CS2030 Discrete Event Simulator Plus (Question)

Tags & Categories

Tags:

Categories:

Related Tutorials

Task Content

Discrete Event Simulator Plus

Problem Description

For this lab, you will extend the simulator to model the more complex behavior of shops, servers, and customers. Specically,

- FIFO queues for customers with a given maximum capacity;
- Two new events, `SERVER_REST` and `SERVER_BACK`, that simulates a server taking a rest and returning back from rest;
- Two types of servers, *human servers* who may rest after serving a customer, and self-checkout counters that never rest;
- Two types of customers, "greedy" customers and "typical" customers.

Customer Queues

Each server now has a queue of customer to allow multiple customers to queue up. A customer that chooses to join a queue joins at the tail. When a server is done serving a customer, it serves the next waiting customer at the head of the queue. Hence, the queue should be a first-in-first-out (FIFO) structure.

Taking a Rest

The servers are allowed to take occasional breaks. When a server finishes serving a customer, there is a probability P_r that the server takes a rest for a random amount of time T_r. During the break, the server does not serve the next waiting customer. Upon returning from the break, the server serves the next customer in queue immediately.

To implement this behavior, introduce two new events, `SERVER_REST` and `SERVER_BACK`, to simulating taking a break, and returning. These events should be generated and scheduled in the simulator when the server decides to rest.

Self-Checkout

To reduce waiting time, self-checkout counters have been set-up. These self-checkout counters never need to rest. Customers queue up for the self-checkout counter in the same way as *human* servers. There is one queue per self-checkout counter.

Customers' Choice of Queue

As before, when a customer arrives, he or she first scans through the servers (in order, from `1` to `k`) to see if there is an idle server (i.e. not serving a customer and not resting). If there is one, the customer will go to the server to be served. Otherwise, a typical customer just chooses the first queue (while scanning from servers `1` to `k`) that is still not full to join. However, other than the typical customer, a *greedy* customer is introduced that always chooses the queue with the fewest customers to join. In the case of a tie, the customer breaks the tie by choosing the first one while scanning from servers `1` to `k`.

If a customer cannot find any queue to join, it will leave the shop.

Writing and Generating Javadoc

You are encouraged to try documenting your classes and methods with Javadoc comments. For more details, see the javadoc guide.

The Task

This task is divided into several levels.

As there are progressively more input through the levels, specific details will be provided in the corresponding levels. Output for the level still comprises of the individual discrete events, and also the statistics at the end of the simulation.

Take note of the following assumptions:

- **There is no longer an upper bound for the number of customers**;
- The format of the input is always correct;
- Output of a `double` value, say `d`, is to be formatted with `String.format("%.3f", d);`

As this lab is a continuation form the previous one, your programs for `rng4` has been made available to you.

Just remember to

- define a `Main` class with the `main` method to handle input;
- compile your program using

```
$ javac -d . *.java
```

- check for output format correctness using the `diff` utility (see specific level for usage details). Note that only **one** test case is provided for this;
- check for styling errors by invoking `checkstyle`. For example, to check styling for all java files

```
$ checkstyle *.java
```

- save a copy of all source files into the appropriate level directory (see specific level for usage details).

Level 1

Implement the customer queues

Input to the program comprises (in order of presentation):

- an `int` value denoting the base seed for the `RandomGenerator` object;
- an `int` value representing the number of servers
- **an `int` value for the maximum queue length, Q_{max}**
- an `int` representing the number of customers (or the number of arrival events) to simulate
- a positive `double` parameter for the arrival rate, λ
- a positive `double` parameter for the service rate, μ

Each queue has a maximum capacity Q_{max}, and a customer cannot join a queue that is full. Note that a customer being served is not inside the queue. When a customer arrives and all the queues are full, then the customer leaves.

Clearly, if $Q_{max} = 1$, then the simulation reverts back to the one that allows only one waiting customer.

