LOW LEVEL DESIGN AND IMPLEMENTATION DOCUMENT

Smart Traffic Light Controller using Deep Reinforcement Learning

UE18CS390B – Capstone Project Phase – 2

Submitted by:

Krishna P Hegde	**PES1201801896**
Abhishek A	**PES1201801935**
Prathvik Nayak	**PES1201802006**
A Lakshmi Prasad	**PES1201802132**

Under the guidance of

Dr. Nagegowda K S
Associate Professor
PES University

August - December 2021

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING
FACULTY OF ENGINEERING
PES UNIVERSITY
(Established under Karnataka Act No. 16 of 2013)

TABLE OF CONTENTS

1. Introduction 4
    1.1 Overview 4
    1.2 Purpose 4
    1.3 Scope 4
2. Design Considerations, Assumptions and Dependencies 5
3. Design Description 5
    3.1 Master class diagram 7
    3.2 Traffic generation 7
        3.2.1 Description 7
        3.2.2 Class Diagram 8
            3.2.3.1 Traffic generator 8
            3.2.3.2 SUMO configuration 8
            3.2.3.3 episode_routes 8
            3.2.3.4 route_net 9
    3.3 Model creation 9
        3.3.1 Description 9
    3.4 Simulation and training 9
        3.4.1 Description 9
        3.4.2 Class Diagram 9
            3.4.2.1 Training_main 11
            3.4.2.2 Training_simulation 11
            3.4.2.3 memory 11
            3.4.2.4 TrafficGenerator 12
            3.4.2.5 TrainModel 12
            3.4.2.6 Utils 12
            3.4.2.7 Visualization 13
    3.5 Testing 14
        3.5.1 Description 14
        3.5.2 Class Diagram 15
            3.5.2.1 Testing_main 15
            3.5.2.2 TestModel 15
            3.5.2.3 Visualization 16

3.5.2.4 Simulation 16
3.5.2.5 Utils 16
3.6 Memory 18
3.6.1 Description 18
3.7 Utils 18
3.7.1 Description 18
3.8 Visualization and analysis study 18
3.8.1 Description 18
3.9 Packaging and deployment diagrams 19

Appendix A: Definitions, Acronyms and Abbreviations 20
Appendix B: References 20
Appendix C: Record of Change History 20
Appendix D: Traceability Matrix 21

1. Introduction

1.1. Overview

This document refers to the component level design process and approach. It gives a detailed description of each module, its logic and goes deep into each module specifications with the help of class diagrams.

1.2. Purpose

The goal of LLD or a low-level design document (LLDD) is to give the internal logical design of the actual program code. Low-level design is created based on the high-level design. LLD describes the class diagrams with the methods and relations between classes and program specs. It describes the modules so that the programmer can directly code the program from the document.

This document provides an overall description of different modules for Smart traffic light controller using deep reinforcement learning.

1.3. Scope

A good low-level design document makes the program easy to develop when proper analysis is utilized to create a low-level design document. The code can then be developed directly from the low-level design document with minimal debugging and testing. Other advantages include lower cost and easier maintenance.

Conventional traffic signals use a regular timer based approach which does not handle dynamic traffic conditions. Due to this, the vehicles have to wait for a long time even if the traffic density is less. In order to improve this we present a Reinforcement Learning approach to control traffic signals so as to reduce traffic congestion in crossroads and intersections.

The scope of this project is to initially simulate traffic in a 4 way intersection and then extend it for multiple intersections using a traffic simulator(SUMO). We use an agent(traffic lights system) to perceive the environment and train this agent to self learn using Reinforcement learning. In order to design a system based on the Reinforcement learning approach, we define the state representation, the action set, the reward function and the agent learning techniques involved. The learning mechanism used here is Deep Q-Learning, which is a combination of two aspects widely adopted in the field of reinforcement learning which are deep neural networks and Q-Learning.

2. Design Constraints, Assumptions, and Dependencies

- This simulation should be done on SUMO software.
- Since this is a simulation, it may not be very precise compared to real world traffic.
- The TraCI interface is used to interact with SUMO using python.
- Training the agent could be a very tedious job as we will need to run a large number of simulations which is time consuming.
- Extending to multiple intersections could be challenging and complex as we will need to have proper communications between agents at different intersections for smooth flow of traffic.
- Further implementation in the real world could be a complex process due to increase in resources such as sensors to scan the traffic.
- The sensors when implemented in the real world should be able to correctly produce the inputs so that the model can produce better outputs.

