

# Elevator Simulation Report

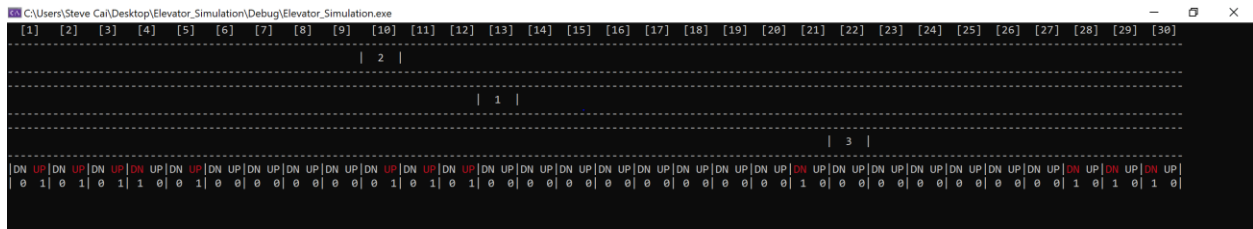
## [6. ReadMe file]

This is a elevator-simulation project that focus on realness in terms of the most common elevators we see in our daily life.

### How to run:

----Compile the source file on Windows (with window.h library )and run.----

For rendering, the project uses window.h console facilities. The elevators are drawn horizontally because we can have more spaces for each floor and thus to be more expressive. For each floor, 'DN' and 'UP' indicate the request buttons and the color indicates whether it is pressed or not. The number below indicates the number of people in line for that direction.



### Whole View:

The project includes three parts:

People-feed: respond to generate wait people for each floor at certain time.

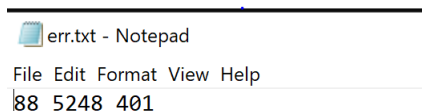
People-In-Line: floor manager thread at each floor get wait people vector, put them in line and generate up/down request.

People-consume: each cart thread goes to certain floor based on the current passengers' target floor and floor request and load/unload people if applicable.

----people feed data sequence can be tracked in datafeed.txt, if you want to repeat the process with same people feed data, comment the first feedData() in main function. Or if you change some relevant parameters below, uncomment that line to generate new people feed data.----

----Final statistical data is recorded in err.txt, the three column represents: Total people picked, total wait time and total running time(both in second).

[This is a sample statistics result based on the setting below.]



## Important parameters:

Here are the important parameters that I defined as macro.

```
#define TOTALTIME 10    //total number quantum for entire people-feed
#define FLOOR_PEOPLE_FEED 3    //ratio of people to feed for this amount of quantum(for stat)

#define TOTALFLOOR 30
#define TOTALCARTS 3
#define CART_CAPA 9    //how many people a cart can hold?

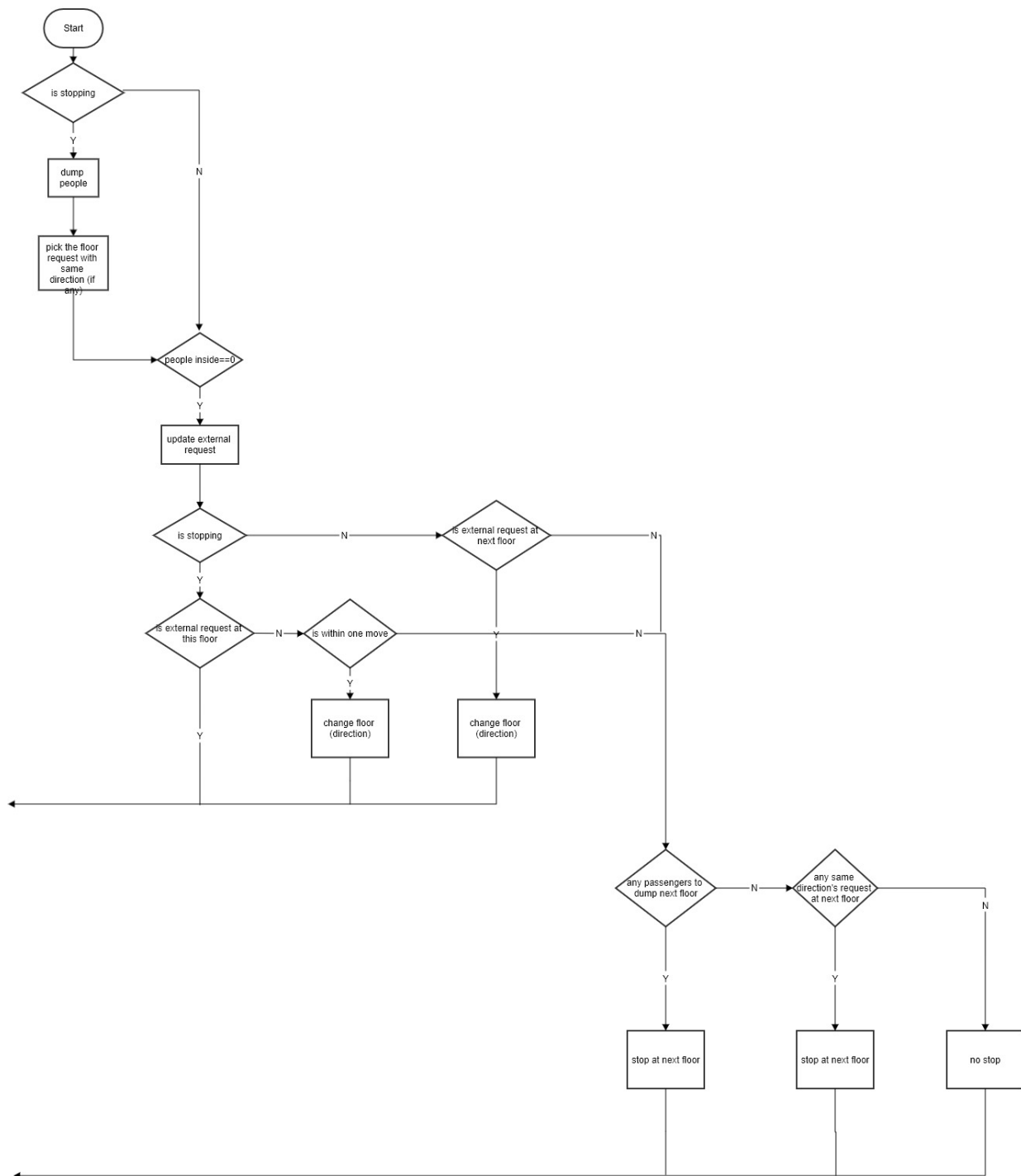
#define TIME_ONE_JUMP 3s    //time from one floor in stopping mode to next floor in stopping mode
#define TIME_ONE_STOP 2s    //time from one floor in moving mode to next floor in stopping mode(vice versa)
#define TIME_ONE_PASS 1s    //time from one floor in moving mode to next floor in moving mode
#define TIME_DUMP 1s    //time to unload people
#define TIME_LOAD 1s    //time to load people
#define PAUSE_CART_ARRIVE 1s    //time for cart to stop steadily
#define PAUSE_FLOORMGR 30s    //time gap for floor manager to put in the people feed in line
```

