CSCI 1933 Project 4

Bus Simulation

Due Date: Friday, April 20th before 11:55pm

Instructions

- **Due:** The project is due on **Friday, April 20** before **11:55 PM**, to be submitted via Moodle.
- **Submission:** Submit a `zip` or `tar` archive on Moodle containing all your `java` source files, as well as your writeup and readme files. You are allowed to change or modify your submission prior to the deadline, so submit early and often, and *verify that all your files are in the submission.* If you are working with a partner, exactly one of you should submit.

  Failure to submit the correct files will result in a score of zero for all missing parts. Late submissions and submissions in an abnormal format (such as `.rar`) will be penalized. Only submissions made via Moodle are acceptable.
- **Writeup format:** The writeup should be submitted as a `pdf` file, which can be exported to by all common word processors and generated from LaTeX files. If you have questions about creating a `pdf` file, please ask for help before the deadline.
- **Readme format:** The readme may be submitted as either a `pdf` file, a Markdown (`md`) file, or as a text file.
- **Partners:** You may work alone or with *one* partner. Please place you and your partner's name, student ID, and x500 in a comment in each `.java` file you submit. Do not share code with students other than your partner.
- **Code Sharing:** If you use online resources to share code with your partner, please ensure the code is private. Public repositories containing your code are prohibited because other students may copy your work. As always, make sure that you credit any sources of ideas or code (including code from the course website).
- **Java Version:** The TAs will grade your code using Java 8. Please ensure your code works in Java 8, which is the version installed on the cselabs computers. Java 6 and Java 7 will work with Java 8 (at least for the purposes of this course).
- **Grading:** We will not be using unit tests to grade this project. You are free to make any design decisions you want, but your code must be reasonably clean, well-designed, and commented thoroughly. Your code may receive a penalty if it is confusing or demonstrates poor knowledge of Java.

  Grading will be done by the TAs; please address grading problems to them **privately**.
- **Questions:** Please post questions related to this project on the Moodle forum. **Do not e-mail the TAs or the class email unless you have private questions.** All questions about how to code Java or questions about understanding this instruction document belong on Moodle. However, please avoid posting your answer code on Moodle, even if it is not working.

Code Style

Part of your grade will be decided based on the "code style" demonstrated by your programming. In general, all projects will involve a style component. This should not be intimidating, but it is fundamentally important. The following items represent "good" coding style:

- Use effective comments to document what important variables, functions, and sections of the code are for. In general, the TA should be able to understand your logic through the comments left in the code.

  Try to leave comments as you program, rather than adding them all in at the end. Comments should not feel like arbitrary busy work - they should be written assuming the reader is fluent in Java, yet has no idea how your program works or why you chose certain solutions.
- Use effective and standard indentation.
- Use descriptive names for variables. Use standard Java style for your names: `ClassName`, `functionName`, `variableName` for structures in your code, and `ClassName.java` for the file names.

Try to avoid the following stylistic problems:

- Missing or highly redundant, useless comments. `int a = 5; //Set a to be 5` is not helpful.
- Disorganized and messy files. Poor indentation of braces (`{` and `}`).
- Incoherent variable names. Names such as `m` and `numberOfIndicesToCount` are not useful. The former is too short to be descriptive, while the latter is much too descriptive and redundant.
- Slow functions. While some algorithms are more efficient than others, functions that are aggressively inefficient could be penalized even if they are otherwise correct. In general, functions ought to terminate in under 5 seconds for any reasonable input.

Your submission for the project will be evaluated for code style. This will not be strict – for example, one bad indent or one subjective variable name are hardly a problem. However, if your code seems careless or confusing, or if no significant effort was made to document the code, points will be deducted.

If you are confused about the style guide, please talk with a TA.

Formatting Tip: IntelliJ IDEA can be used to auto-indent your code! This option is available under the 'Code >> Reformat Code' menu item.

