CSCI 1300
Assignment 6: Elevator

Assigned: Thursday November 20, 2003
Due: Tuesday December 9, 2003

Honor Code Reminder

Please adhere to the policy presented on the first day of class: All work you submit to be graded must be entirely your own. Breaking this rule will result in an F grade for the entire course. You may ask friends and family for help in understanding the syntax of the C++ language, for working through examples in the text, and for understanding compilation and run-time errors, but they may not write or help you write the code for any homework assignment. If you are stuck, see the professor, TAs, or undergraduate gurus.

The Assignment

The goal of this project is to build a realistic simulation model of the elevators in the Engineering Center and the passengers who ride the elevators. The Engineering Center has nine floors, numbered 0 through 8, and three elevators. In the simulation, passengers will make a *call* for an elevator from a given floor in a particular direction (up or down) and will wait for the elevator to arrive, open its doors, and admit them. Once inside the elevator, passengers will select a *destination* floor and will ride the elevator to the destination.

The simulation involves three parts. First, you need to model the elevators themselves, keeping track of what floor they're on and in what direction they're moving. Second, you need to model the passengers—when and where they appear and what destinations they choose. Third, you need to simulate the *control policy* of the elevators, which determines the actions they take. Actions include: moving up, moving down, opening and closing doors.

In the real world, time flows continuously. In a digital computer, we model time in discrete *steps*. For this simulation, let's assume a step size that corresponds roughly to the passage of three seconds of time in the real world, enough time for the elevator to travel one floor up or down. Assume that time starts at time step 1. On each time step of the simulation, five operations must be performed:

1. Display the state of the simulation
2. Passengers generate new elevator calls
3. Elevator 1 acts
4. Elevator 2 acts
5. Elevator 3 acts

State of the simulation

The state of the simulation includes the following information:

- what floor each elevator is on
- what direction each elevator is headed (The meaning of the elevator's direction will be clearer when we explain the control of the elevators.)
- whether the doors of each elevator are open or closed
- how many passengers are waiting on each floor to go in each directions (if the number of passengers waiting is greater than zero, then the call button on that floor in that direction has been pressed)
- how many passengers in each elevator are waiting to go to each floor (if the number of passengers waiting is greater than zero, then the destination button for that floor has been pressed)
- Initially, assume that all elevators begin the simulation on floor 1, with the doors open, headed in direction "up". There are no passengers waiting to board elevators, and all elevators are empty.

Generating calls

At each time step, a new passenger can arrive at the elevators on floor F and make a call in direction D with probability p_call(F,D). If a passenger is already waiting to board an elevator on floor F in direction D, then the simulation must keep track of the fact that two passengers are now waiting to board in that direction. Note that you do not worry about the passenger's destination until they board the elevator.

The probability p_call(F,D) will be read from a file. By necessity, p_call(8,up) and p_call(0,down) will both be 0.

For fans of probability theory, note that the expected number of time steps between calls is 1/p_call(F,D) and the probability distribution characterizing the number of calls, n, in a given interval of t time steps follows a Poisson distribution:

$$p(n) = p\_call(F,D)^n \exp(-t\ p\_call(F,D))$$

Elevator actions

On each time step, an elevator can perform *one* of the following actions:

- move up one floor
- move down one floor
- open doors and discharge passengers (if any)
- admit passengers (if any) and close doors

Moving up or down one floor involves updating the elevator state.

Admitting passengers means clearing the count of the number of passengers waiting to board, and, for each passenger who was waiting, choosing a random destination from the probability distribution: p_destination(F,D,*destination*) where F is the current floor of the elevator, D is its current direction, and *destination* is the set of possible destination floors (0-8). This probability distribution is read from a file at the start of the simulation. You may assume that if the elevator is on floor F and the direction is up, the p_destination(F,D,F–k) = 0 for k>=1.

Discharging passengers means clearing the count of the number of passengers waiting to go to that floor.

Computing passenger wait time statistics

To evaluate a control policy, we need to know how long, on average, passengers waited from the time they first pressed the call button to the time when they arrived at their destination. One way to do this would be to maintain data structures that kept track of individual passengers. However, this will be quite complicated to code, and there is a simpler solution. Suppose you know the average time step at which a passenger presses the call button, and you know the average time step at which a passenger is discharged from the elevator at their destination floor. The difference between these averages is the mean wait time. You can compute the average call time by summing the time step at which each call is generated, and dividing by the total number of calls. You can compute the average discharge time by summing the time step at which a discharge is made for each passenger and dividing by total number of discharges.

For this scheme to work, all passengers must have arrived at their destinations by the time the simulation ends. We achieve this with the constant DEAD_TIME_AT_END (see below).

Constant variables

You should have a set of constant variables that control some parameters of the simulation, including:

RANDOM_SEED	seed for the random number generator
N_TIME_STEPS_IN_SIMULATION	number of time steps in simulation
STEP_BY_STEP_OUTPUT	if "true", then produce output for each time step; if "false", only print summary statistics

DEAD_TIME_AT_END	starting from time step N_TIME_STEPS_IN_SIMULATION – DEAD_TIME_AT_END + 1, do not allow passengers to make new calls. If DEAD_TIME_AT_END is large enough, say 200, we can be confident that all passengers will have arrived at their destinations by the end of the simulation (you can check for this to be certain), and therefore that the mean passenger wait-time statistic will be accurate.

