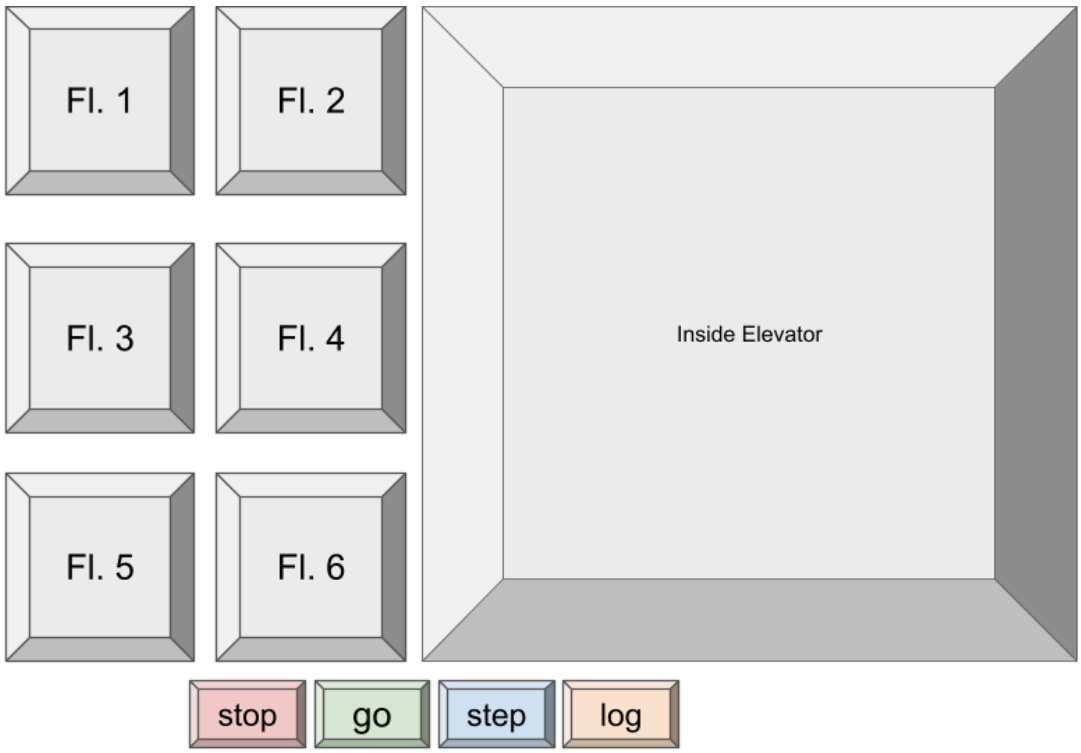
Elevator Simulation Design Document, v2

Team Name: Updog
Team Members: Arnav and Sarp

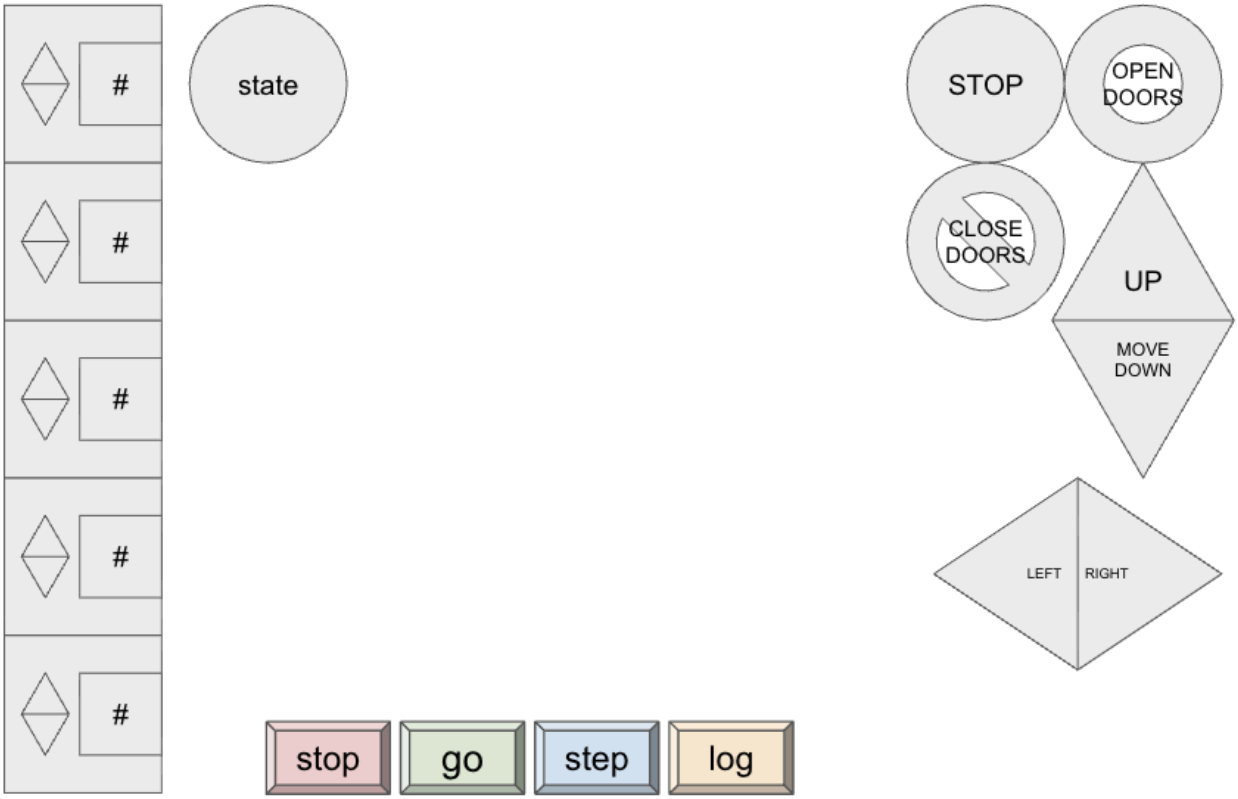
1. **GUI Mockup**

   This can be hand-drawn or designed in slides or drawIO - but paste a picture here. It should be clear how you will convey all of the required information and controls to the users.

Advanced GUI, if we have time:



Basic GUI, initial implementation



2. **Identify the ownership of each class:**

   Arnav
   Default Package: ElevatorSimController.java, ElevatorSimulation.java
   Building package: CallManager.java
   passengers package: Passengers.javalic

   Sarp
   Building package: Building.java, Elevator.java, Floor.java

3. **Identify (first pass) all of the externally visible methods (public and package level) *other than Getters/Setters* that you think you will need for each class.**

   ElevatorSimController:
   // Constructor instantiates a new instance of the Elevator Sim Controller
   public ElevatorSimController(ElevatorSimulation gui)
   // Method enables logging for the Elevator Sim Controller instance
   public void enableLogging()
   // Goes through each step of the simulation, incrementing the time during each pass
   public void stepSim() {
   // Prints out everyone currently in the queue

public void dumpPassQ()

ElevatorSimulation:
// Instantiates a new elevator simulation
public ElevatorSimulation()
~~// Starts the simulation~~
~~public void start(Stage primaryStage) throws Exception~~
~~// Allows the the user to change millisPerTick through the command line~~
~~public static void main (String[] args)~~

Building:
// Instantiates a new building.
public Building(int numFloors, String logfile)
// Initialize building logger.
void initializeBuildingLogger(String logfile)
// Update elevator - this is called AFTER time has been incremented.
public void updateElevator(int time)
// Process passenger data.
public void processPassengerData()
// Enable logging.
public void enableLogging()
// Close logs, and pause the timeline in the GUI.
public void closeLogs(int time)
// Add a new passenger group to the appropriate floor queue.
public void addPassengerToFloor(Passengers p, int time)