Abstract

Decision making requires data which often can be obtained from observation. For example, determining when a restaurant is busy is possible by collecting data on the times that people arrive and depart. Based on this information, the owner of the restaurant can determine how many workers should be scheduled at different times during the day.

However, there are situations where information gathering like this may be very difficult or impractical. To assist in these kinds of decisions, computer modeling can be used. Using an object oriented language (like Java) together with basic data structures that we have covered so far in class, we will be applying a technique known as *discrete event simulation* to gather data and create a report. Discrete event simulation is based on the creation of objects that represent each of the components in the system. Time is modeled using an agenda (or priority queue) where future events are scheduled in chronological order. Only time steps where an event occurs are considered by the simulation program, which is quite efficient.

Introduction

Suppose the popularity of a bus route has increased and now the customers are complaining about longer waits and full buses. In order to keep ridership up, the bus company has decided to run a simulation to help figure out whether they should **add more buses along this route** or **buy larger buses for the route**.

The company wants to minimize the *average travel time* (wait time + ride time). However, they also want to maximize the *Passenger Miles Per Gallon* (PMPG). The PMPG is calculated by multiplying the *Miles Per Gallon* of the bus by the *Average Occupancy* of the vehicle. Normal Buses run at 6MPG and hold 40 passengers, while Extended Buses run at 4MPG and hold 60 passengers. In the case that both these buses are run at full capacity all the time they have the same PMPG; however, in practice they may not be run at full capacity (to reduce wait time). You are tasked to create the simulation and return a report with your findings.

The Car Wash example that we "acted out" in lecture has many parallels to this project. You may find it helpful to use some of the code for the Car Wash example in your simulation project, and that is O.K. Specifically, you should use the priority queue (PQ.java) as-is. Be sure to credit any code "borrowed" from the posted lecture examples.

Setup

We will assume the following adjustable system parameters:

- A variable number of buses.
- Bus lengths, i.e. you may have a mix of larger and normal sized buses.

- A variable average inter-arrival rate of passengers at each stop (this will be referred to as the *load*).

We will assume the following constraints:

- Passengers will be passive, but they will contain data such as:
  - Time they arrived at the stop
  - Destination/stop they want to go to
  - Direction of the stop they want to go to (eastbound or westbound)
- Arrival of a passenger at a stop will be provided by an arrival class that will generate arrivals for all the stops (determined statistically, see below)
- Wait/travel time will be determined by arrival time and passenger count (see below)
- If a passenger cannot get onto the bus (when the bus is full) they must wait for the next one

Simulation of the system is achieved by creating classes to model the **behavior of each element** of the system as well as **each activity (event) that occur**s in the system. There is NOT an overall controller that calls for actions to happen at particular times except for a main driver loop that runs queued events until time runs out. Each element in the system models its own behavior and interactions with other elements in the system. Each NON-passive element class has an `Event` class associated with it. The associated event defines a `run()` method (as discussed in lecture) that simulates the specific behavior of that event.

Some of the classes that we will want are:

- Bus Stops (`Stop`) - contains `Passenger` queues
- Agenda (`PQ`)
- `Event` - an interface for all our events
- `BusEvent` - occurs each time a `Bus` arrives at a stop
- `PassengerEvent` - one occurs for each `Passenger` arriving at a stop

Let's look at each of these components, and see what they should contain. Please note that you may need to include other information, but what is given here is considered fundamental.

PassengerEvent

This is instantiable once for each `Passenger` creation event. One `PassengerEvent` will be made for each stop, allowing you to have some stops that are more popular than others. This will implement the `Event` interface similar to how the `CarMaker` class was implemented as described in lecture. `PassengerEvent` will reschedule itself (using the agenda), create a `Passenger`, decide where they want to go on the route, which direction that is from the current stop, and place the `Passenger` in the appropriate queue at the current stop.

Stop

This is instantiable once for each stop on the route. Each `Stop` will have two queues associated with it (one for eastbound passengers and one for westbound passengers), and a name to designate which stop it is on the route.

