**Elevator Simulation Report**
**CS166, Spring 2020**
**Berfin Karaman**

**Group members:** Michael Chen, Khoi Pham, Zineb Salim

The main aim of this assignment is to implement an elevator simulation by using python. In this report, I will evaluate the efficiency of the different strategies that we chose to implement as a group. Each passenger is an individual agent with an aim to travel to a different floor. Their interaction with the building and elevator creates the path that the elevator follows which is an emergent property that we try to model in this assignment.

## Strategies

---

The first strategy that we chose to implement was the simple example strategy that was given in the assignment description. In this strategy, the elevator starts from the bottom floor of the building. Then it starts going up by stopping at each floor until it reaches the top floor. When the elevator reaches the top floor, it starts going down by stopping at each floor. Meanwhile, if there is a passenger that is waiting for the elevator, it picks up the passenger if there is space in the elevator. At the same time, if the current floor and the destination floor of a passenger that is in the elevator are the same, the passenger gets off on the current floor. In this case, the passenger reaches its end destination and exits the elevator simulation. For this strategy, we assumed that:

- There is a set capacity for the elevator and the passengers cannot violate this capacity.
- All passengers have one destination. Once they reach their destination floor, they leave the simulation.
- Besides all the passengers are initiated at the beginning of the simulation, there will not be any new passengers for the elevator.

The second strategy that we come up with as an improvement for the simple strategy is named as elevatordemand(). The main difference between the strategies is the way the next floor of the elevator is defined. In this algorithm, the elevator compiles a list of all the requests from the passengers in the elevator and the passengers that are waiting for the elevator. Then the next floor of the elevator is defined based on the closest requested floor to the current position of the elevator. After it reaches the next floor, the passengers exit and enters to the elevator. Then the list of requested floors is updated based on past activity. This process continues until each passenger arrives at their destination. The same assumptions hold. This algorithm is more

realistic and efficient since it decides the next floor based on request and does not stop on the floor if there is no request from or to that floor.

## Evaluation of efficiency

As a group, we decided to have two metrics to evaluate the performances of the algorithms. The first one is to calculate the total time it takes the elevator to deliver each passenger to their destination floor. Secondly, we calculate the average passenger waiting time. Average passenger waiting time is important because if you are designing a product, consumer satisfaction is important and one way to improve consumer satisfaction in this scenario is to minimize the passenger waiting time. We added the time that passes for the elevator to travel between floors, the time that it takes for elevator doors to open and close and lastly the time it takes the passengers to get in and out together to calculate the total time. To make our algorithm more realistic, we randomized the time that it takes for a passenger to enter or exit to the elevator. For each passenger, we initiated the starting time and based on that we calculated the waiting time for each passenger to find the average waiting time.

After we decided on the metrics that we wanted to evaluate, we run the simulation for multiple buildings and multiple passenger sets for both of the algorithms. The main reason why we wanted to simulate with different settings is that we wanted to see if our algorithms outperform the simple one indifferent scenarios. Also, multiple scenarios increase the accuracy of our conclusion regarding the efficiency of the algorithm over another.