3. Design Description

- **Traffic generation**
  - This module is responsible for generating traffic and creating a route file for the SUMO simulator. Each call to this module will generate random traffic simulation using Weibull distribution.

- **Model creation**
  - This module is responsible for the creation of deep neural network models for training and predicting the outputs. We will be using 2 different models, Artificial neural networks and Convolutional neural networks.

- **Simulation and training**
  - The Simulation class handles the simulation. In particular, the function run allows the simulation of one episode. Also, other functions are used during run to interact with SUMO, for example: retrieving the state of the environment (get_state), set the next green light phase (_set_green_phase) or preprocess the data to train the neural network (_replay). Two files contain a slightly different Simulation class: training_simulation.py and testing_simulation.py. Which one is loaded depends if we are doing the training phase or the testing phase.

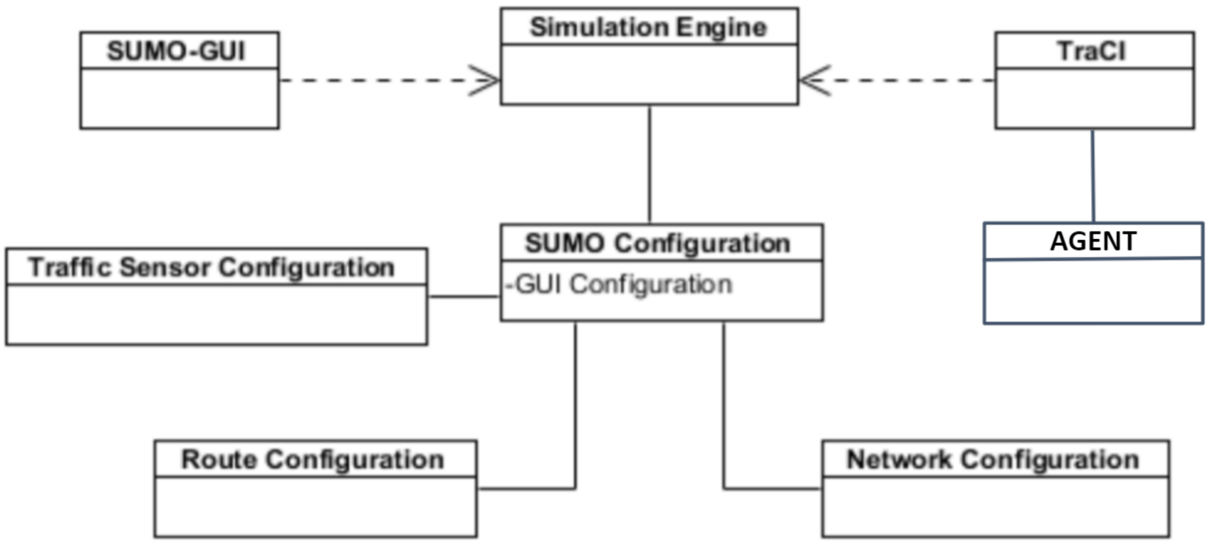
- **Memory**
    - This module handles the memorization for the experience replay mechanism. It has multiple functions to add samples to memory and retrieve a batch of samples for training.

- **Utils**
    - This file contains directory related functions such as loading models for testing and creating different versions of trained models, importing configuration settings, etc.

- **Testing**
    - This module is responsible for testing the model with a single test episode running a completely different and random simulation with emergency vehicles configured in such a way that they can ignore the red lights. We will be testing both the models(ANN and CNN) for multiple intersections as well.

- **Visualization and analysis study**
    - Once the testing is done we will be performing an analysis study to see the performance of the agent in both the models compared to normal traffic, to study the reward trend, and see the agent's behaviour in different levels of traffic.