> **Info:** `University Ave and 27th Street SE` and `Union Depot` each only need to have one queue, since passengers who arrive at those stops cannot go further east or west, respectively.

There are exactly 10 bus stops:

- University Ave and 27th Street SE
- Raymond Ave Station
- University Ave and Fairview Ave
- University Ave and Snelling Ave
- University Ave and Lexington Parkway
- University Ave and Dale Street
- University Ave and Marion Street
- Cedar Street and 5th Street
- Minnesota Street and 4th Street
- Union Depot

BusEvent

This implements the `Event` interface. A `BusEvent` is created for every arrival of a bus at a stop. When a bus arrives at a stop, the `BusEvent` causes the bus associated with it to look at its passenger list to see if there are passengers that wish to exit the bus. If there are, the bus removes those passengers. The `BusEvent` will then look at passengers at the `Stop` that want to go the direction the bus is going and put as many of them as possible on itself. Finally, the `BusEvent` will create a new `BusEvent` and schedule it (via the agenda) for the arrival at the next stop at a time in the future depending on the number of passengers that got off and got on. If the `Bus` has reached the last stop on either side, it will start going the other direction. For example, if an eastbound bus arrives at the Union Depot Stop, it will then leave the Union Depot Stop going westbound.

PQ

This is the priority queue (likely to be called the agenda within your code). This code is provided. You will need one instance of it. This one instance will be used to schedule all events for your simulation.

Statistics

You will want to keep track of relevant `Statistics` such as:

- How full each bus is
- The maximum time a passenger spends waiting at a stop
- The maximum queue length at a stop
- etc.

Randomness

You will need to use a random distribution to model the arrival of Passengers at each stop. This is determined by method calls made in the `run()` method for the `PassengerEvent`. You will also need to randomly generate which stop each `Passenger` wants to go to.

For the arrival of passengers at a stop, you should start with an average inter-arrival rate of 1 Passenger every 120 seconds. But, you will want to run your model with both higher and lower demand by decreasing and increasing this inter-arrival time. To more realistically represent the pattern of arrival of Passengers, we will introduce some randomness according to the table below (calculations shown using a 120 second inter-arrival time as an example):

- 10% of the time: 75% above average arrival interval $(120 + 0.75 \times 120)$
- 15% of the time: 50% above average arrival interval $(120 + 0.50 \times 120)$
- 20% of the time: 20% above average arrival interval $(120 + 0.20 \times 120)$
- 10% of the time: right at average arrival interval (120)
- 20% of the time: 20% below average arrival interval $(120 - 0.20 \times 120)$
- 15% of the time: 50% below average arrival interval $(120 - 0.50 \times 120)$
- 10% of the time: 75% below average arrival interval $(120 - 0.75 \times 120)$

Downtown stops (listed below) are more popular than others, therefore, at these stops passengers should arrive 50 percent more frequently than at normal stops. They are also *two* times as likely to be a destination for a passenger than another stop.

Of the 10 stops, there are 3 downtown stops (Cedar Street and 5th Street, Minnesota Street and 4th Street, and Union Depot), and 7 normal stops (those not listed in the downtown category). If we sum the weight (likelihood) of each stop, we get 13 $(2 \times 3 + 1 \times 7)$. This means each downtown stop has a 2/13 chance of being chosen, and each other stop has a 1/13 chance of being chosen.

Determining processing time

To determine the amount of time a bus will take between stops, we use these rules:

- A bus will take 3 minutes (180 seconds) between each stop
- A bus will wait for at least 15 seconds at each stop
- It takes a passenger 2 seconds to get off and 3 second to get on

Therefore, a bus will take at least 3 minutes and 15 seconds to get from one stop to another. If the time it takes passengers to get on and get off is larger than 15 seconds, it will take 3 minutes plus the amount of time it took for passengers to get on and off. Even though each stop isn't the same distance from the next one, the times between each stop has been simplified to make the calculation easier.

