Market Simulation User Report

Maksym Turkot

03/28/2021

Introduction

This document outlines overview of project goals, challenges, and solutions to those challenges. It also provides a user guide, a diagram of major project components, program verification discussion, and data analysis.

Project Goals

The goal of the project was to create a simulation of a market and calculate certain metrics based on multiple runs of different configurations of the simulation (different numbers of different types of stalls, and their speed).

Determining time customers spent standing in line:
Goal:
In order to determine the average time customer spent waiting in line, the simulation records times when customer entered the line, and when customer left. Time in line is then calculated and added to the list. At the end Experiment retrieves this list and calculates average, minimum, and maximum value for each customer. This data is then written to output file.
Challenges:
It was a little tricky to get the data onto the same CSV file in an organized fashion.
Solutions:
I figured how to use dataToCsV method to put relevant data in order in the file. Using separate list for each metric was also useful.

Determine the average line size at each stand:
Goal:
Every minute (clock iteration) Experiment gathers line length data from each stand and adds those values to the list. In the end, those values are averages for each stall.
Challenges:
This metric is slightly different from other two, in that it is calculated every minute, instead of upon certain event. It wasn't apparent at first where to put the data recorder.
Solutions:
Since stall's sell() method is called every minute, without exceptions, data recorder was put there. A separate method would probably make the code cleaner.

Determine the average length of time customers spend shopping:
Goal:
Each Customer knows initially when they will enter the market, and record the minute (clock tick) when they leave. Then, once the market closes, Experiment asks each customer to calculate time they spent in the market (difference between time entered and time departed), and records this data.
Challenges:

Recording time customer entered was easy — it was determined upon initialization. Time left, however, involved three situations: customer is done shopping and leaves before market closes; customer leaves because they were in a queue and market closed; customer finishes their purchase and leaves because market is closed now.
Solutions:
The first event is simple: when customer has no more groceries, they record the time and leave. The second and third event is handled by the market. When market is closed and customer is not shopping, they should leave. When market is closed and they are still purchasing, they will simply leave once they try to move to the next stall.

Using Gaussian Random Distribution:
Goals:
- In the Customer class, their arrival at the market will be calculated as a distribution with a median of 20 minutes after the market has opened, but not less than 0.
- In the Customer, the grocery list is generated as a list of random values between 1 and 6. The list is never bigger than 6 and never smaller than 1.
- In the Stall class, the time each purchase takes place is be set by a configuration, and varied with the Gaussian distribution.

Challenges:
Using different seeds for the market was a little more difficult than expected. Initially Gaussian was static, so it wasn't apparent how to pass a new seed to it. Making every class remember the seed and pass it to the method did not seem efficient.
Solutions:
Gaussian was made non-static and would have an instant initialized by the market for each run.

Create six very different configurations:
Goal:
Configurations differ by the number of stalls for each product and the time it takes each stall to sell a product. Since the goal is to help the manager determine the number and type of stalls, these are the main and only configuration variables. There is a separate configuration file for each market configuration, and ConfigManager retrieves this data. After that, it provides it to Experiment.
Challenges:
The first challenge was to have the program read configuration files and provide data to the market. Next, it was necessary to create six distinct configuration that made sense.
Solutions:
I used first lab code, as well as provided data analysis code to structure the file management classes. I thought about different configurations of number of stalls and speed and used provided examples to create different configurations.

Create simulations of stalls based on queues:
Goal:
For this goal I created a Queue class, as was done in the labs. Each stall instantiates one queue, and each queue has customers as nodes. Each queue can add, remove, and do some other actions to its nodes, as the stall orders.
Challenges:
This process was not too challenging, since the structure is fairly straightforward, and I have practiced list creation.

Create simulations of customers using state machines:
Goal:

Each of three types of customers has a unique implementation of a state machine behavior. First one simply goes through their list in order and visits stalls corresponding to next item. Second one searches for stall that has the good on the list and has the shortest queue. Third one simply chooses the first stall they find that has the desired good.
Challenges:
Making sure that all cases are covered was not as simple as seemed initially. There were quite a few NullPointerExceptions, and then incorrect program behavior along the way.
Solutions:
Going back to the drawing board and then rewriting the code made the program much cleaner and it ended up behaving as expected.

User Guide

PROGRAM STRUCTURE:
In the project folder user may find the “data” folder. In it, there is a “configs” folder, containing configuration files, “logs” folder, containing program run logs, and “output” folder, containing raw data CSV files.

