Practical: Repair the Elevator

Student for a Day Computing Science

Assignment 1: Improve the elevator algorithm

In your daily life, you encounter Computing Science in many unexpected places. An example of this is an elevator: it may seem simple and straightforward, but moving people up and down efficiently takes some more effort than you would expect. During this practical, you will program a simulation of an elevator, with the goal to transport as many people as possible during a short period.

To start the practical, log in to your PC and open a browser.

- Visit `https://elevator.fabianaukema.nl/`
- This page contains an editor and the elevator simulator. Play around with the existing code to get used to the environment.
- Adapt the code so that it will pass challenge 1.

As may have seen, the first challenge fails because the elevator never visits the 2nd floor. To fix this add `elevator.goToFloor(2);` to the code to make the elevator visit the 2nd floor as well.

Why does this work? The object `elevator` represents an elevator. An elevator has a function which you can call to make the elevator visit a specified floor: `goToFloor(<floornumber>)`. With the line `elevator.goToFloor(2);` you instruct the elevator to visit the 2nd floor.

- This implementation is rather inefficient and impractical. Why is that?
- Try passing some more challenges.

By instructing the elevator to visit the 3rd and 4th floor as well, you will pass a few extra challenges. Eventually, you will get stuck at challenge 4, as it does not work with two elevators. Even if you instruct the second elevator to move between levels 0 and 7, it will still fail the 4th challenge. To improve this we have made a solution that passes more challenges.

- Go to `https://elevator.fabianaukema.nl/initialSolution.js`
- Copy this code to the elevator simulator and try how many challenges you will pass.

You will probably be able to get to challenge 8, but for some challenges, it may take a few attempts. To get further, we have to make a few changes to the program.

The program has two lists (`floorsWithPassengersDown` and `floorsWithPassengersUp`) that are used to register the floors where people are waiting. When the elevator is not moving people around, it uses these two lists to determine where to go to. Right now, pushing a button to let the elevator know you're waiting on a floor has no effect. Our task now is to make sure the program registers on which floors the request buttons are pressed.

First, we have to look up whether the up or down button is pressed on each floor and register this in the right list. To do this we have to iterate through all floors and look if one of the buttons has been pressed. You can do this by adding a *while*-loop at line `20`. This is an example of a *while*-loop:

```
var f = 0;
while (f < floors.length) {
    //code that has to be executed
    f++;
}
```

- Implement the *while*-loop.

This code will iterate the value of i from 0 to the number of floors, defined by `floors.length`. You have to check for every floor whether the buttons are pressed, but you need to access the floor object itself to do so.

A floor f can be accessed by using `floors[f]`, which will give you the `f`th floor. Whenever a button is pressed, it triggers an event, which you can catch by using the `on`-function. In this code, `function()` is the function that is executed whenever the event is triggered.

```
floors[f].on('event', function(){/* some code */})
```

- Take a look at the documentation page to find out which events you need to catch to register the button presses on floors.
- The event's floor level is stored in the variable `this.level`
- Implement the *while*-loop with the functions to add the floor number where a certain button is pressed to the right list.

In JavaScript you can use the function `.push(item)` to add an item to a list.

Assignment 2: Sorting optimisation

You can make the code even more efficient by sorting the `floorsWithPassengers`-lists. To do that you can use a sorting algorithm called bubblesort, which is an $\mathcal{O}(N^2)$-algorithm.

- Implement the bubblesort algorithm. Find a place in the code to add your implementation.
- Go to `https://elevator.fabianaukema.nl/solutions/bubblesortaddition.js`, here you will find the code on the next page.
- Find a place in the elevator code to add the code.

```
floorsWithPassengers = bubbleSort(floorsWithPassengers);
var floorNumber;
if (floorsWithPassengers[0] >= this.currentFloor()) {
    floorNumber = floorsWithPassengers[0];
} else {
    //get closest floor
    var z = 0;
    while (z < floorsWithPassengers.length) {
        if (floorsWithPassengers[z] >= this.currentFloor()) {
            break;
        }
        z++;
    }
    if (floorsWithPassengers.length-z > z){
        floorNumber = floorsWithPassengers[z];
    } else {
        floorNumber = floorsWithPassengers[z-1];
    }
}
removeFloorFromPassengersOnFloor(floorNumber);
this.goToFloor(floorNumber);
```

Assignment 3: make your own solution

Try to come up with an other optimisation to make your elevator even more efficient.