CallManager:
// Instantiates a new call manager.
public CallManager(Floor[] floors, int numFloors)
// Update call status.
void updateCallStatus()
// Prioritize passenger calls from STOP STATE
Passengers prioritizePassengerCalls(int floor)
// Returns true if there are any pending calls (up or down) in the building.
boolean hasPendingCalls()
// Check whether there are up calls above the given floor (strictly above).
boolean hasUpCallsAbove(int floor)
// Check whether there are down calls below the given floor (strictly below).
boolean hasDownCallsBelow(int floor)
// Is there a call on the specified floor in the given direction (UP/DOWN)
boolean isCallAtFloorDirection(int floor, int dir)

Elevator:
// Instantiates a new elevator.
public Elevator(int numFloors, int capacity, int floorTicks, int doorTicks, int passPerTick)
// Update curr state.

void updateCurrState(ElStates nextState)
// Update floor
public void updateFloor()
// Update door
public void updateDoor()
// Determines if there are passengers to board at the given floor
public boolean passengersToBoard(Floor floor)
// Returns true if there are passengers to get off at the given floor
public boolean passengersToGetOff()
// Returns true if the elevator is physically at a floor
public boolean isAtFloor()
// Increases the delay
public void increaseDelay(boolean ifBoarding)
// Returns true if the delay is up
public boolean delayDone()
// boards the passengers on a floor
public Passengers board(Floor floor)

Floor:
// Instantiates a new Floor, with a specified up and down queue length
public Floor(int qSize)
// Allows visibility into the queue, represented as a string, with the exact visualization up to us
String queueString(int dir)
// add a passenger to a specified queue; return true if successful
public boolean addPassenger(int dir, Passenger p)
// remove a passenger from the specified queue, returning the Passenger object if successful
// null if unsuccessful
public Passenger removePassenger(int dir)
// peek at the top passenger, same as remove, but without the actual removing part:
// return Passenger object if successful.
public Passengers peekPassenger(int dir)

Passenger:
// Instantiates a new passenger
public Passengers(int time, int numPass, int on, int dest, boolean polite, int waitTime)
// Gets a formatted string for the passenger instance
public String toString()

State:
// return the currState
public ElStates getCurrState()
// return the currFloor
public int getCurrFloor()
// return the numPassengers
public int getNumPassengers()
// Constructor for State
State(ElStates currState, int currFloor, int numPassengers)

4. **Project Management, Work Flow, Conflict Resolution and Milestones**

For the first pass of this document, you should answer the following question:
- how will you manage/communicate turning in new code to your shared repository,
*- what actions will you take should there be a problem **before** coming to talk with me?*

You should also have a placeholder for a schedule of milestones - this will be completed separately, but should be pasted in when done. Using Google Sheets, or a table, is a good way of doing this.

We will ensure communication between Sarp and I, in and out of school, and express ideas openly. ***I will email, rather than text Sarp for communications regarding the elevator project, to ensure he is able to see my messages. I can also use Google Chat if I have an urgent message, as I know he uses it more often. However, I will avoid calling/texting him as limitations on screen time hinder communication that way.*** *If we have issues or conflicts of interest, we will communicate our thoughts to each other respectfully and try to combine our different ideas to contribute effectively to the project. We will always go over our goals and what we plan to accomplish every class and whenever we work on the project together so that we can both have a grasp on what the other is doing and how the project is going. If a code-related problem arises, we will attempt to debug it together; however, we will realize when we are stuck and need help. If a disagreement arises, we will also attempt to resolve this between ourselves, but will talk to Mr. Murray if we need help resolving it.*

Green = finished

Red = unfinished tasks to complete

Milestone	Owner	Date
Passenger class completed	Arnav	Dec 4
Basic static GUI Layout	Arnav	Dec 5
Floor class completed	Arnav	Dec 5
Finishing building class	Sarp	Dec 8
basic Stepsim, incrementing time, etc.	Arnav	Dec 8
First finished draft	Both	Dec 11
FSM correctly transitions	Sarp	Dec 11

Correct state transitions and elevator logic	Both	Dec 16
Passing all tests 100%	Sarp	Jan 6