CS166 Assignment 1: Elevator Simulation

Yoav Rabinovich, Jan 2019
Worked with: Evgeny Yurtaev, Vy Tran & Nikesh Shrestha

For this assignment, our team implemented a model of an elevator in Python, considering different operational strategies by simulating them on different populations of passengers.

Strategies and Populations
We implemented two strategies for the operation of the elevator. The Naïve Strategy had the elevator operating much like it would on a Saturday in Israel: stopping at every floor independently of the passengers' needs. The Smart Strategy models standard elevator behavior: It travels to the nearest requested floor, prioritizing those in the current direction of travel.

We have also implemented a feature that allows us to choose between modelling a residential or a commercial building. This effectively represents a different generator function for our population of passengers: The former is highly skewed in favor of the ground floor for origin and destination point, while the latter has a more uniform distribution that allows for more variety in travel paths. By this we meant to display the dependence of the efficiency of each strategy on the population parameters.

Efficiency
We collected three different metrics for analysis of efficiency: Waiting Time refers to the average time each passenger spent waiting for the elevator; Movement Time refers to the average time each passenger spent in the elevator; and Journey Time refers to the combination of both. This allows us to analyze the impact of our strategy choice for each of these metrics, and optimize for whatever weighting of them we should find appropriate for a certain application.

Simulation, Results and Analysis
In this model, we made certain assumptions about passenger behavior: Passengers aren't consistent entities, in essence travelling only once for the duration of the simulation. They also don't make mistakes in operation: they always enter the elevator if the doors open on their origin floor while the direction matches their destination and the capacity allows, and they always get off as planned. Importantly, we also included parameters for the length of time it takes the elevator to move one floor, and the amount of time it takes a passenger to enter or leave the elevator, so that the time an elevator spends on a floor is proportional to the amount of people loaded or unloaded, without a minimum duration as some elevators have in reality.

We tested both strategies many times for both building types. Overall, the smart strategy overperformed at every test. As expected, in a residential building waiting time was improved considerably more than movement time with the smart strategy, since a disproportionate amount of people were waiting at the ground floor at any given moment, while the elevator visited it with no greater frequency than other floors. The movement time was also improved since it's effective to move straight to destination floors, but since opening doors at empty floors was instantaneous in our model, this didn't scale horribly (this is a target for improvement in further iterations of the model). In simulations of the mall building type, the smart strategy proved more efficient, but much less so: since passengers are likely to arrive at different floors, and since malls typically have less floors in total for the naïve strategy to waste it's time.

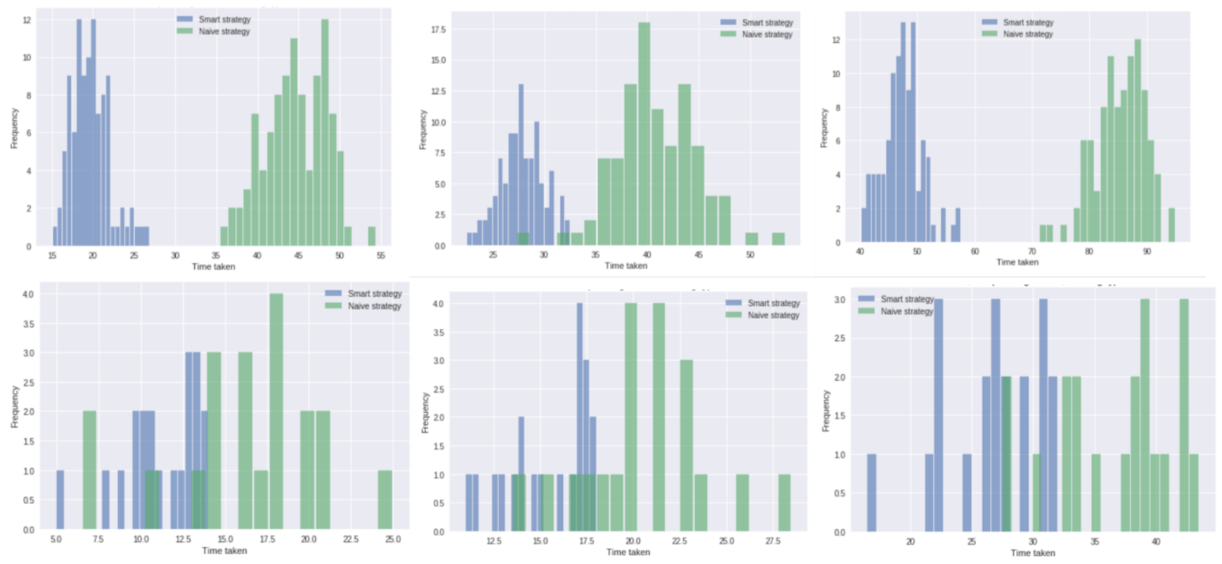

Figure 1. Histograms of results from simulations. Blue for smart strategy, green for naïve. Columns from left: waiting time, movement time, journey time. Top row: 100 simulations of a residential building with 11 floors and 50 passengers. Bottom row: 20 simulations of a mall with 6 floors and 10 passengers.

Contributions

I was designated as a code supervisor for this assignment, so I usually worked alongside my teammates supervising their work or reviewed it. I was the major contributor for the elevator class and for the time-keeping mechanisms, contributed to implementing the strategies, and most of my time went towards debugging throughout the process.

Python Simulation Insights

While object-oriented programming isn't new to me, I'm not used to coding with a group. This project made evident the importance of clear formatting and commenting, as reviewing code written by students less versed in conventions was very difficult and clarifying my code to them was also critical. I also made note of the cyclical modeling process our project has undergone, growing organically from our workplan: when encountering bugs, or strange results, we discussed and implemented features to more accurately or robustly represent the studied system. For example, we ended up modeling the loading process as the opening of doors since we noticed our elevator was unnaturally efficient when loading passengers by choice of convenient destinations, while in reality passengers decide when to get on based only on direction information. This was also my first time including a verbose argument, which proved critical to optimize debugging capabilities and readability for simulations.

Link to Code: *https://github.com/Shesh6/CS166--Modeling-Simulation-and-Decision-Making/blob/master/Assignment%201%20-%20Elevator%20Simulation.ipynb*