3.1. Master Class Diagram

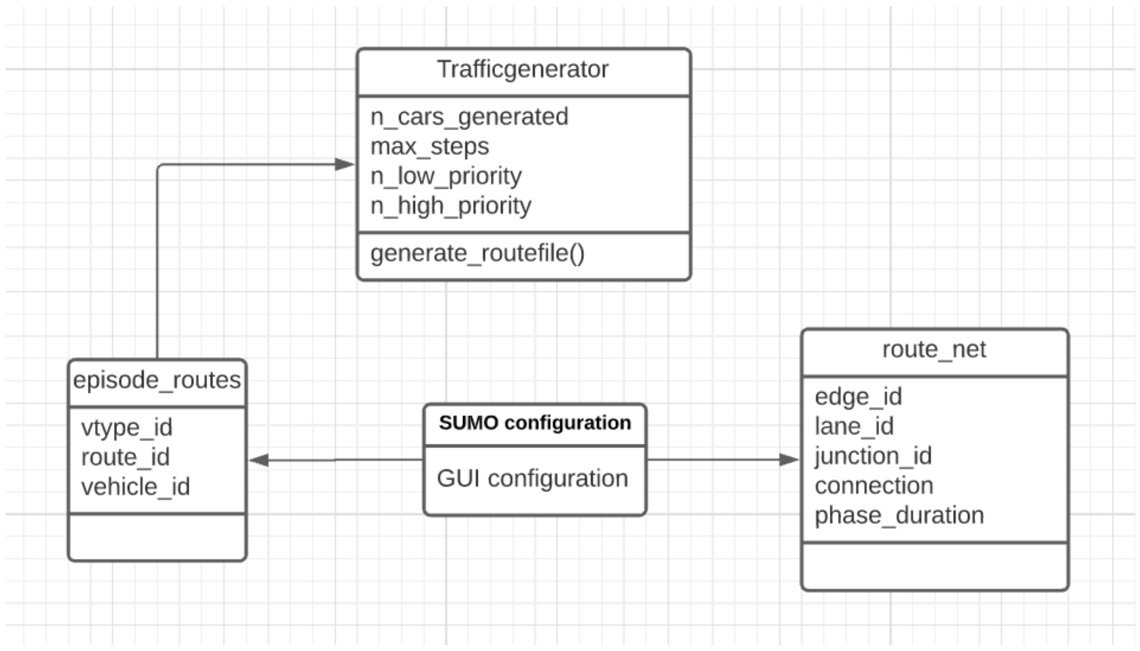


3.2. Traffic generation

3.2.1. Description

- The road network structure(environment.net.xml) and the initial traffic light sequence with durations for both single and double intersection environments was created using netedit software.
- The traffic on these networks was generated using a python script(generator.py).
- The generator.py file takes the parameter 'no of cars per episode' and generates traffic with all kinds of vehicles. You can even pass no of high priority and low priority vehicles.
- The generation of vehicles is distributed according to Weibull distribution.

3.2.2. Class Diagram



3.2.3.1. Trafficgenerator

- This class has attributes such as the number of cars that needed to be generated, max_steps, and no of high and low priority vehicles. Using these attributes it generates a route file using the generate_routefile() method.

  - ❖ generate_routefile()-This method creates a route file and generates different route id's for the simulation using Weibull distribution.

3.2.3.2. SUMO Configuration

- This class represents the main configuration file which makes use of both the route file and the net file and the SUMO gui to run the simulation.

3.2.3.3. episode_routes

- This class represents the file having all the attributes such as vehicle id, vehicle type, route id, etc. All these attributes are used to generate different routes.

3.2.3.4. route_net

- This class represents the file having all attributes such as edge id, lane id, junction id, etc. All these attributes are used to generate the network configuration and the different traffic light phases and its durations.

- **Data members of all classes**

Data Type	Data Name	Description
int	n_cars_generate d	no of cars to be generated in the simulation.
int	max_steps	the duration of each episode, with 1 step = 1 second (default duration in SUMO).
int	n_low_priority	no. of low priority vehicles.
int	n_high_priority	no. of high priority vehicles.
string	vtype_id	type of vehicle.
alphanumeric	vehicle_id	vehicle id for a particular type.
string	route_id	route id for each route.
alphanumeric	edge_id	edge id in the network.
alphanumeric	lane_id	lane id in the network.
string	junction_id	junction id in the network.
int	phase_duration	duration of each phase in a traffic light.

3.3. Model creation

3.3.1. Description

We will be creating two models.

