Air Travel Simulation

Version of 29 March; future revisions will clarify the description in reaction to student questions.

Unlike the previous simulation we considered which were based on mathematics of differential equations, Lotka-Volterra, this simulation is simple: the state of an air travel system changes from one minute to the next by events. The simulation we plan here is simplified compared to real life. There are no timetables, no worries about fuel, no failures of equipment, no strikes by employees, and so on.

What are events? Passengers arriving, jets taking off, jets landing, passengers transferring themselves from one jet to another. In the simulation, time is measured in minutes. Think of the simulation starting at minute number zero, and then time increase to larger minute numbers. When the simulation goes from minute n to minute n+1, numerous events can occur. Maybe one passenger arrives at the airport ready to travel, and during the same minute a jet lands at a different airport. The simulation has to keep track of all these things. Potentially, each event and the time of the event is written to a file, or perhaps displayed graphically on a screen during the simulation. That's not your job: your task instead is just to design object-oriented software that will do the simulation.

What would such a simulation be useful for? The usefulness is to run experiments and get statistics on the simulated travel time for passengers under different scenarios of how the travel system is configured, how frequently passengers use the system, how much extra delay is due to congestion and waiting, and so on.

Inputs

The simulation depends on some inputs. These inputs include

- A map (in the sense of a road map, not a Java or Kotlin map) of 20 cities with air corridors between them; let us call the cities A, B, C, ..., S, T. The map shows that not all pairs of cities have an air corridor between them: maybe there is no corridor between city C and city R. However, the map is guaranteed to have indirect routes between any pair, so getting from C to R may be a path, like C to J, then J to R.
- A table of the minimum time it takes for a jet to go from one city to another: it is a number in the range [10,80] minutes, and it depends on the specific pair. So, the table may say it takes 10 minutes to fly from C to J, and 55 minutes to fly from J to R.
- A listing of jet types and their passenger capacity. For instance, maybe there could be two types of jet, one with 60 passengers maximum and another with 150 passengers maximum.
- A table of the specific jets in the air travel simulation is another input: this table has as many rows as there are jets in the system. Each table entry has the jet's type (so we know it's passenger capacity), a serial number, and a route: for each jet, there is a fixed route that it perpetually

follows. It could be that jet number 3127 continually flies from A to B, then B to C, then C to A. Each jet in the table thus follows a *cyclic* route, so it is always going through the same path of cities.

Initially in the simulation, no jet is flying, they are all at terminals, at random locations (but each jet is at a city in its cyclic route). In addition to the data described above, that are inputs, there are some inputs that are functions (instead of tables, maps or lists). These input functions, described below, are things that the simulator should call when it runs.

- Each city has a function `newpassenger(t)` which returns a list of passengers who arrive to the airport at minute t. The returned list could be empty (if no passengers arrive at minute t). If it's clearer notation, you can make parameters `newpassenger(t,x)` to mean the list of new passengers at time t at city x.
- Another function `nextdest(p)` returns, for passenger p, the choice of the next destination in the air travel system.

Passengers

Let us consider the properties and behavior of passengers. When a passenger p arrives to the airport at some city, say city J, that passenger has a ticket to travel from J to a destination city, say D. Direct flights from J to D may not exist, so perhaps the passenger has to take an indirect (multi-city) route. The `nextdest(p)` function chooses which city is the next goal in going to the destination city D. We assume that `nextdest(p)` is a *smart* function: if there is a direct flight to D, then `nextdest(p)` will return D as the place to travel next. Otherwise, `nextdest(p)` will choose another city that is closer to the final destination. Thus the simulation is more like bus travel than real-world, current air travel. When a passenger arrives to the final destination, then that passenger exits the air travel simulation (disappears); the simulator should produce an output which indicates that the passenger left the simulation. Each passenger has a unique identifier so it should be possible to calculate how long a particular passenger was in the simulation.

Jets and Cities

Unlike the real world, jets wait until it is at least half full of passengers. So, if a jet is going from city K to city M and has capacity of 60 passengers, the jet won't request to taxi/takeoff until there are at minimum 30 passengers on board. We trust that that `newpassenger(t)` function provides plenty of passengers so that infinite waiting won't happen.

