. Elevator one control panel button for floor 2 is pressed
4. Elevator two control panel button for floor 3 is pressed

Expected Results:

1. Elevator one will move to floor 4
2. Elevator two will move to floor 4
3. Elevator one will move to floor 2
4. Elevator two will move to floor 3

Calling an emergency from the Elevator Control Panel



Here the event listener inside the Control Panel class is invoked and the page is reloaded.

```
// Class for control panel of each elevator

class ControlPanel {

...

    addEmergencyListener(){

      this.emergencyButton.addEventListener("click", () => {
```

```
      alert('Emergency button pressed. Security has been alerted.');
      location.reload();
      })
    }
```

Unit Test: Pressing Emergency Button from Control Panel

Test Case 1

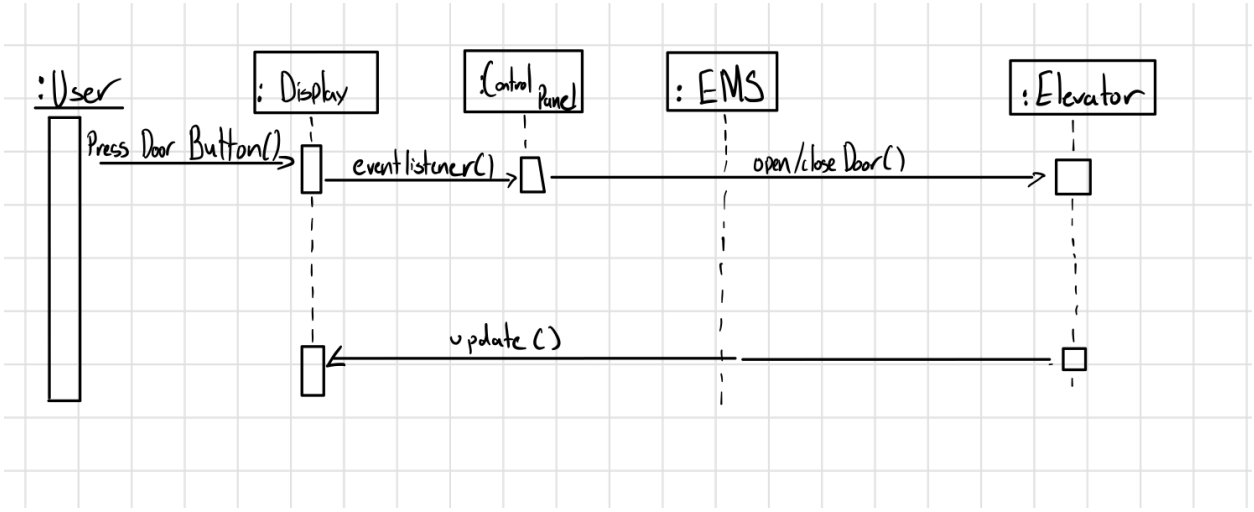
Instructions:

1. Press floor button four
2. Press floor button two
3. Emergency Button is pressed on elevator one

Expected Results:

1. Elevator one moves to floor four since both elevators are on the same floor.
2. Elevator two moves to floor two since it's the closest
3. Security is alerted
4. The simulation is reloaded and the elevators return to the same floor

Opening or Closing the door of an elevator through the Control Panel



Here, the open or close door method is called straight to the elevator. The open and close doors have already been shown above and are straightforward. They add a scale transform to the elevator door style sheet.

```
addEventOpenClose(elevNum) {
      // Add event listeners for open/close button
      let openButton =
this.controlPanel.getElementsByClassName("openBtn")[0];
      openButton.addEventListener("click", () => {
      ems.elevators[elevNum].openDoor();
      })
      let closeButton =
this.controlPanel.getElementsByClassName("closeBtn")[0];
      closeButton.addEventListener("click", () => {
      ems.elevators[elevNum].closeDoor();
      })}
```

Unit Test: Opening or Closing Elevator doors through Control Panel

Test Case 1:

Instructions:

1. Press floor button five
2. Wait for elevator one to arrive on floor five
3. Press floor button two
4. Wait for elevator two to arrive on floor two
5. Press close button on elevator one after one second
6. Press open button on elevator two after four seconds

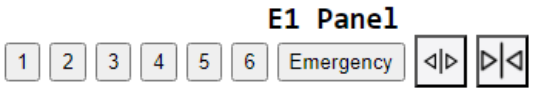
Expected Results:

1. Elevator one moves to floor five since both elevators are on the same floor
2. Elevator two moves to floor two since it’s the closest
3. Once elevator one arrives, after one second, doors will close.
4. Once elevator one arrives, after four seconds, doors will reopen.