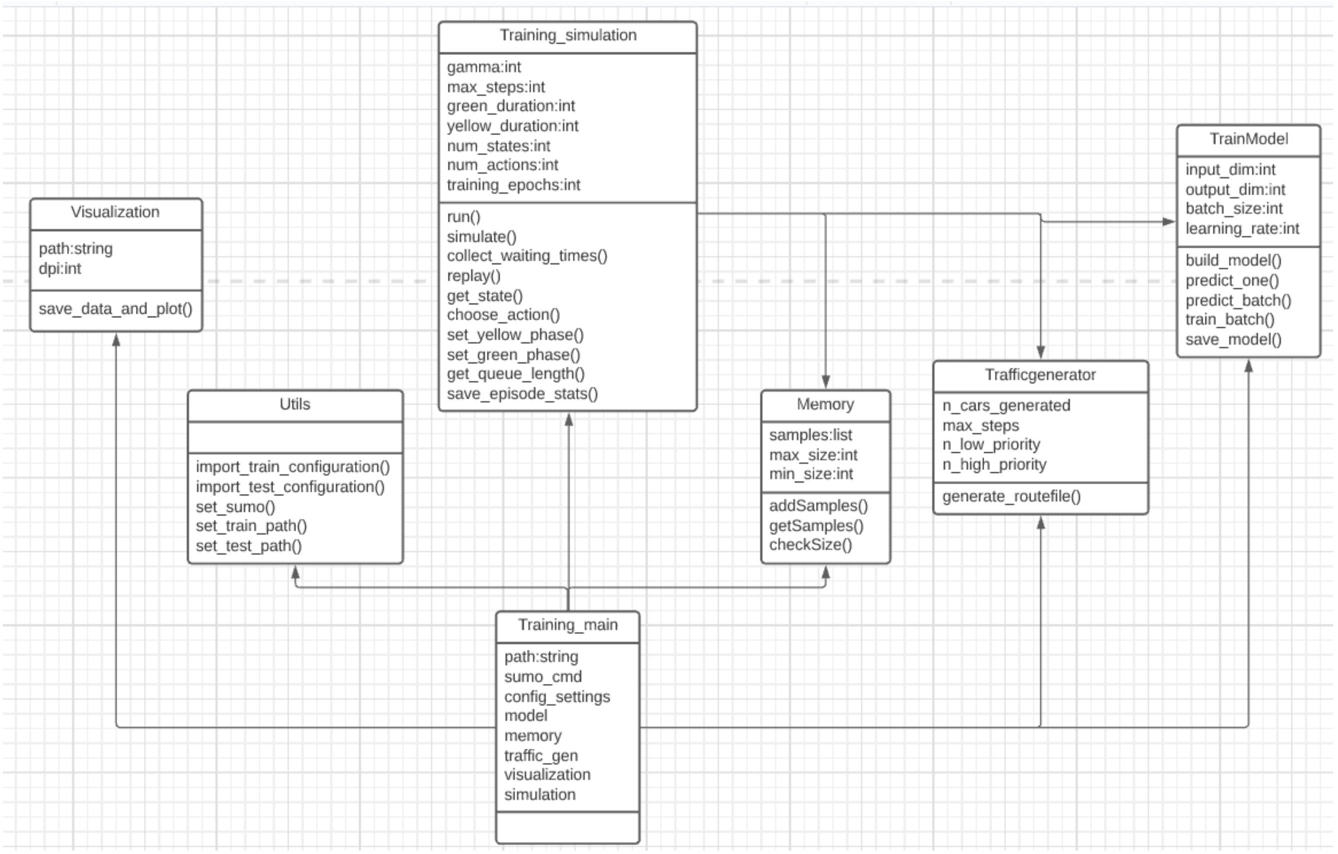
- The first model consists of a fully connected ANN with 80 neurons in the input later, 5 hidden layers with 400 neurons each with relu activation function and an output layer of 4 neurons with a linear activation function.
- For the second model we will be using a CNN, where the input will be two matrices (position and velocity) obtained using DTSE. We will be performing a few convolutions on these inputs to obtain essential features and finally obtain a linear output of size 4, each representing Q-value for each action.

3.4. Simulation and training

3.4.1. Description

- Here we'll be making use of traci api to connect to the simulations.
- From the simulations we will be retrieving multiple state spaces using the DTSE method and store them in memory.
- We will discretize the incoming lanes into cells to identify the presence or absence of vehicles in them.
- We will be training the model using an experience replay method to improve its efficiency and reduce correlation between consecutive simulation steps.
- The model will also be trained to select an action using the epsilon greedy.

3.4.2. Class diagram



3.4.2.1. Training_main

- This class represents the file training_main which is used to run the training simulation. It is associated with all other files/classes and has various attributes and objects.

