CS166 Assignment 1 - Elevator Simulation

Dennis Kageni, Ziquan Fu, Luis Gonzalez

Spring 2020

Code: https://colab.research.google.com/drive/1AeHVHzR3daTwkM_ets4xhmGMK27iE5xD

For the following assignment, the team implemented a model simulating an elevator in Python. Using Object-Oriented Programming (OOP) techniques, we simulate two operating strategies on a population and conclude by comparing the performance of each strategy that we implemented.

Strategies:

1. The Naive Strategy

This strategy was similar to the one suggested in the assignment instruction. The elevator is initialized randomly to a floor. It then moves in one direction where, on each floor, it makes a decision as to whether or not to open the door depending on whether there's a passenger waiting. Given the elevator's capacity when the door opens, the passenger then decides to get into the elevator only if it is not at full capacity. If no other passenger needs to leave the elevator or if the elevator is full, the elevator moves to the next floor. This happens until the elevator gets to the last floor in that direction. Afterwards, the elevator changes direction and the process repeats until it gets to the last floor in that direction. The simulation ends when all passengers have been transported to their respective floors

2. Improved Strategy

This strategy is modeled after the elevators in our residence hall. Once again, we initialize the elevator onto a random floor. The passenger still determines whether to get into the availability given how many people are already in the elevator. The elevator runs in one direction and only stops to pick up passengers who are going in the same direction. When all the passengers have been dropped off and there are no remaining requests in that direction, only then can the elevator respond to requests in the other direction. Otherwise, it should stop and wait for a call.

Efficiency

The metrics we used to compare the performance of each strategy were :

- The time taken to complete the simulation
- The waiting time before the elevator picked up a waiting passenger

Simulation, Results, and Analysis

For our model, we made the following assumptions about passenger behavior:

1. Passengers only use the elevator once for the duration of the simulation
2. The passengers do not make any mistakes; they always correctly communicating their floor choice to the elevator and they always get on or get off their respective floors

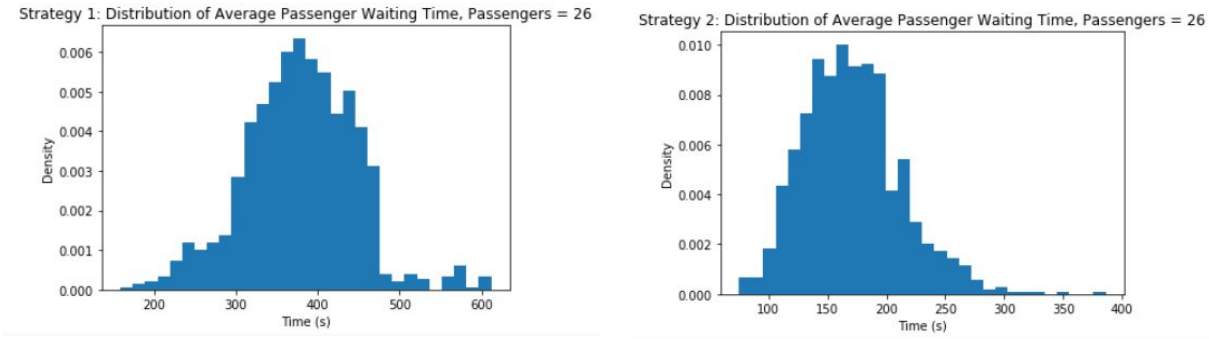

Figure 1. Histogram comparing the average waiting time between Strategy 1 and 2 over 1000 simulations

As expected, the improved strategy performed better in almost all the scenarios. In Figure 1, we can see that the average waiting time for the naive strategy reduces drastically when we reduce the number of passengers. Not only are the tails narrower - where we can infer that the variability of waiting time reduces with fewer passengers - but also the mean shifts to the left. Therefore, Strategy 1 would be better than Strategy 2 for smaller buildings that have fewer people using the elevator at a time. Conversely, Strategy 2 is preferred for more people. In Figure 2, observe how with 21 passengers, Strategy 2 outperforms strategy 1 for both average passenger waiting time and total simulation time

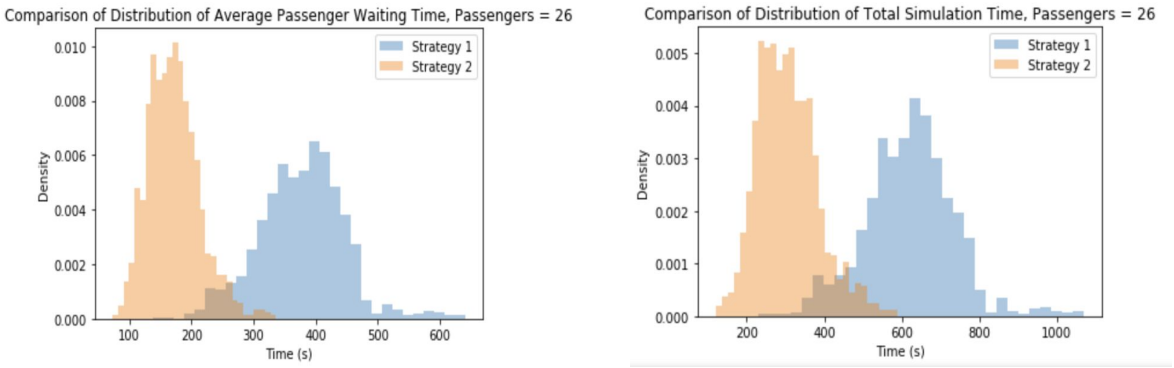

Figure 2. Strategy 2 outperforms strategy 1 for both average passenger waiting time and total simulation time

Contributions:

1. Annotated code for Strategy 2
2. Wrote the decide ( ) method for Strategy 2

Insights:

This assignment motivated the utility of Classes to group object-oriented constructs. More importantly, the assignment also helped understand the importance of sufficiently designed classes - especially those that interact with other classes - and well-annotated code. Since we were all writing the code, a lot of time was spent debugging where the main challenge came from forgetting to update the names of our methods after changing them or forgetting the value we used when initializing classes. Lastly, I realized the importance of beginning with simple simulations that work and using that as the foundation for building more complex ones. In particular, modeling simulations after something you understand (e.g) the elevator in the res hall helps a great deal too.