Assumptions

- Assume that passenger arrival distribution and destination stop will use the distributions given.
- Assume that the maximum number of buses is 18 (one for each stop going each direction)
  - Buses at `University Ave and 27th Street SE` and `Union Depot` cannot go any further east and west respectively. Consequently, there can only be at most 18 buses running at a given time.
- Use the rules above for calculating the amount of time between each stop for a train.

Variables

- Load: the average inter-arrival rate of passengers
- Number of buses (1-18, this can also be considered the frequency of buses)
- Bus Size

What to do

First, get the simulation system to work, but start small and verify each feature works before proceeding to the next one. Then, produce a detailed report that gives a convincing presentation for what the bus company needs to do to satisfy its customers while maximizing their fuel economy. In order to provide a convincing argument, you will need to collect good statistics, verify them, and present them in a clear manner.

Something to watch for and correct when you run your simulations....If left unchecked, the busses will tend to clump together and even pass each other in situations where there are not a lot of busses in the system. This is a problem because busses that are clumped together will not serve evenly the potential riders at some stops. You should check to see if clumping is happening. If it is, try to incorporate something into the Bus class to keep a bus from getting to close to the bus in front of it. In the real world, a bus will hang out at a stop if the driver can see another bus

ahead of it less than two stops away. We recommend that you first determine whether clumping is happening and prove it with some output from your simulator. Then, build in some code into your Bus class to slow down busses that are getting too close to the bus ahead of it. Note that this will only make sense when the number of busses is fewer than about one half the number of stops in each direction.

The problem is purposely vague so that you can decide what statistics you need to gather and what simulation tests you should run. But, you need to present enough information so that the bus company can decide how many buses and what bus size it needs for various loads. (Keep in mind that the bus company will want statistics for the busy times (lunch and rush hour) and well as off-peak times such as weekends and nights. A significant portion of the grading will be based on the write-up of your results which will include your simulation results presented in a useful format, conclusions and recommendations. This means that you will need to have a working simulator in order to have results to write about.

The Write-up

The write-up must be submitted as a `.pdf` document along with the rest of your source code. 25% of your project grade will be somehow connected to the write-up. When deciding on statistics to maintain and the simulations to run, think about the need to support your conclusions in your write-up. The write-up should be in the form of a report–such as a consulting report–that you might submit to the bus company. The report should include data (such as average travel time and passenger miles per gallon) summarized in a concise and readable fashion along with specific data that proves that the simulation results are correct. Please be sure that the report is concise.

For example, do the numbers make sense? Are your simulation runs at equilibrium? (That is, as you run the same model with the same parameters for longer times, do the stats tend to stay constant?) Certainly, some statistics are more difficult to properly calculate than others. So, don't try to do it all at once. Get some stats going, and verify them, before moving on.

In order for this to be a successful project, you need to start NOW, and have your simulator working about 5-7 days before the project is due. That will give you enough time to run all the tests, make modifications and produce the report. Remember, the bus company will be interested in how many buses and what size buses they will need for various loads, average number of passengers at each stop, the average and maximum passenger wait times, and average travel time. You will need to run your simulation multiple times. It will take some thinking to know what runs to make, how to effectively present your findings and deal with, or account for, bus clumping and passing.

The Readme

A common expectation of large software projects is that they provide a readme file to familiarize the user with the project. You must provide a readme file along with your project; it needs to contain (at-least) the following items, which are often included in readme files for other projects:

- The project name and author information.
- Instructions for running your project on a new machine.
- An overview of the project organization and hierarchy.
- The data structures and algorithms used in the project, and why they are a good choice.
- Any known bugs or issues associated with the project.

As this project is largely left up to student design, the intent of this readme file is to familiarize your TAs with your project. A good readme file will make grading the project easier, and be the capstone on your submission.