3.4.2.2. Training_simulation

- This class represents the file training_simulation. It is responsible for running the simulations for training using various methods and attributes.

  - ❖ run() - Runs an episode of simulation, then starts a training session.
  - ❖ simulate() - Execute steps in sumo while gathering statistics.
  - ❖ collect_waiting_times() - Retrieve the waiting time of every car in the incoming roads.
  - ❖ choose_action() - Decide whether to perform an explorative or exploitative action, according to an epsilon-greedy policy.
  - ❖ set_yellow_phase() - Activate the correct yellow light combination in sumo.
  - ❖ set_green_phase() - Activate the correct green light combination in sumo.
  - ❖ get_queue_length() - Retrieve the number of cars with speed = 0 in every incoming lane
  - ❖ get_state() - Retrieve the state of the intersection from sumo, in the form of cell occupancy.
  - ❖ replay() - Retrieve a group of samples from the memory and for each of them update the learning equation, then train.
  - ❖ save_episode_stats() - Save the stats of the episode to plot the graphs at the end of the session.

3.4.2.3. Memory

- The Memory class handles the memorization for the experience replay mechanism.

  - ❖ add_sample() - Add a sample into the memory.
  - ❖ get_samples() - Get n samples randomly from the memory.
  - ❖ checkSize() - Check how full the memory is.

3.4.2.4. Trafficgenerator

- This class has attributes such as the number of cars that needed to be generated, max_steps, and no of high and low priority vehicles. Using these attributes it generates a route file using the generate_routefile() method.

    - ❖ generate_routefile()-This method creates a route file and generates different route id's for the simulation using Weibull distribution.

3.4.2.5. TrainModel

- This class represents a file that defines the neural network and different methods to train the neural network and predict the outputs.

    - ❖ build_model() - Build and compile a fully connected deep neural network.
    - ❖ predict_one() - Predict the action values from a single state.
    - ❖ predict_batch() - Predict the action values from a batch of states.
    - ❖ train_batch() - Train the nn using the updated q-values.
    - ❖ save_model() - Save the current model in the folder as h5 file and a model architecture summary as png.

3.4.2.6. Utils

- This class represents a file that contains some directory-related functions, such as automatically handling the creation of new model versions and the loading of existing models for testing.

    - ❖ import_train_configuration() - Read the config file regarding the training and import its content.
    - ❖ import_test_configuration() - Read the config file regarding the testing and import its content.
    - ❖ set_sumo() - Configure various parameters of SUMO.
    - ❖ set_train_path() - Create a new model path with an incremental integer, also considering previously created model paths.
    - ❖ set_test_path() - Returns a model path that identifies the model number provided as argument and a newly created 'test' path.

3.4.2.7. Visualization

- This class is used for plotting data and analysing the results.

  - ❖ save_data_and_plot() - Produce a plot of performance of the agent over the session and save the relative data to txt.

- **Data members of all classes**

Data Type	Data Name	Description
string	path	the name of the folder that will contain the model versions and results
list	sumo_cmd	the command to set SUMO gui and configure various parameters of SUMO.
dict	config_settings	dictionary of training configurations
object	model	object of TrainModel class
object	memory	object of Memory class
object	traffic_gen	object of TrafficGenerator class
object	visualization	object of Visualization class
object	simulation	object of Simulation class
int	input_dim	the size of the state of the env from the agent perspective
int	output_dim	the number of possible actions
int	batch_size	the number of samples retrieved from the memory for each training iteration
int	learning_rate	the learning rate defined for the neural network
list	samples	list of training samples stored in memory
int	max_size	the max number of samples that the memory can contain