The following is a sample run of the program. Command to run the program is highlighted.

```
$ echo "1 1 1 5 1.0 1.0" | java Main
0.000 1 arrives
0.000 1 served by 1
0.313 1 done serving by 1
0.314 2 arrives
0.314 2 served by 1
0.417 2 done serving by 1
1.205 3 arrives
1.205 3 served by 1
1.904 3 done serving by 1
2.776 4 arrives
2.776 4 served by 1
2.791 4 done serving by 1
3.877 5 arrives
3.877 5 served by 1
4.031 5 done serving by 1
[0.000 5 0]
```

```
$ echo "1 2 1 10 1.0 1.0" | java Main
0.000 1 arrives
0.000 1 served by 1
0.313 1 done serving by 1
0.314 2 arrives
0.314 2 served by 1
0.417 2 done serving by 1
1.205 3 arrives
1.205 3 served by 1
1.904 3 done serving by 1
2.776 4 arrives
2.776 4 served by 1
2.791 4 done serving by 1
3.877 5 arrives
3.877 5 served by 1
3.910 6 arrives
3.910 6 served by 2
```

```
3.922 6 done serving by 2
4.031 5 done serving by 1
9.006 7 arrives
9.006 7 served by 1
9.043 8 arrives
9.043 8 served by 2
9.105 9 arrives
9.105 9 waits to be served by 1
9.160 10 arrives
9.160 10 waits to be served by 2
10.484 7 done serving by 1
10.484 9 served by 1
10.781 9 done serving by 1
11.636 8 done serving by 2
11.636 10 served by 2
11.688 10 done serving by 2
[0.386 10 0]
```

```
$ echo "1 2 2 10 1.0 1.0" | java Main
0.000 1 arrives
0.000 1 served by 1
0.313 1 done serving by 1
0.314 2 arrives
0.314 2 served by 1
0.417 2 done serving by 1
1.205 3 arrives
1.205 3 served by 1
1.904 3 done serving by 1
2.776 4 arrives
2.776 4 served by 1
2.791 4 done serving by 1
3.877 5 arrives
3.877 5 served by 1
3.910 6 arrives
3.910 6 served by 2
3.922 6 done serving by 2
4.031 5 done serving by 1
9.006 7 arrives
9.006 7 served by 1
9.043 8 arrives
9.043 8 served by 2
9.105 9 arrives
9.105 9 waits to be served by 1
9.160 10 arrives
9.160 10 waits to be served by 1
10.484 7 done serving by 1
10.484 9 served by 1
10.781 9 done serving by 1
10.781 10 served by 1
10.833 10 done serving by 1
11.636 8 done serving by 2
[0.300 10 0]
```

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test1.in | diff - test1.out
```

Make a copy of your Java programs to the level directory by typing the Unix commands

```
$ mkdir simplus1
$ cp *.java simplus1
```

Level 2

Implementing server breaks

Input to the program comprises (in order of presentation):

- an `int` value denoting the base seed for the `RandomGenerator` object;
- an `int` value representing the number of servers
- an `int` value for the maximum queue length, Q_{max}
- an `int` representing the number of customers (or the number of arrival events) to simulate
- a positive `double` parameter for the arrival rate, λ
- a positive `double` parameter for the service rate, μ
- **a positive `double` parameter for the resting rate, ρ**
- **a `double` parameter for the probability of resting, P_r**

Include two events `SERVER_REST` and `SERVER_BACK`, to simulating taking a break, and returning. These events should be generated and scheduled in the simulator when the server decides to rest.

To decide if the server should rest, a random number uniformly drawn from `[0, 1]` is generated using the `RandomGenerator` method `genRandomRest()`. If the value returned is less than P_r, the server rests with a `SERVER_REST` event generated. Otherwise, it continues serving the next customer.

As soon as the server rests, a random rest period T_r is generated using the `RandomGenerator` method `genRestPeriod()`. This variable is an exponential random variable, governed by the resting rate, ρ. A `SERVER_BACK` event will be scheduled at T_r + `now`.

The following is a sample run of the program. Command to run the program is highlighted.

