CS261 Group 29 Planning and Design

Add your names here

January 2025

Contents

1 Given Requirements for document

You are also required to submit a Planning and Design Document. This should address the design of your software solution together with a detailed plan as to how (and when) this design is to be implemented. There are a number of things to consider in the design of software systems, including:

- Extensibility – how you can extend your solution given increasing demands;
- Robustness – how your solution will tolerate unpredictable or invalid input;
- Reliability – how the system will perform under everyday conditions;
- Correctness – how accurately your solution meets the requirements of the customer;
- Compatibility and portability – how easy your proposed system is to install and execute;
- Modularity and reuse – how well your system is divided into independent components and whether you have reused existing code;
- Security – whether your system can withstand hostile acts and influences;
- Fault-tolerance – whether your system can withstand and recover from component failure.

This list is not exhaustive and there will be other concerns which your group will want to address. You will also want to adopt trusted design patterns or design methodologies to provide a template for the actual design of your system. No one method will be favoured (by the assessors) over another, but you will be graded on the process of selecting an appropriate methodology and your use of it. Each group will have to submit these two short reports on tabula on Monday 3rd February 2025. Each member of the team should submit these reports along with a team contribution form for the first half of the project (see below).

2 Team Planning

2.1 Time Management

In our meetings we hae discussed how we are going to manage our time and have agreed to the following deadlines:

- Requirement Analysis: 22/01/2025
- Planning and Design: 29/01/2025
- Back-End Development and Testing: 26/02/2025
- Front-End Development and Testing: 26/02/2025
- Dragons' Den Video: 1/03/2025
- Final Report: 10/03/2025

Below you can see a Gantt chart of our planned work schedule.

Task	Start	End
Group Software Engineering Project CS261	6/1/25	10/3/25
Initial Meetings	6/1/25	13/1/25
Requirement Analysis	13/1/25	22/1/25
Planning and Design	22/1/25	29/1/25
Back-End Development and Testing	29/1/25	26/2/25
Front-End Development and Testing	5/2/25	26/2/25
Dragons Den Video Presantation	26/2/25	1/3/25
Final Report	1/3/25	10/3/25

Figure 1: Gantt Chart of Planned Work Schedule

2.2 Risk Assessment and Management

We have identified the following risks and have agreed on the following mitigation strategies:

2.2.1 Technology Limitations

- Risk description: Team/Team members may be unfamiliar to some tool, libraries, or frameworks, which may cause delays or reduced performance.
- Risk Level: Tolerable

- Risk Likelihood: Moderate
- Mitigation Strategy: Assign tasks to team members based on their expertise in relevant technologies while ensuring everyone is involved in meaningful roles to maintain prductivity and foster teamwork.

2.2.2 Rollback Challenges

- Risk description: Lack of a version control system could prevent from rolling back to the software's last stable state in case of errors
- Risk Level: Catastrophic
- Risk Likelihood: Low
- Mitigation Strategy: Utilise github to always maintain a stable version of the software and updating it when being sure that the changes will not affect its usability.

2.2.3 Testing Risks

- Risk description: Insufficient testing may reduce confidence in the software
- Risk Level: Serious
- Risk Likelihood: Moderate
- Mitigation Strategy: Unit tests will be designed to test the software to make sure that it is working properly.

2.2.4 Time Management

- Risk description: Underestimating task duration or improper prioritization might result in delayed work.
- Risk Level: Serious
- Risk Likelihood: Low
- Mitigation Strategy: The team is meeting in regular intervals to ensure work efficiency and mitigate time related risks.

2.2.5 Requirement Misalignment

- Risk description: During the development of the software, the end product might not be the same as the one describe in the delivrables due to unforeseen circumstances
- Risk Level: Catastrophic
- Risk Likelihood: Low
- Mitigation Strategy: Ensure constant internal communication between the team.

2.2.6 Organisational Risks

- Risk description: Uneven distribution of workload or misscommunication may lead to an uncomplete project and delayed work.
- Risk Level: Serious
- Risk Likelihood: Low
- Mitigation Strategy: The team is meeting in regular intervals to ensure work efficiency and mitigate time related risks.

2.2.7 Team Membr MIA

- Risk description: Team member is not able to complete their amount of work due to unforeseen circumstances, thus delaying work.
- Risk Level: Serious
- Risk Likelihood: Moderate
- Mitigation Strategy: Team analyses the remaining work from missing member and prioritises and reallocates tasks based on the analysis.

3 Front-End

4 Back-End

For the back-end, we have decided to implement it in Python due to the whole group being familiar with the language, and to make interfacing between the front-end and back-end simple.

4.1 Simulation

We plan to use objects to simulate the junction configurations and calculate the junction efficiency metrics and overall scores, with the structure of classes being as follows:

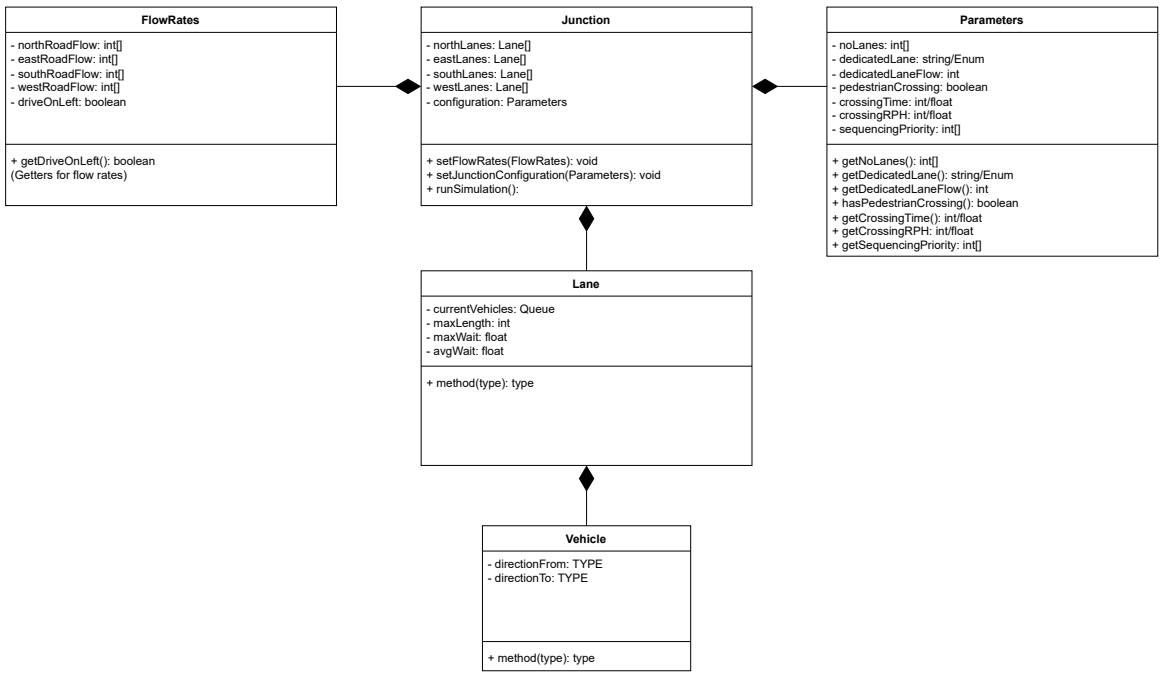

Figure 2: Junction Simulation Class Diagram

The Junction class contains Lane objects representing each lane of a road entering the junction, grouped together in arrays based on the direction they're arriving at the junction from. Two functions will be called by the front-end to set up the simulation: setFlowRates to set the flow rates from each direction to the other directions, passed in the form of a FlowRates object, and setJunctionConfiguration to set the specific settings of the junction (e.g. is there a left turn lane, how many lanes there are on each incoming road) passed in the form of a Parameters object. For each junction configuration generated by the user, a Junction object will be created and be passed a Parameters object corresponding to those configuration settings. We decided to use objects for passing the data instead of passing each setting as an individual parameter (setJunctionConfiguration) or passing the flow rates as a 2D-array (setFlowRates)

since objects would be easier to work with (for example, no scope for miscommunication about what the indices represent) and make it easier to extend the capabilities of the software in the future.

Each Lane object contains a queue of Vehicle objects, representing the vehicles in that lane waiting to transit the junction. Each Lane will calculate its own maximum wait time (maxWait), average wait time (avgWait) and maximum queue length (maxLength), which will be compared with the same values of the other Lanes in its direction by the Junction object, in order to get the three values for the direction.

4.2 Interfacing with Frontend

We will use (Insert Library like flask) to create a basic web server, we will define our API using the OpenAPI schema. Defining an API using this allows us to create a client for our frontend application to use very easily making sure that we have a consistent usage and implementation of our API across both components of the system.

The frontend will have some basic validation logic for input data however we will validate data on the backend as well as providing an easy to use endpoint to check that a set of data is in fact valid.

Usage of the API would allow us to add usage monitoring and authentication in the future, this is not a concern if the system is to be limited in scope and adoption but if in the future it was to expand and be provided as service to others then this would be useful.

4.3 Validation and Error handling

There are a large variety of parameters that can be inputted into the system, it is important that they are properly validated, this will be achieved through functions that check each of the parameters is in the required range and that there are no conflicts between certain parameters. For example the sum of each direction's output flows must be equal to the inflow as well as the requirement for each flow to be greater than or equal to 0.

We will use automated testing to check that the system validates sets of input parameters correctly on each build, any changes to the validation code will be be then checked automatically that they produce the correct result.

As we will use an API to interface between the frontend and backend we can indicate the validity of a request using https status codes, using code '400 - Bad Request' with a description of the error will show what the issue is.

5 Deployment

One of the key requirements of the system is to be portable and easy to deploy, to facilitate this we will containerise the backend using docker/podman containers. Planning for this from the start will allow us to easily meet the future needs of the client.

5.1 Modularity

Separation of the frontend and backend into separate services will allow us to more easily expand the system in the future. The backend simulation is connected to an API which can then be accessed using different applications, the frontend and backend can be developed independantly and can be swapped in and out as long as they maintain the API used to communicate between eachother.

Docker/Podman is an open standard and will allow for these containers to be ran on any platform with a container engine.

5.2 Scalability and Fault tolerance

Usage of containerisation in this manner would allow us to scale the system as the needs of it grow, if for example we have multiple applications taking data from the backend at a time we can run multiple instances of the backend and use a load balancer to distribute client calls between each instance. If the client requires a higer level of uptime than a single instance allows, If they need fault tolerance than a single geographic location and isntance allows then this would be simple to achieve using a set of containers in a distributed network.

Developing inside a container allows for a consistent build and deployment environment everytime, if a client has an installation of docker/podman then they would be able to run the software just as they

would any other container. Containers are largely similar to bare metal performance so in small use cases (start of the program lifecycle) there is no significant impact by using containers.