For people feed, I just use a simple model, each floorMgr feed TOTALTIME times waitpeople, for each feed, the count is only 1 or 0 based on the ratio of FLOOR\_PEOPLE\_FEED to TOTALTIME(It is convenient for statistics). Between two feeds, it will pause for PAUS\_FLOORMGR time. You can change this part for different performance metrics.

Then you can also change the floor number and cart number and the rendering would change correspondingly. The CART\_CAPA is limited to be less than 10 in rendering aspect for simplicity.

For the various speed of carts, I simplified it to be three combinations (each carts can speed up to max velocity or down to stop within one floor distance.) For the time of loading/unloading people, I also set it to be fixed among of time.

## Design process and work flow:



I have to say that this project really taught me several things. It is more an algorithm problem(or we say system design) than a engineer project. In terms of realness based on the common elevators, **the essential principle you abstract is more important than implementation skills. Those principle can be summarized directly by observation, or by summarizing corner cases and exclude the infeasible ones.**

Although I had a rough design at first, though the implementation I found that some points are not that essential and would block the functionalities of other aspects if you keep stubborn to pursue that.

Basically it is a producer-consumer problem. In reality each floor's up/down button can be pressed to generate floor requests, and carts would respond and finally clear those requests. Carts and floorRequestMgr should be implemented as individual threads because they are independent and they are core roles here.

For the WaitPeople feed, I finally decided to use separate methods to generate a vector of WaitPeople for each floor. Each idx represents rough time slot and can only feed 1/0 waitpeople based on predefined intensity, for simplicity and convenience of further statistics.

Floor requests can be handled in two ways, one is to use a central scheduler to 'push'(allocate) the new request to some cart, another is to let carts 'poll'(pre-empt) the floor request once they go close. Obviously the second one are more close to the reality.

Floor request should be put in a global containers for all carts to access concurrently. The most intuitive way is to use a vector for each floor's up/down request. And block the cart thread when the vector is empty. It is convenient to check the next floor's request for each carts and add request for each floorRequestMgr  $O(1)$ . But when one cart have no passengers inside while there is some request at some floor, they must access the whole vector  $O(N)$ . (I once considered to use separate locks for each floor to improve performance, but for accessing the request count, you have to use a whole lock for the vector, which means you need different grained locks and likely to have deadlock.) however, I choose another structure, set to keep the performance of operation to be  $O(\log n)$  because I think that both operation would comp up evenly.

For carts, except the pre-emptive pick, before picking some floor's request with opposite direction, they would need to do some check first. For example, when there is only one cart at floor5(10 floors in total) and floor 6 has a down request, before picking this request and change direction, in reality if there are some up request at floor 7, cart would definitely pick that first. To handle this, before pick the opposite floor request, you should check sequentially the following floors' request with same direction. I use a ExternalRequest to mark this and always double check if the request is there when cart is empty.

For rendering, I use window.h facilities to render on console. For better performance, I let each object to render itself in parallel. For carts we need to change the location and amount of people inside. For each floor we record the request button, and amount of waitPeople for direction.

RequestMgr: floor request collection

floorRequestMgr: to maintain waitPeople line and report floor request for each floor(and render)

Cart: to poll requests, check requests and load/dump people(and render)

Datafeed methods(data stored in datafeed.txt)

WaitPeople

For performance metrics, I add a timepoint field in waitpeople to indicate the starting waittime of each waitPeople when put in line, and calculate the waitTime when picked into carts. Each Carts maintains

the totalWaitTime of their own passengers to avoid locks. And we will add them and record pickedPeopl, totalWaitTime and totalRunning time for the program in err.txt.