Each city's airport has limited facilities for landing, takeoff and taxi from the runway to the terminal. It any time in the simulation there can be only one jet taking off or landing at each city. (Of course, in different cities, jets can takeoff and land at the same time.) Jets take two minutes to takeoff and two minutes to land. This means that if two jets would like to take off from city E at the same minute n, then one can take off at minute n, and the other has to wait for minute n+2. Such extra time for landing and takeoff is not included in the input table of travel times between cities.

The taxi travel from runway to terminal is also constrained to be one at a time. If a jet that has landed needs to taxi to the terminal, and at the same time two other jets want to taxi to the runway, then two of them have to wait. The time for taxi from runway to terminal, or from terminal to runway, is five minutes. Obviously this is totally unrealistic because in real live, taxi from runway to terminal takes, like, forever.

Even at the same city, say city K, there can be more than one jet moving at the same time. For example, at minute n perhaps one jet starts landing (finishing the landing at minute n+2) and another jet starts to taxi from the terminal to the runway, finishing at time n+5. That would imply that the landing jet has to wait until at least time n+5 before it can taxi to the terminal: there is only one taxi lane used for both directions of taxi travel.

Interactions

There are two interactions that take time, passengers with checkin counters and jets with air traffic control. When a passenger arrives at a city that is not the final destination, then that passenger needs to check in. Here is the protocol for passenger p:

1. passenger presents ticket
2. checkin counter says OK
3. passenger (use `nextdest(p)`) selects another city for the next place to go
4. checkin counter says OK, and passenger walks to the waiting area for the chosen city (and if there is a jet with empty seats already there the passenger goes on board)

This interaction between passenger and checkin counter takes three minutes, which includes walking time to the waiting area/jet.

Another interaction is between jet and traffic control:

1. jet informs traffic control of request to land
2. traffic control says ok, if the runway is not in use, or
3. traffic control rejects the request, telling jet to request again later

Similarly there is an interaction for runway to terminal travel:

1. jet requests traffic control permission to taxi from runway to terminal
2. traffic control says ok, if the taxi lane is not in use, or
3. traffic control rejects the request, telling jet to request again later

The departure interaction:

1. jet informs traffic control request to takeoff (jet in line, near runway)
2. traffic control says ok, if the runway is not in use, or
3. traffic control rejects the request, telling jet to request again later

But getting the runway needs a different interaction

1. jet requests from traffic control permission to taxi from terminal to runway
2. traffic control says ok, if the taxi lane is not in use, or
3. traffic control rejects the request, telling jet to request again later

These interactions betweeen jet and traffic control take no extra time, but of course the events of landing, going from and to terminal or runway, they do take time to complete.

Assignment

Create and describe an object-oriented design, including classes, interfaces, and methods, for this air travel simulation. You should write something that could be handed off to a competent programmer of Kotlin to write the actual code.

This should be like an essay or report, but include diagrams of the classes and possibly interactions. Make a convincing case that you have *thought through* how such a simulation would be implemented (programmed with code). One test of your answer is this: would there be any surprises in programming this design, like unexpected needs for other classes, objects and interfaces that are not written in the design?

Evaluation

The assignment is due each Friday for the remainder of the semester. After submission (on ICON) a brief evaluation, through a comment, will be written on ICON - that may take a few days to appear, because evaluation takes some time. If the first submission is perfect, you will get full credit on this assignment.

If the first submission is not perfect, you can resubmit a revised design for the next (or later) Friday. Here is the rubric for scoring:

150 points - satisfactory design on first attempt

130 points - satisfactory design after two submissions

100 points - satisfactory design after three submissions

60 points - satisfactory design after four submissions

partial credit - unsatisfactory design and no more submissions

According to these rules, you can wait and submit something for this assignment on the last Friday of the semester (but then you get no second chance). Waiting before you submit to be confident your design is complete has some value, since submitting an unsatisfactory design will get comments and you would have to submit a revised version (getting a reduced score)

Feedback

If your design is unsatisfactory, there will be brief comments on ICON (which as a pitifully limited area for commenting). Typical comments could be like this:

- Your design puts everything into one main class, using arrays to model all the entities of the simulation. That is not an object-oriented design.
- What data structure(s) model the map of cities and air corridors?
- The report says that the simulation does events at each time unit. What class/object/method increases the time during the simulation?
- Where in your design is the data structure (class) that is used to determine what are events at each time unit?
- Your design has 20 classes, one for each city. Does each class have to be written as new code? Won't that result in lots of duplicated code?