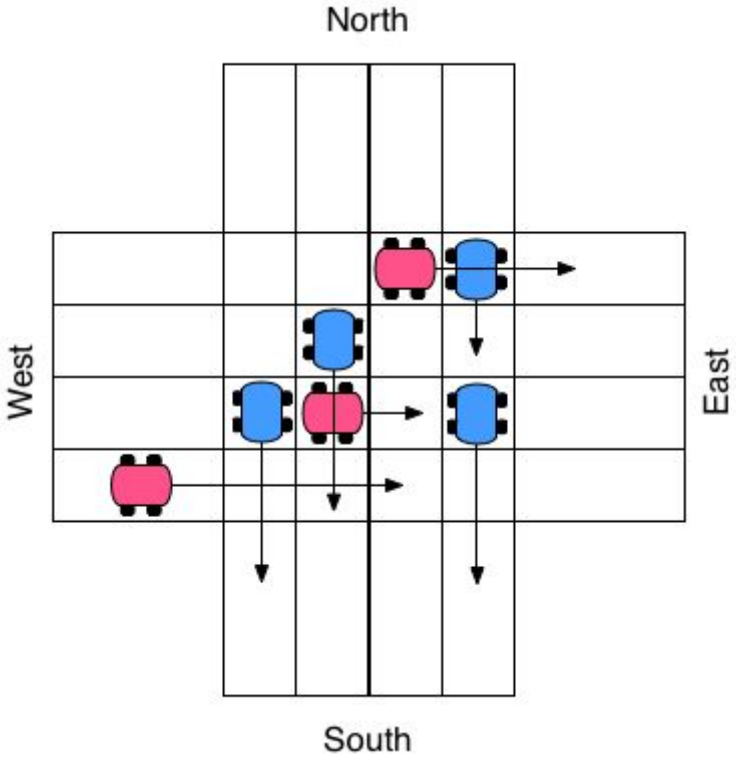
AP(IT) Coursework 2018

Introduction

If we are to believe the hype, driverless cars will soon be the norm. Glasgow City Council (GCC) has decided to plan ahead and attempt to model the effect this will have at the many intersections in the city centre. In particular, once cars don't have to rely on error-prone human drivers, they feel that traffic lights can be removed, leaving cars free to merge safely across one another (see Figure).

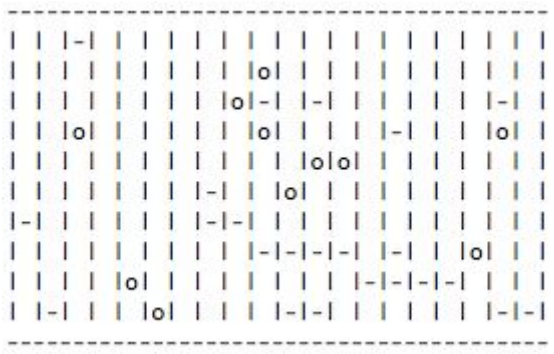


You are part of the team tasked with modelling this scenario and have been asked to produce some of the code infrastructure required for the model, as well as an example model for a single junction.

Specifications - level 1

- You should model the intersection as a N x M grid (N does not necessarily equal M)..i.e. N East-West lanes and M North-South lanes.
- Vehicles move either from North to South or from West to East.
- Only one vehicle can be in any one grid square. You should enforce this with a ReentrantLock (and a condition).
- If a vehicle wants to move into an occupied position it should stop and wait for the position to become free.

- If uninterrupted, vehicles should move at their own constant speed (you should model this by having a fixed delay once entering a position before attempting to move into the next one). Each vehicle should have a randomly allocated constant speed.
- Vehicles should be randomly generated to start in either the first row (North-South vehicles) or first column (East-West vehicles) and each should exist on its own thread. When a vehicle gets to the opposite edge it 'falls off'.
- A separate object (on its own thread) should draw the grid (including its contents) every 20 milliseconds (see example below)
- North to South vehicles should be drawn with a different character than West to East ones.
- The simulator should stop after the grid has been drawn z (e.g. 2000) times.
- In your main method, you should make an example grid with ten rows and twenty columns (ten and twenty lane roads are a bit unrealistic but it makes the plotting more fun).



Example screenshot. '-' vehicles are going from left to right (West to East), 'o' from top to bottom (North to South)

Specifications - level 2

- Different traffic generators should be able to be applied to different rows / columns. Extend your code to allow for this, implementing at least two different generators (perhaps they generate traffic with a different distribution of times, or traffic with a different distribution of speeds). Note that you're not being tested here on your ability to implement different random generators but instead on your ability to make modular code that allows for different traffic generator classes that adhere to some common design.
- Extend your code to have lanes going in either direction. E.g. create a 10 row by 20 column demo where the top 5 rows go from left to right and the bottom five from right to left. Similarly, the first ten columns go from top to bottom and the second ten from bottom to top. Marks will be awarded here for neatness of the solution (i.e. I can imagine a solution that has lots and lots of 'if' statements somewhere and that is not what I'm after).
- Create the capability for each generator to report the time it takes all vehicles it generates to travel through the grid. Once the simulator has finished, a statistics class should be able to create a report that gives the maximum, minimum, mean and variance times for each generator.

What to submit

- Your code. A class called APSpec1.java should include the main demonstrating the functionality required for spec 1. If you attempt spec 2, a separate main method is (in APSpec2.java) is required. Both main methods should call the same classes...i.e. I don't want to see completely different classes for spec 2 than for spec 1 (although you might need a one or two additional ones).
- A report describing your submission. The document *must* have the following headings (and no others):
    - Assumptions - what have you assumed that is missing in these specs.
    - Class design - description of each class (can be commented UML if you like, or plain text if you prefer).
    - Testing - how have you tested your code? The bar is a bit higher than in Programming last semester.
    - Questions (see below)

Questions

Under the 'questions' heading in your report, answer the following questions. Clearly number your answers:

1. Critique: one of the key characteristics of good object oriented programming is the ability to make code easily extensible. Describe an extension of the system that your current design could not handle (you may not use the example given in point 3 below)
2. As 1., but for an extension that it can handle (you may not use the example given in point 3 below)
3. One possible extension is the stringing together of multiple intersections to produce a model of a larger city area. Describe how you would do this, giving details of any changes in your classes that would be required. You do not have to implement these changes.
4. Reflection: write a short summary of your experience of this project. How did you tackle the problem, how might you do things differently if you did it again?

Suggestions

- Good code should be extendable. Try and avoid coding something that is too restrictive. Remember that you're being asked to provide components to go into the simulator. The particular system you simulate is therefore just an example of the use of your classes.
- Comments are important. I don't have a preference for style, but make sure enough details are present someone else to understand your code.
- Develop incrementally. I suggest solving the 'only one vehicle per intersection' problem first. This is (I think) the hardest bit!

Marking scheme

Your final mark will be a band (A1-H).
It will be computed by combining marks from the following categories:

- Functionality [50%; comprising 35% for spec 1 and 15% for spec 2]
    - A fully functioning implementation of specification 1 can achieve 35%. A fully functioning spec 2 can achieve a further 15%.
- Design [30%]: marks awarded for a design that is extendable, clear, and adheres to sound object oriented philosophy. Note: an implementation that is functional but not extendable can still get full marks for spec 1 functionality but it will be severely penalised for design. Note 2: you can achieve full marks for design regardless of whether or not you attempt spec 2.
- Documentation [20%] Marks awarded for the categories requested above. Note that the mark here will not include the design itself, but marks will be awarded for the presentation of the design. Code comments will be assessed as part of this category.