```
$ echo "1 2 2 10 1.0 1.0 0 0" | java Main
0.000 1 arrives
0.000 1 served by 1
0.313 1 done serving by 1
0.314 2 arrives
0.314 2 served by 1
0.417 2 done serving by 1
1.205 3 arrives
1.205 3 served by 1
1.904 3 done serving by 1
2.776 4 arrives
2.776 4 served by 1
```

do it agyer a done event

```
2.791 4 done serving by 1
3.877 5 arrives
3.877 5 served by 1
3.910 6 arrives
3.910 6 served by 2
3.922 6 done serving by 2
4.031 5 done serving by 1
9.006 7 arrives
9.006 7 served by 1
9.043 8 arrives
9.043 8 served by 2
9.105 9 arrives
9.105 9 waits to be served by 1
9.160 10 arrives
9.160 10 waits to be served by 1
10.484 7 done serving by 1
10.484 9 served by 1
10.781 9 done serving by 1
10.781 10 served by 1
10.833 10 done serving by 1
11.636 8 done serving by 2
[0.300 10 0]
```

```
$ echo "1 2 2 20 1.0 1.0 0.1 0.5" | java Main
0.000 1 arrives
0.000 1 served by 1
0.313 1 done serving by 1
0.314 2 arrives
0.314 2 served by 1
0.417 2 done serving by 1
0.417 server 1 rest
1.205 3 arrives
1.205 3 served by 2
1.904 3 done serving by 2
1.904 server 2 rest
2.752 server 2 back
2.776 4 arrives
2.776 4 served by 2
2.791 4 done serving by 2
3.556 server 1 back
3.877 5 arrives
3.877 5 served by 1
3.910 6 arrives
3.910 6 served by 2
3.922 6 done serving by 2
3.922 server 2 rest
4.031 5 done serving by 1
4.771 server 2 back
9.006 7 arrives
9.006 7 served by 1
9.043 8 arrives
9.043 8 served by 2
9.105 9 arrives
```

```
9.105 9 waits to be served by 1
9.160 10 arrives
9.160 10 waits to be served by 1
9.225 11 arrives
9.225 11 waits to be served by 2
10.148 12 arrives
10.148 12 waits to be served by 2
10.484 7 done serving by 1
10.484 9 served by 1
10.781 9 done serving by 1
10.781 server 1 rest
11.205 13 arrives
11.205 13 waits to be served by 1
11.636 8 done serving by 2
11.636 11 served by 2
11.688 11 done serving by 2
11.688 12 served by 2
12.429 14 arrives
12.429 14 waits to be served by 2
13.109 15 arrives
13.109 15 waits to be served by 2
14.644 server 1 back
14.644 10 served by 1
15.013 10 done serving by 1
15.013 server 1 rest
15.178 12 done serving by 2
15.178 14 served by 2
15.264 16 arrives
15.264 16 waits to be served by 1
15.338 14 done serving by 2
15.338 15 served by 2
15.524 17 arrives
15.524 17 waits to be served by 2
15.940 18 arrives
15.940 18 waits to be served by 2
17.793 19 arrives
17.793 19 leaves
18.207 15 done serving by 2
18.207 17 served by 2
18.765 20 arrives
18.765 20 waits to be served by 2
19.103 17 done serving by 2
19.103 18 served by 2
20.146 18 done serving by 2
20.146 server 2 rest
40.474 server 1 back
40.474 13 served by 1
40.480 13 done serving by 1
40.480 16 served by 1
41.056 16 done serving by 1
41.056 server 1 rest
44.634 server 1 back
```

```
57.110 server 2 back
57.110 20 served by 2
57.852 20 done serving by 2
57.852 server 2 rest
60.022 server 2 back
[6.025 19 1]
```

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test2.in | diff - test2.out
```