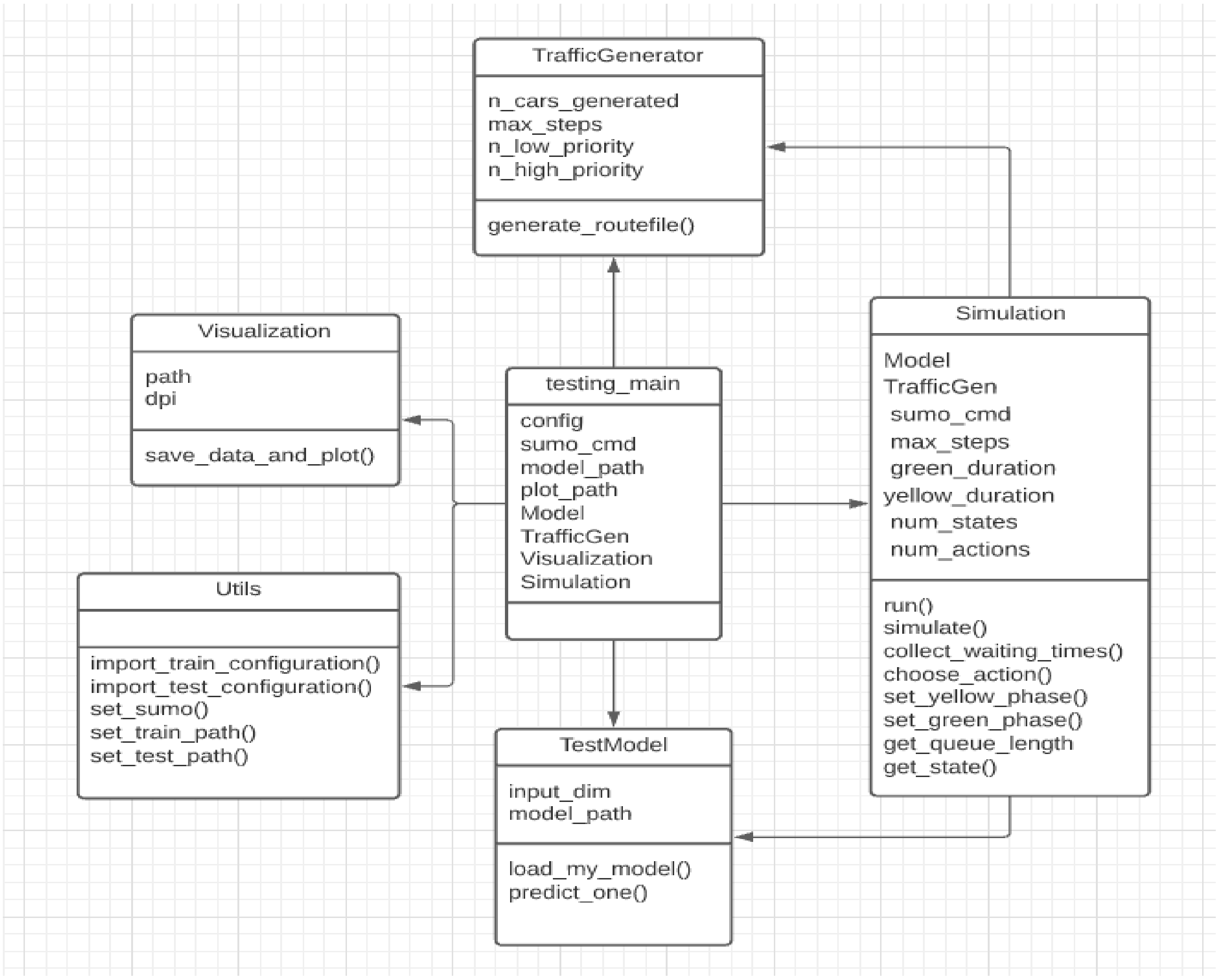
int	min_size	the min number of samples needed into the memory
int	n_cars_generated	no of cars to be generated in the simulation.
int	max_steps	the duration of each episode, with 1 step = 1 second (default duration in SUMO).
int	n_low_priority	no. of low priority vehicles.
int	n_high_priority	no. of high priority vehicles.
int	dpi	dots per inch
int	gamma	the gamma parameter of the bellman equation
int	green_duration	the duration in seconds of each green phase
int	yellow_duration	the duration in seconds of each yellow phase
int	num_states	the size of the state of the env from the agent perspective
int	num_actions	the number of possible actions
int	training_epochs	the number of training iterations executed at the end of each episode

3.5. Testing

3.5.1 Description

- We will be testing the model with a single test episode running a completely different and random simulation with emergency vehicles configured in such a way that they can ignore the red lights.
- We will be testing both the models(ANN and CNN) for multiple intersections as well.

3.5.2. Class diagram



3.5.2.1. Testing_main

- This class represents the file testing_main which is used to run the testing simulation.

