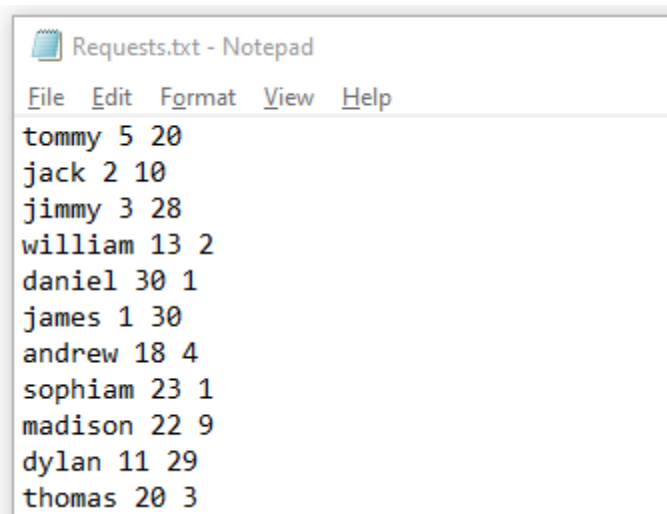


In this report I will introduce you to know the detail about my projects in two parts. The first part is related to the backend program writing in C++ and the second part is related to the frontend interface program writing in C#.

Let's talk about the backend program first. The actual works in the backend program is to simulate the elevator movement and create the data. One of the most important session in this part is how does the elevator move and how does it know when to move and how does it know which floor it should go to. In short, when, where, how.

**WHEN.** In my program, there are three threads present the elevators and one center-control system thread. For the center-control thread, it loads the all requests first in the Requests.txt file customized by the user, which is stored under the project folder. As it shown below.



```
Requests.txt - Notepad
File Edit Format View Help
tommy 5 20
jack 2 10
jimmy 3 28
william 13 2
daniel 30 1
james 1 30
andrew 18 4
sophiam 23 1
madison 22 9
dylan 11 29
thomas 20 3
```

For each line in the file, it represents a request. For each word in the line, the first represents the people's name, and the second represents the source floor and the third represents the destination floor. The center-control system loads all requests by one time and stored inside member variable. Inside the center-control system, it will run all the time until all the request is being dispatched. By calling the member function **strategieChooseElevator()**, the center-control system will access to each elevator object and get the elevator's status, current floor and etc. In this function, by comparing each elevator's status (UP, DOWN, STOP) and the distance between the request made floor with the current elevator floor, the center-control system will dispatch the request to a suitable elevator. The rules for elevator to take request obey the following rules:

1. **Stop State Priority.** If two elevators are in STOP state and one elevator is moving even though it is very close to the request made floor, the center-control system will dispatch the request to the elevator which now is in STOP state. For example, elevator 1 and 2 now are in STOP state, and elevator now is moving up from floor 8th. The request made at floor 10th and go to floor 20th. The center-control will dispatch the request to the elevator between elevator 1 and 2 randomly. All in all, elevator in STOP state pick request first.

2. **Principle Of Same Direction.** If the direction of the request made floor to the request destination is same the elevator status, the elevator will create the rank value to compare with other two elevator. The elevator owns the lowest rank value will pick up the request. For example, three

elevators now are moving up, and the request is made at floor 5, and its destination is 10. The center-control system compares their rank value and decide which elevator should pick request. All in all, if the elevator is not in STOP state, it will never pick up the request in the opposite direction.

**3. Smallest Rank Value First.** The rank value will create only when the elevator meets the requirement of the second rule. When it is being created, it means that the center-control system needs to compare which elevator is much close to the request made floor. For example, elevator 1 is moving up from floor 5th now. And the elevator 2 is moving up from floor 7th now. The request made at floor 10th and go to 20th. The rank value of the elevator 2 is  $10-7=3$ . The rank value of the elevator 1 is  $10-5=5$ . The rank value of the elevator 2 is much closer to the request made floor, the center-control system will dispatch the request to the elevator2.

By following the rules, the center-control system dispatches the requests until the request pool is empty.

**WHERE.** After obtaining the request, the elevator will move forward to the request destination until the request finish. By the rules, we know, the elevator only picks the request when its state is same as the direction of the request made floor to the request destination floor or in STOP state. If the elevator picks in STOP state, we just need to set the elevator state when it arrived at the request made floor. If the elevator picks request in same direction, the elevator state just keep state.

**HOW.** The elevator thread is running even though the center-control system is dispatching the requests. By calling the elevator member function **move()**, the elevator moves. Inside this function, it decides the elevator's next floor, if the elevator state is UP, the next floor is  $\text{currentfloor}++$ . if the state is DOWN, the next floor is  $\text{currentfloor}--$ . Inside this function, it will check the current floor with the request made floor, if the current floor is equals to the request made floor, the request will be deleted in the requestlist, and it will also be added to the arrivelist. Both vectors are the member variables. For each movement, the elevator will also check the current floor with the request destination floor. If the current floor equals to the request destination floor, the request will be deleted in the arrivelist, that means the request is finished. After finishing all the requests, the elevator will keep in STOP state until a new request is dispatched to it.

That's all the detail about the backend program.

Now let's talk about the frontend interface. The main idea of this interface is the lines and the colors. By using C# Windows Presentation Foundation, some lines and colors are created to presents the border, elevators, the request made floor, and the request destination floor. The frontend interface is ugly though, it presents the elevator movement.



The elevator 1 picks the request, and now moves to the blue block, which means the request destination floor. The elevator 1 is now moving to the request destination floor to finish the request. The other two elevators are also busy on its way to pick up the requests.

Conclusion. The elevator backend program works well as I expect. But sometimes, the suspicious fault occurs when the center-control access to the elevator's member variable, and then the elevators will go out of border.

My first ideal is using C++ CLI to connect the C++ backend program with the C# frontend interface however the Microsoft doesn't support the compiling the CLI when the backend program contains thread. The real-time of the whole program is not able to achieve. Then I choose to use the file system to connect the backend program and the frontend program. The frontend interface program doesn't work as I expect. Because the time points created in the C++ backend program is not accurate. When it is applied in the C# frontend program, sometimes the request made floor and destination floor show up earlier or later. All in all, there is still a frontend interface.