## Results and evaluation

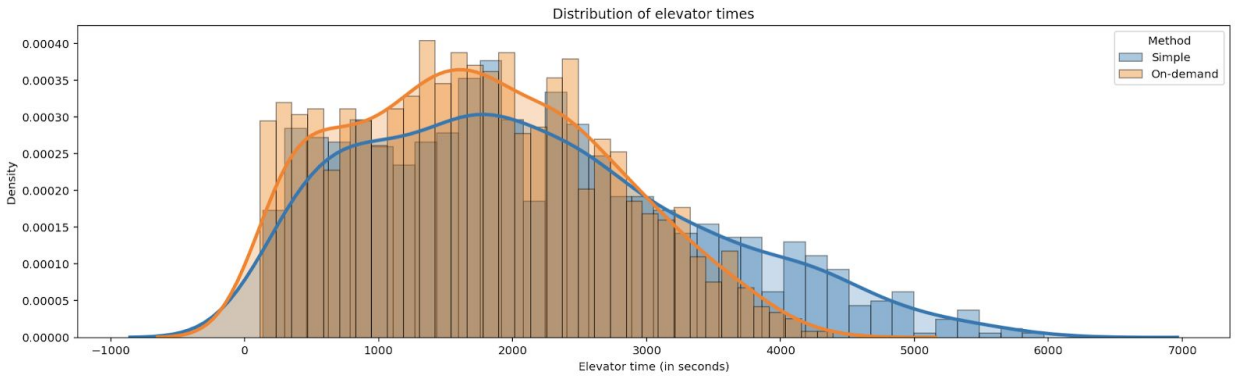


*Figure 1.* Distribution of the total time that it takes for the elevator to complete a simulation. In total there are 1000 simulations for each algorithm. Blue shows the result for the simple method while the orange shows the results for the on-demand algorithm.

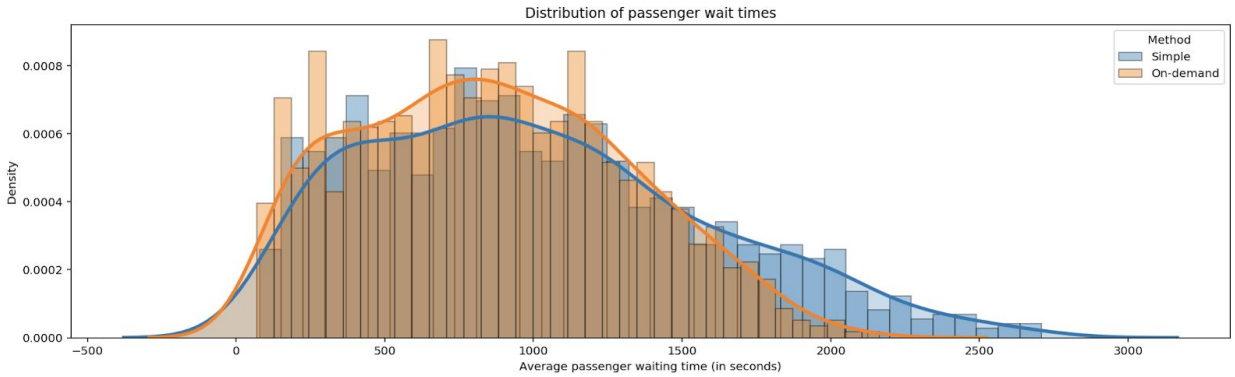


*Figure 2.* Distribution of the average passenger waiting time per simulation. In total there are 1000 simulations for each algorithm. Blue shows the result for the simple method while the orange shows the results for the on-demand algorithm.

As we can see from figure 1, on average there is a bigger possibility that the on-demand algorithm will perform better than the simple algorithm because its distribution is more skewed towards left compared to the simple algorithm. This means that generally, the on-demand algorithm runs the simulation faster than the simple algorithm. In figure 2, we evaluated the second metric. In figure 2, we can see that the on-demand algorithm again outperforms the simple algorithm when we compare the average passenger waiting time. Even though, there is a bigger possibility that the on-demand algorithm is more efficient, the difference between them is not drastic due to the overlaps of the distribution. The on-demand algorithm can be improved more to increase the efficiency of these metrics.

## Contribution

---

We work collaboratively for the passenger and elevator class during the class. Then we met the distribute and work together after class. During this time we work majorly on the building class. I personally worked on adding and removing passengers from the elevator and updating the passenger list. Throughout the project, we helped each other when one is stuck at a certain stage. Finally, I went through the code to comment and make it more readable.

## Main takeaways

---

As we were working through this assignment, I realized how we can use Python to simulate relatively simple emergent properties. This will enable us to analyze complex systems from another perspective. It shows us how we can make the system more efficient by chancing

some components of it. Additionally, I realized how using class structures in Python helps us to cooperate on the same code in an easier manner. It makes it easier to work on different methods and classes individually.