Make a copy of your Java programs to the level directory by typing the Unix commands

```
$ mkdir simplus2
$ cp *.java simplus2
```

Level 3

Include self-checkout counters

Input to the program comprises (in order of presentation):

- an `int` value denoting the base seed for the `RandomGenerator` object;
- an `int` value representing the number of servers
- **an `int` value representing the number of self-checkout counters, N_{self}**
- an `int` value for the maximum queue length, Q_{max}
- an `int` representing the number of customers (or the number of arrival events) to simulate
- a positive `double` parameter for the arrival rate, λ
- a positive `double` parameter for the service rate, μ
- a positive `double` parameter for the resting rate, ρ
- a `double` parameter for the probability of resting, P_r

There are N_{self} self-checkout counters set up. In particular, if there are `k` human servers, then the self-checkout counters are identified from `k + 1` onwards.

The following is a sample run of the program. Command to run the program is highlighted.

```
$ echo "1 2 0 2 20 1.0 1.0 0.1 0.5" | java Main
0.000 1 arrives
0.000 1 served by 1
0.313 1 done serving by 1
0.314 2 arrives
0.314 2 served by 1
0.417 2 done serving by 1
0.417 server 1 rest
1.205 3 arrives
1.205 3 served by 2
1.904 3 done serving by 2
```

```
1.904 server 2 rest
2.752 server 2 back
2.776 4 arrives
2.776 4 served by 2
2.791 4 done serving by 2
3.556 server 1 back
3.877 5 arrives
3.877 5 served by 1
3.910 6 arrives
3.910 6 served by 2
3.922 6 done serving by 2
3.922 server 2 rest
4.031 5 done serving by 1
4.771 server 2 back
9.006 7 arrives
9.006 7 served by 1
9.043 8 arrives
9.043 8 served by 2
9.105 9 arrives
9.105 9 waits to be served by 1
9.160 10 arrives
9.160 10 waits to be served by 1
9.225 11 arrives
9.225 11 waits to be served by 2
10.148 12 arrives
10.148 12 waits to be served by 2
10.484 7 done serving by 1
10.484 9 served by 1
10.781 9 done serving by 1
10.781 server 1 rest
11.205 13 arrives
11.205 13 waits to be served by 1
11.636 8 done serving by 2
11.636 11 served by 2
11.688 11 done serving by 2
11.688 12 served by 2
12.429 14 arrives
12.429 14 waits to be served by 2
13.109 15 arrives
13.109 15 waits to be served by 2
14.644 server 1 back
14.644 10 served by 1
15.013 10 done serving by 1
15.013 server 1 rest
15.178 12 done serving by 2
15.178 14 served by 2
15.264 16 arrives
15.264 16 waits to be served by 1
15.338 14 done serving by 2
15.338 15 served by 2
15.524 17 arrives
15.524 17 waits to be served by 2
```

```
15.940 18 arrives
15.940 18 waits to be served by 2
17.793 19 arrives
17.793 19 leaves
18.207 15 done serving by 2
18.207 17 served by 2
18.765 20 arrives
18.765 20 waits to be served by 2
19.103 17 done serving by 2
19.103 18 served by 2
20.146 18 done serving by 2
20.146 server 2 rest
40.474 server 1 back
40.474 13 served by 1
40.480 13 done serving by 1
40.480 16 served by 1
41.056 16 done serving by 1
41.056 server 1 rest
44.634 server 1 back
57.110 server 2 back
57.110 20 served by 2
57.852 20 done serving by 2
57.852 server 2 rest
60.022 server 2 back
[6.025 19 1]
```

```
$ echo "1 2 1 2 20 1.0 1.0 0.1 0.5" | java Main
0.000 1 arrives
0.000 1 served by server 1
0.313 1 done serving by server 1
0.314 2 arrives
0.314 2 served by server 1
0.417 2 done serving by server 1
0.417 server 1 rest
1.205 3 arrives
1.205 3 served by server 2
1.904 3 done serving by server 2
1.904 server 2 rest
2.752 server 2 back
2.776 4 arrives
2.776 4 served by server 2
2.791 4 done serving by server 2
3.556 server 1 back
3.877 5 arrives
3.877 5 served by server 1
3.910 6 arrives
3.910 6 served by server 2
3.922 6 done serving by server 2
3.922 server 2 rest
4.031 5 done serving by server 1
4.771 server 2 back
9.006 7 arrives
9.006 7 served by server 1
```

```
9.043 8 arrives
9.043 8 served by server 2
9.105 9 arrives
9.105 9 served by self-check 3
9.160 10 arrives
9.160 10 waits to be served by server 1
9.225 11 arrives
9.225 11 waits to be served by server 1
9.402 9 done serving by self-check 3
10.148 12 arrives
10.148 12 served by self-check 3
10.200 12 done serving by self-check 3
10.484 7 done serving by server 1
10.484 10 served by server 1
11.205 13 arrives
11.205 13 served by self-check 3
11.574 13 done serving by self-check 3
11.636 8 done serving by server 2
11.636 server 2 rest
12.429 14 arrives
12.429 14 served by self-check 3
12.589 14 done serving by self-check 3
13.109 15 arrives
13.109 15 served by self-check 3
13.974 10 done serving by server 1
13.974 11 served by server 1
14.869 11 done serving by server 1
15.264 16 arrives
15.264 16 served by server 1
15.500 server 2 back
15.524 17 arrives
15.524 17 served by server 2
15.531 17 done serving by server 2
15.531 server 2 rest
15.940 18 arrives
15.940 18 waits to be served by server 1
15.978 15 done serving by self-check 3
16.307 16 done serving by server 1
16.307 18 served by server 1
16.883 18 done serving by server 1
17.793 19 arrives
17.793 19 served by server 1
18.535 19 done serving by server 1
18.765 20 arrives
18.765 20 served by server 1
21.773 20 done serving by server 1
40.992 server 2 back
[0.322 20 0]
```