User Manual

User Interface Controls

1. E1/E2 Panel



Usage:

- Interior of the first elevator (left)
- **Buttons** 1 - 6: floor choice, on click will send lift to corresponding floor
    - Clicking on the same floor button, while being on the floor, will open the door until another floor number is requested
    - For example, clicking “5” at floor 5 will open doors
    - Elevator will fully move to floor before any other floor number can be processed
- **Emergency Button:** click to display an alert that informs that the emergency services have been contacted
- **Open door Button:** click to open elevator gates
    - Clicking the open button while the elevator is moving will **not** open the doors
    - Will only open doors when elevator is **stationary/stopped**
- **Close door Button:** click to close elevator gates
    - Clicking the close button while doors are open, will **close** the doors

2. Floor numbers beside lift (1-6)

Usage:

- Each button is labeled once, depending on its floor level
- Click the button will request a lift to be sent to the floor
- Clicking the floor button while the lift is on that current floor, will keep the doors closed, until the open button is pressed

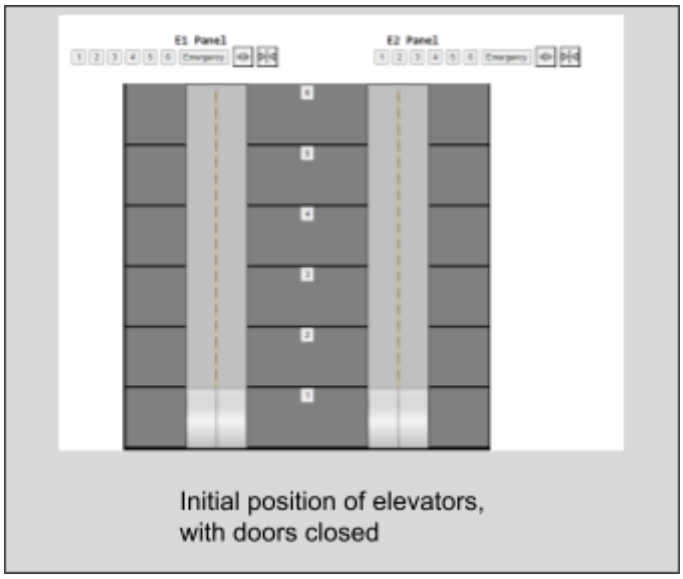
How to request an elevator

1. Elevators are defaulted to floor 1
2. From the elevator representation, click any floor number

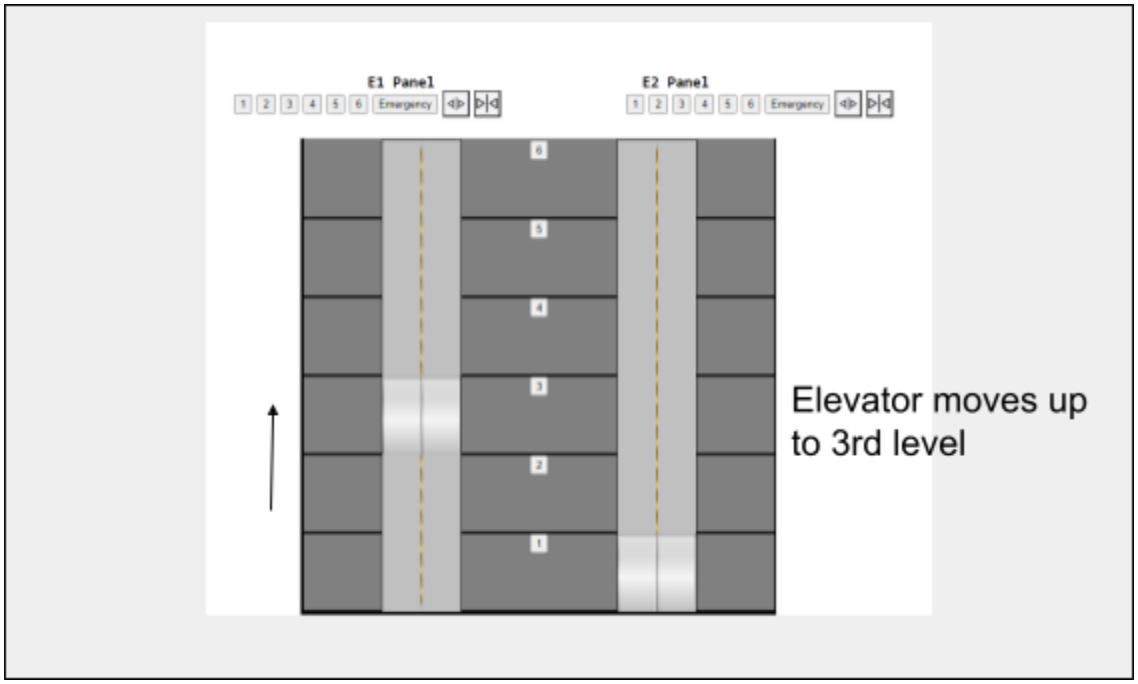
3. Elevator will go up or down depending on where the elevator is
    a. If a floor number that is clicked is **less** than the current floor, the elevator on the **closest** will go down to requested floor
    b. If a floor number that is clicked is **greater** than the current floor, the elevator on the **closest** will go up to requested floor
4. When elevators are equally distanced from a requested floor, the elevator on the **left** will respond to its request
5. Once initial floor number has been requested, the panels can be used
    a. Elevators must have moved **at least once** before using the panel
    - Ex. Left elevator moves up to 2, then E1 Panel can be used to manage passengers