3.5.2.2. TestModel

- This class represents a file that defines the neural network and different methods to train the neural network and predict the outputs.
    - ❖ load_my_model() - Load the model stored in the folder specified by the

model number, if it exists
- ❖ predict_one() - Predict the action values from a single state

3.5.2.3. Visualization

- This class is used for plotting data and analysing the results.
  - ❖ save_data_and_plot() - Produce a plot of performance of the agent over the session and save the relative data to txt

3.5.2.4. Simulation

- This class represents the file testing_simulation.
  - ❖ run() - Runs an episode of simulation, then starts a training session.
  - ❖ simulate() - Execute steps in sumo while gathering statistics.
  - ❖ collect_waiting_times() - Retrieve the waiting time of every car in the incoming roads.
  - ❖ choose_action() - Decide whether to perform an explorative or exploitative action, according to an epsilon-greedy policy.
  - ❖ set_yellow_phase() - Activate the correct yellow light combination in sumo.
  - ❖ set_green_phase() - Activate the correct green light combination in sumo.
  - ❖ get_queue_length() - Retrieve the number of cars with speed = 0 in every incoming lane
  - ❖ get_state() - Retrieve the state of the intersection from sumo, in the form of cell occupancy.

3.5.2.5. Utils

- This class represents a file that contains some directory-related functions, such as automatically handling the creation of new model versions and the loading of existing models for testing.
  - ❖ import_train_configuration() - Read the config file regarding the training and import its content
  - ❖ import_test_configuration() - Read the config file regarding the testing and import its content
  - ❖ set_sumo() - Configure various parameters of SUMO
  - ❖ set_train_path() - Create a new model path with an incremental integer, also considering previously created model paths
  - ❖ set_test_path() - Returns a model path that identifies the model number provided as argument and a newly created 'test' path.

- **Data members of all classes**

Data Type	Data Name	Description
dict	config	dictionary of testing configurations
list	sumo_cmd	the command to set SUMO gui and configure various parameters of SUMO.
string	model_path	path to the folder containing the model version.
string	plot_path	path to the folder containing the model version and test results
object	model	object of TestModel class
object	traffic_gen	object of TrafficGenerator class
object	visualization	object of visualization class
object	simulation	object of simulation class
int	input_dim	the size of the state of the env from the agent perspective.
int	n_cars_generated	no of cars to be generated in the simulation.
int	n_low_priority	no. of low priority vehicles.
int	n_high_priority	no. of high priority vehicles.
string	path	path to the folder containing the model versions and results.
int	dpi	dots per inch.
int	max_steps	the duration of each episode, with 1 step = 1 second (default duration in SUMO).
int	green_duration	duration of green light phase.
int	yellow_duration	duration of yellow light phase.
int	num_states	the size of the state of the env from the agent perspective.
int	num_actions	no of possible actions.

3.6. Memory

3.6.1 Description

- We will be implementing the experience replay mechanism.
- The memory.py file will handle the memorization for the experience replay mechanism.
- A function adds a sample into the memory, while another function retrieves a batch of samples from the memory.

3.7. Utils

3.7.1 Description

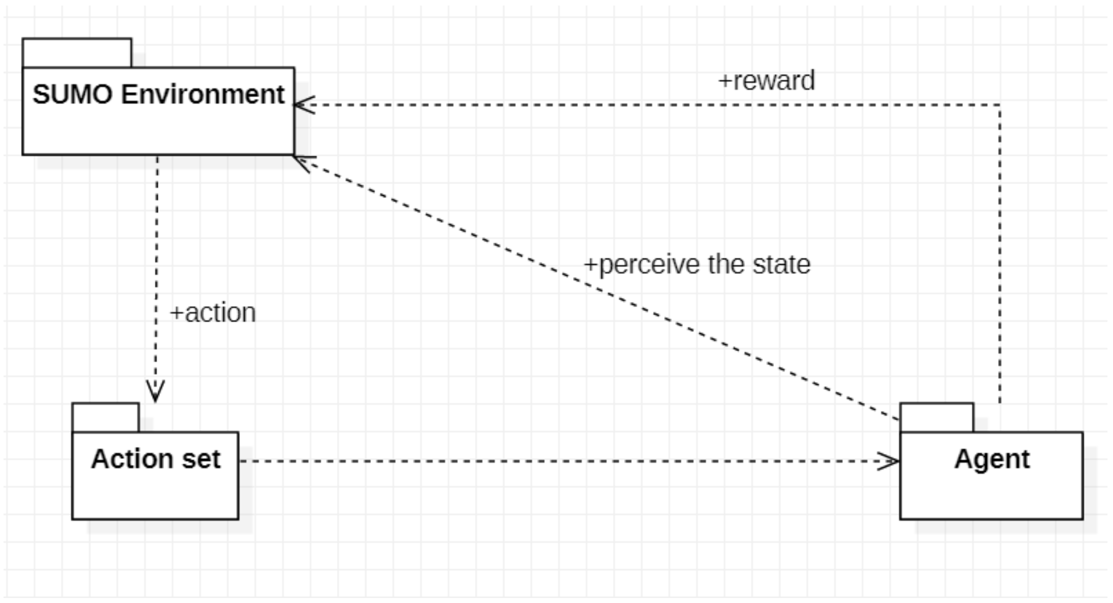
- This file contains directory related functions such as loading models for testing and creating different versions of trained models, importing configuration settings, etc.