CONFIGURING AND RUNNING THE PROGRAM:
In order to configure the program:
User may change configuration files in the “configs” folder described above. The format of config file should be as follows: five comma-separated positive integers (which will indicate number of stalls for each type) in the first line, and five corresponding comma-separated positive integers (which will indicate speed of each stall type) in the second line.

In order to run the program:
1. Run the “package.bluej” file.
2. Right-click the “Experiment” class.
3. Click on the “void runExperiment()” method.
4. The program runs, and once blue bar in the bottom-right disappears, generated log and output files may be found in corresponding folders described above.

Project Components and Diagram

COMPONENTS:

1. Class Experiment

This class requests configuration info from ConfigManager, instantiates market objects with corresponding parameters, runs each configuration five times with different seeds, sets the clock running for a given market, gathers simulation info and orders OutputManager to write performance logs and output files.

2. Class ConfigManager

This class reads data from config files and passes it to the Experiment.

3. Class OutputManager

This class creates output files with data provided.

3. Class Clock

This class implements a clock by using a loop to iterate through ticks, and runs market actions each tick.

4. Class Gaussian

This class implements the Gaussian randomizer. It receives a value, and returns a value equal to or close to it, by using Gaussian distribution.

4. Class Market

This class initializes customer, stall, and gaussian objects with corresponding parameters received from the Experiment. It then runs the market actions by telling each customer to do next shopping step and stalls to do next selling step as the time goes by (the clock ticks). It also gathers necessary log info.

5. Class Stall

This class constructs stall objects. It also manages the selling process, and moves customers to its queue upon request and removes them from it.

6. Class Queue

This class constructs a queue. It manages queue's actions, such as adding, removing, retrieving, and counting customers in it.

7. Class Customer

This class constructs customers, generates their groceries, and calculates average times standing in line and shopping. Shopping logic is implemented in children classes

8. Class CustomerOne

This class extends Customer and implements unique shopping logic. This customer simply goes through their list in order and visits stalls corresponding to next item.