Example scenario requesting a lift to “Floor 3”

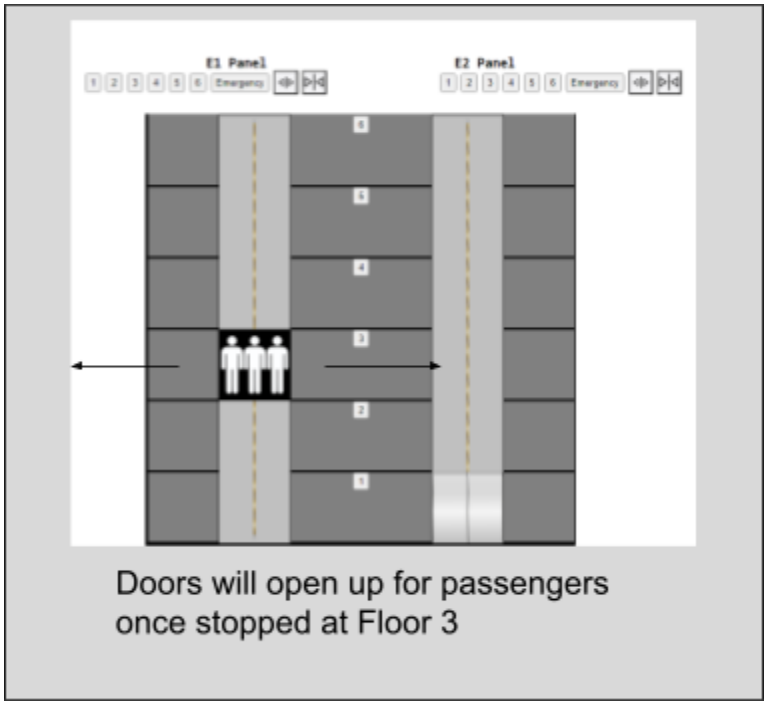
1. **Elevators are initially started at Floor 1, stationary until user requests for a floor**



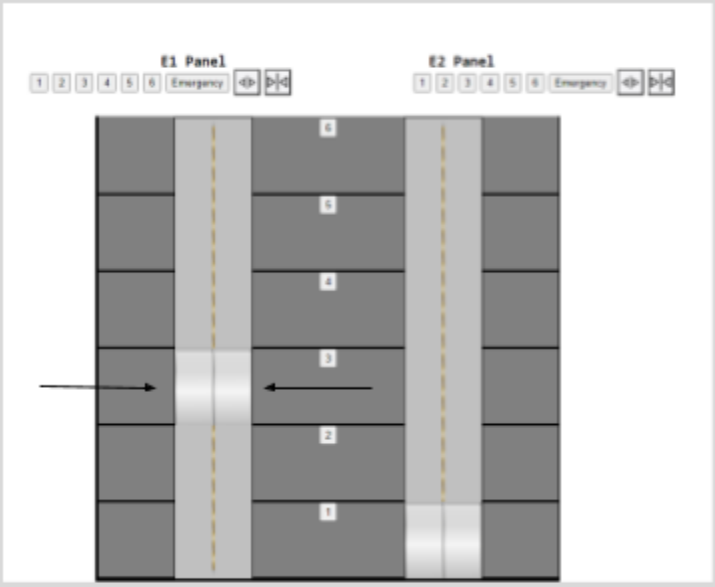
2. **When user clicks “3” on the elevator interface, the lift will move upwards beside the button “3”**



3. **Doors will open, showing the passengers of the lift, for anyone that wants to enter/exit the lift**



4. **Doors will close after all passengers have completed their trip, or want to proceed to another floor. Until another floor is requested, the lift will move accordingly.**



Doors close up, remains stopped until next call