Call probabilities

The call probabilities, p_call and p_destination should be read from a data file called `probabilities.dat`, formatted as reals in the range 0.0–1.0. First comes p_call then p_destination. We will post a sample file on the class handouts web page. The data will appear in row-major order, meaning that the order of numbers in the file will be:

p_call(0,down)
p_call(0,up)
p_call(1,down)
p_call(1,up)
...
p_call(8,down)
p_call(8,up)
p_destination(0,down,0)
p_destination(0,down,1)
...
p_destination(0,down,8)
p_destination(0,up,0)
...
p_destination(0,up,8)
...
p_desination(8,up,0)
...
p_desination(8,up,8)

Elevator control policy

Implement the following policy to control the elevators. Once you have this policy working and your code entirely bug-free, you may explore alternative policies with the goal of reducing the mean passenger wait time.

The policy for all three elevators is identical. Assume F is the current floor of the elevator being considered, and D is the current direction. What follows are the rules of the elevator, in priority order. That is, if the first rule applies, run it, otherwise check the next one, and so forth.

(1) If passengers are waiting to be discharged on F, open the doors and discharge passengers.

(2) If passengers are waiting to be admitted on floor F in direction D and the doors are closed, open the doors.

(3) If the doors are open, admit any passengers waiting on floor F for direction D and close doors.

(4) If the elevator contains passengers, move elevator one floor in direction D.

(5) If there are calls on floors above F (if D = up) or below F (if D = down), move elevator in direction D.

(6) If there are calls on floor F in the direction opposite D, then toggle D (up to down, or down to up) and open doors.

(7) If there are calls on floors below F (if D = up) or above F (if D = down), then toggle D and move the elevator on floor in the new direction D.

(8) If the current floor is greater than 1, set direction to down and move elevator in direction D

(9) If current floor is 0, set direction to up and move elevator in direction D.

(10) Do nothing.

Output

Your program should output certain information at each time step, including:

- the state of the elevators (the floor and direction of each, the status of the doors, and the number of passengers on board) at the start of the time step
- the state of the passengers waiting to board (the number on each floor and each direction) at the start of the time step
- the action taken by each elevator

At the end of the simulation, your program should output the average passenger wait.

For an executable version of Professor Mozer's program, go to the handouts web page. When STEP_BY_STEP_OUTPUT is set, the output at each time step looks something like this:

```
****************************** TIME 619 ******************************

                          0   1   2   3   4   5   6   7   8
                         ---------------------------------
up passengers waiting     0   4   0   0   0   0   0   0   0
elevator 1                .   .   .   .   .   .   O>  1   3
elevator 2                .   .   .   .   .   .   .   X>  .
elevator 3                .   .   .   .   .   .   .   .  <O
down passengers waiting   0   0   0   1   0   1   0   0   3

New floor calls:  0 up  1 up  3 up
Elevator 1 closes doors. [3]
Elevator 2 moves one floor up. [5]
Elevator 3 admits 3 passengers (destinations 2 6 0); closes doors. [3]
```

The lines "up passengers waiting" and "down passengers waiting" refers to passengers waiting on floors 0-8 who want to go up or down. In this example, four passengers are waiting on floor 1 to go up, 1 passenger is waiting on floors 3 and 5 and 3 passengers are waiting on floor 8 to go down.

For each elevator, the X or O symbol indicates the floor that the elevator is on. "X" means the door is closed, "O" open. The ">" and "<" indicate the current direction.

The display above shows all information except the number of passengers riding the elevator who want to disembark on the elevator's current floor.

Below the state of stimulation are new floor calls that were made during that time step (i.e., calls from passengers who want to ride the elevator from a given floor in a given direction). And below this are the actions taken by each elevator. The number in brackets is the rule that operated for that elevator at that time step. In this example, elevators 1 and 3 operated by rule 3; elevator 2 operated by rule 5.

Bells and Whistles

Once you have your program working to specification, you may create alternative elevator control policies, and see if you can devise one which improves on the policy we have given you. Hand in both the basic assignment, plus a version of your assignment with the improved policy (or better, put them both in one program, and ask the user which to run).

Handing in Your Program

Please follow these directions carefully. Your TA will have a difficult time grading the assignments as is. Points will be deducted if you do not follow our directions.

When you hand in your program, set the constants in your program as follows: RANDOM_SEED to 10, N_TIME_STEPS_IN_SIMULATION to 2000, STEP_BY_STEP_OUTPUT to false, and DEAD_TIME_AT_END to 200. In addition to uploading your source code, upload sample output from 50 time steps of your simulation using our sample data. Call this file `sample_output.txt`.

Grading

This assignment is worth a total of 100 points. However, in determining your final course grades, I have decided that Assignments 3 and 4 will be counted 50% more heavily than Assignments 1 and 2, and Assignments 5 and 6 will be counted twice as heavily as Assignments 1 and 2.

You are expected to adhere to the class style guide. Have a look at the guide to remind you of our expectations.

Very roughly, here is how we will deduct points in grading:

- up to 15 points off failure to follow style guidelines
- up to 10 points off each significant bug
- up to 15 points off poor breakdown into function
- up to 10 points off random-number generation errors
- up to 10 points off output is not well formatted