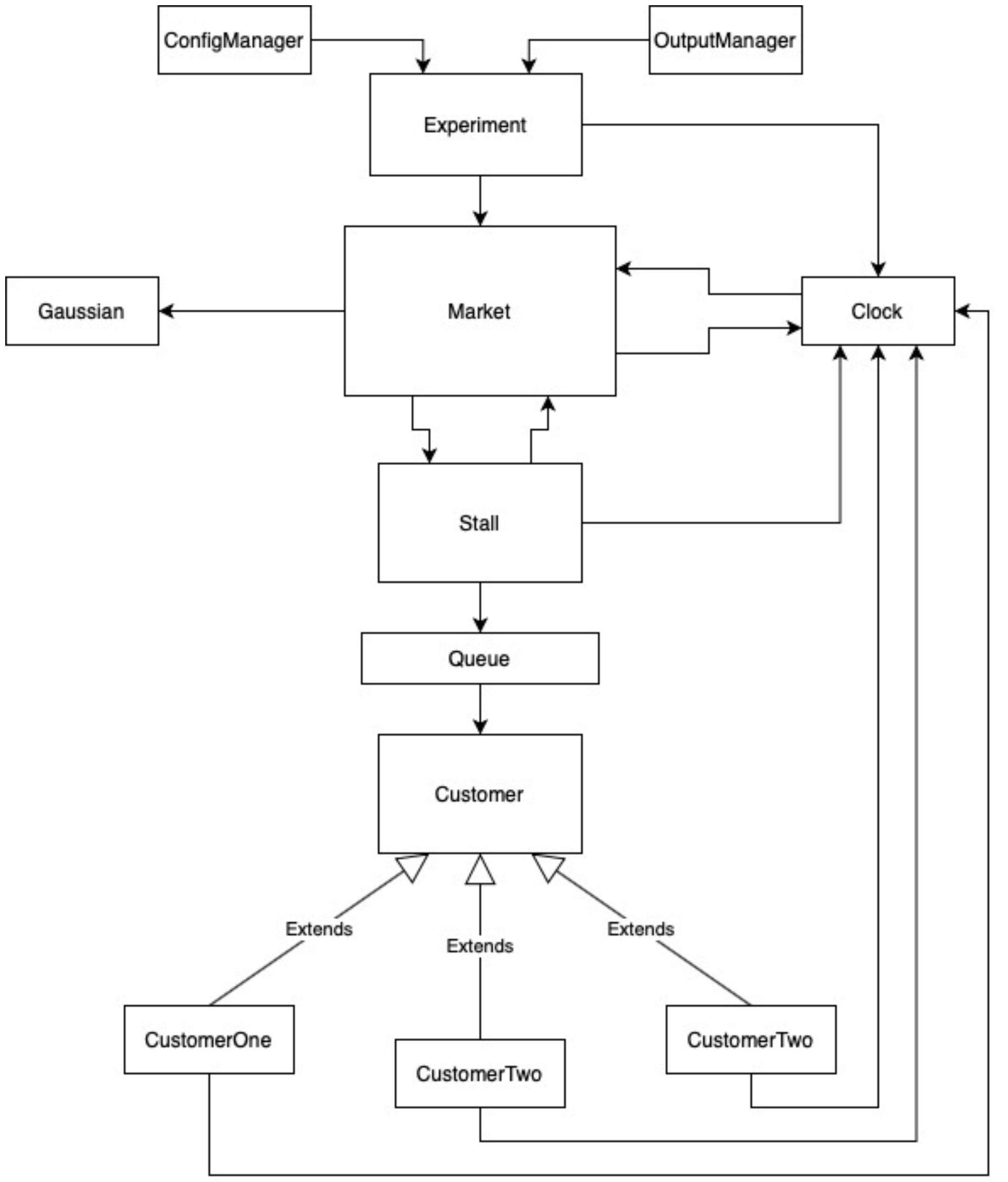
9. Class CustomerTwo

This class extends Customer and implements unique shopping logic. This customer searches for stall that has the good on the list and has the shortest queue.

10. Class CustomerThree

This class extends Customer and implements unique shopping logic. This customer simply chooses the first stall they find that has the desired good.

DIAGRAM:


ConfigManager
OutputManager
Experiment
Gaussian
Market
Clock
Stall
Queue
Customer
Extends
Extends
Extends
CustomerOne
CustomerTwo
CustomerTwo

Verification of Program's Functionality

Program's functionality was verified and tested using six config files with different configuration parameters. Each configuration outlined certain combinations of numbers of stalls selling each good, and time it took each stall to complete a trade. Configurations can be summarized as follows:

1. One stall for each product, most are slow.
2. Two stalls for each product, most are slow.
3. One stall for each product, all are fast.
4. One stall for each product, with half being slow and another half being fast.
5. Three products have two slow stalls, three have only one quick stall.
6. Different number of stalls for various products with different speeds.

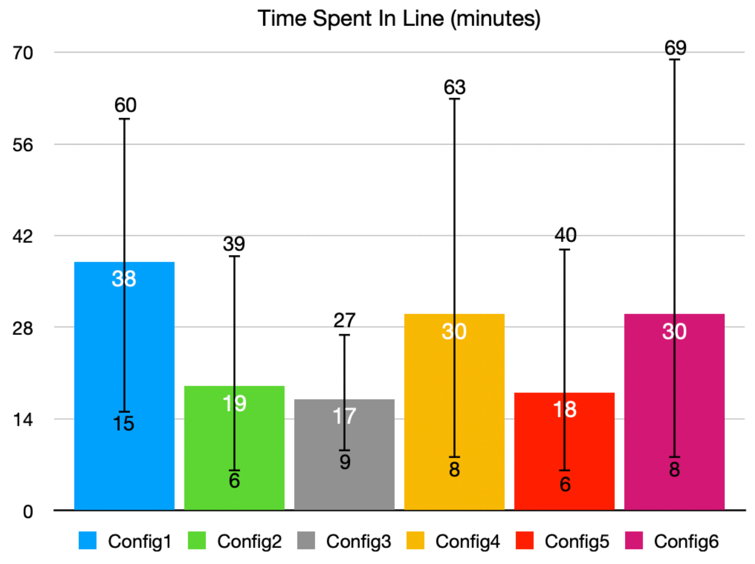
Each configuration was run five times with different random seeds. This produced different data for each run, helping eliminate random error in the data.