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test3.in | diff - test3.out
```

Make a copy of your Java programs to the level directory by typing the Unix commands

```
$ mkdir simplus3
$ cp *.java simplus3
```

Level 4

Include greedy customers

Input to the program comprises (in order of presentation):

- an `int` value denoting the base seed for the `RandomGenerator` object;
- an `int` value representing the number of servers
- an `int` value representing the number of self-checkout counters, N_{self}
- an `int` value for the maximum queue length, Q_{max}
- an `int` representing the number of customers (or the number of arrival events) to simulate
- a positive `double` parameter for the arrival rate, λ
- a positive `double` parameter for the service rate, μ
- a positive `double` parameter for the resting rate, ρ
- a `double` parameter for the probability of resting, P_r
- **a `double` parameter for the probability of a greedy customer occurring, P_g**

An arriving customer is a greedy customer with probability P_g. To decide whether a typical or greedy customer is created, a random number uniformly drawn from `[0, 1]` is generated with the `RandomGenerator` method `genCustomerType()`. If the value returned is less than P_g, a greedy customer is generated, otherwise, a typical customer is generated.

The following is a sample run of the program. Command to run the program is highlighted.

```
$ echo "1 2 1 2 20 1.0 1.0 0.1 0.5 0" | java Main
0.000 1 arrives
0.000 1 served by server 1
0.313 1 done serving by server 1
0.314 2 arrives
0.314 2 served by server 1
0.417 2 done serving by server 1
0.417 server 1 rest
1.205 3 arrives
1.205 3 served by server 2
1.904 3 done serving by server 2
1.904 server 2 rest
2.752 server 2 back
2.776 4 arrives
2.776 4 served by server 2
2.791 4 done serving by server 2
3.556 server 1 back
3.877 5 arrives
3.877 5 served by server 1
3.910 6 arrives
```