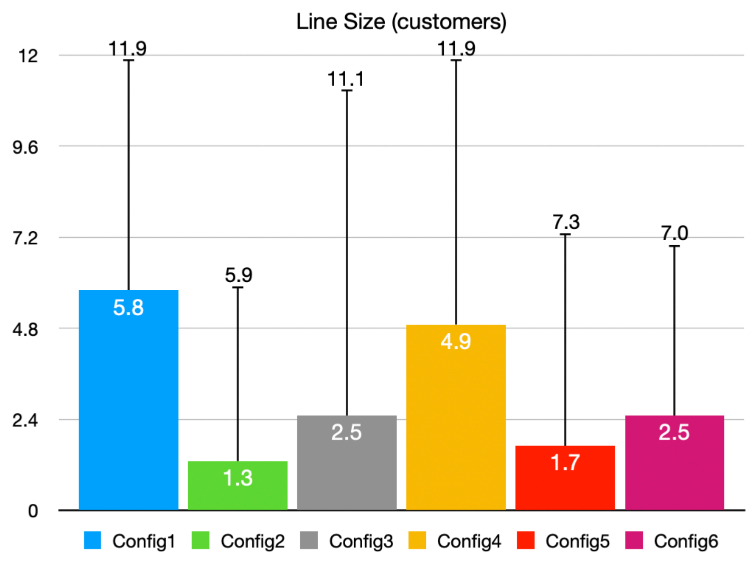
Program's performance for each configuration and random seed is written in the corresponding log file. This tremendously helped with debuting, and was an excellent verification tool. Additionally, final collected data is returned as output files, which also allow to check if result makes sense, and program functions correctly. Key functionality was unit-tested.

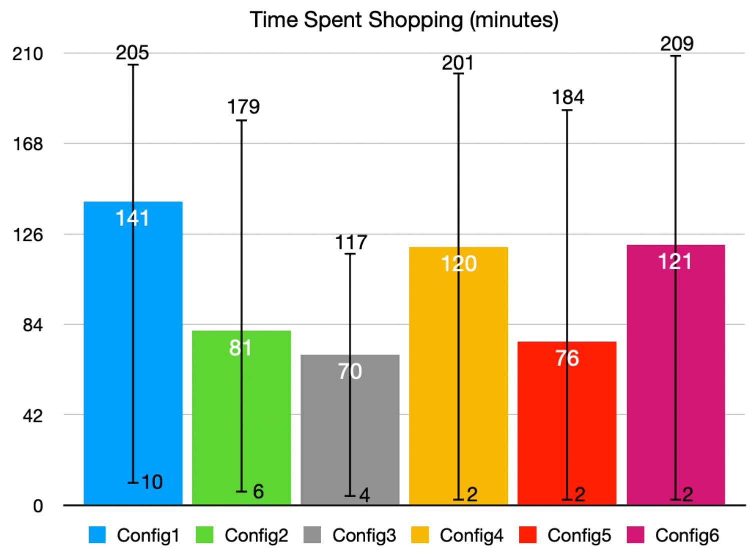
Data Analysis

Program gathered average time each customer spent standing in line, as well as minimum and maximum time, average length of a queue for each stall, as well as minimum and maximum length, and time each customer spent in the market. This data was summarized outside of the program by taking the average value of each metric for a configuration as a whole.

Program output data is summarized in the following graphs. (The midpoint is average value, upper bound is maximum, lower bound is minimum. For Line Size graph minimum line length is always zero.)







As can be seen from the data, configurations 2, 3, and 5 (C2, C3, C5) are the most efficient. From the Time Spent in Line graph, C3 has the lowest average waiting time for customers (17 minutes). Indeed, even if there is only one stall per item per market, speed of purchase fully compensates for the lack of stalls, even though the minimum time spent in line is smaller for C2, where there are two slow stalls per product. In terms of total time spent shopping, C3 performs at 70 minutes on average, which is also the smallest in the simulation. Efficiency is confirmed by the smallest maximum time shopping, which is way lower than the closest runner-up.

C2, however, performs better in terms of line size. Since now there are two stalls per product, this allows lines to be split. C2 is quite close to C3. However, maximum time spent shopping is way bigger (179 compared to 117).

C5 is also a very attractive option for the market manager. It performs well in all metrics, with reasonable maximum and minimum values. Its configuration of stalls both splits lines for some purchases while cutting the overall waiting time for others. It is reasonable to recommend market manager to use configuration 5.

Configurations 1, 4, and 5 are relatively inefficient when it comes to all metrics. Two configure the most efficient market, it is important to balance waiting time by increasing speed of some stalls, and having multiple stalls to split up lines.