```
3.910 6 served by server 2
3.922 6 done serving by server 2
3.922 server 2 rest
4.031 5 done serving by server 1
4.771 server 2 back
9.006 7 arrives
9.006 7 served by server 1
9.043 8 arrives
9.043 8 served by server 2
9.105 9 arrives
9.105 9 served by self-check 3
9.160 10 arrives
9.160 10 waits to be served by server 1
9.225 11 arrives
9.225 11 waits to be served by server 1
9.402 9 done serving by self-check 3
10.148 12 arrives
10.148 12 served by self-check 3
10.200 12 done serving by self-check 3
10.484 7 done serving by server 1
10.484 10 served by server 1
11.205 13 arrives
11.205 13 served by self-check 3
11.574 13 done serving by self-check 3
11.636 8 done serving by server 2
11.636 server 2 rest
12.429 14 arrives
12.429 14 served by self-check 3
12.589 14 done serving by self-check 3
13.109 15 arrives
13.109 15 served by self-check 3
13.974 10 done serving by server 1
13.974 11 served by server 1
14.869 11 done serving by server 1
15.264 16 arrives
15.264 16 served by server 1
15.500 server 2 back
15.524 17 arrives
15.524 17 served by server 2
15.531 17 done serving by server 2
15.531 server 2 rest
15.940 18 arrives
15.940 18 waits to be served by server 1
15.978 15 done serving by self-check 3
16.307 16 done serving by server 1
16.307 18 served by server 1
16.883 18 done serving by server 1
17.793 19 arrives
17.793 19 served by server 1
18.535 19 done serving by server 1
18.765 20 arrives
18.765 20 served by server 1
```

```
21.773 20 done serving by server 1
40.992 server 2 back
[0.322 20 0]
```

```
$ echo "1 2 1 2 20 1.0 1.0 0.1 0.5 0.9" | java Main
0.000 1(greedy) arrives
0.000 1(greedy) served by server 1
0.313 1(greedy) done serving by server 1
0.314 2(greedy) arrives
0.314 2(greedy) served by server 1
0.417 2(greedy) done serving by server 1
0.417 server 1 rest
1.205 3(greedy) arrives
1.205 3(greedy) served by server 2
1.904 3(greedy) done serving by server 2
1.904 server 2 rest
2.752 server 2 back
2.776 4(greedy) arrives
2.776 4(greedy) served by server 2
2.791 4(greedy) done serving by server 2
3.556 server 1 back
3.877 5(greedy) arrives
3.877 5(greedy) served by server 1
3.910 6(greedy) arrives
3.910 6(greedy) served by server 2
3.922 6(greedy) done serving by server 2
3.922 server 2 rest
4.031 5(greedy) done serving by server 1
4.771 server 2 back
9.006 7(greedy) arrives
9.006 7(greedy) served by server 1
9.043 8(greedy) arrives
9.043 8(greedy) served by server 2
9.105 9(greedy) arrives
9.105 9(greedy) served by self-check 3
9.160 10 arrives
9.160 10 waits to be served by server 1
9.225 11(greedy) arrives
9.225 11(greedy) waits to be served by server 2
9.402 9(greedy) done serving by self-check 3
10.148 12(greedy) arrives
10.148 12(greedy) served by self-check 3
10.200 12(greedy) done serving by self-check 3
10.484 7(greedy) done serving by server 1
10.484 10 served by server 1
11.205 13(greedy) arrives
11.205 13(greedy) served by self-check 3
11.574 13(greedy) done serving by self-check 3
11.636 8(greedy) done serving by server 2
11.636 server 2 rest
12.429 14(greedy) arrives
12.429 14(greedy) served by self-check 3
```

Check the format correctness of the output by typing the following Unix command

```
$ java Main < test4.in | diff - test4.out
```

Make a copy of your Java programs to the level directory by typing the Unix commands

```
$ mkdir simplus4
$ cp *.java simplus4
```