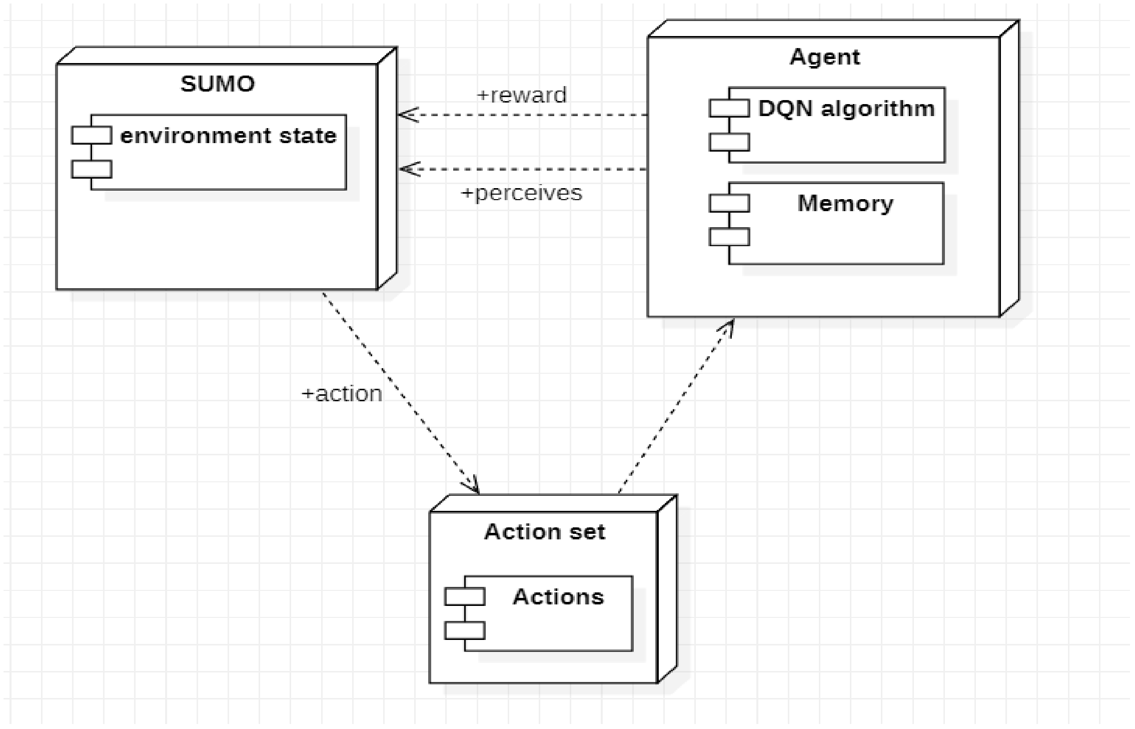
3.8. Visualization and analysis study

3.8.1 Description

- Once the testing is done we will be performing an analysis study to see the performance of the agent in both the models compared to normal traffic, to study the reward trend, and see the agent’s behaviour in different levels of traffic.

3.9. Packaging and Deployment Diagrams





Appendix A: Definitions, Acronyms and Abbreviations

1. SUMO - Simulation of Urban MObility
2. TraCI - Traffic Control Interface
3. API - Application programming interface
4. RL model - Reinforcement learning model
5. RAM - Random Access Memory
6. TCP - Transmission Control Protocol

Appendix B: References

1. SUMO documentation : https://sumo.dlr.de/docs/

   Copyright © 2001-2021 German Aerospace Center (DLR) and others.
2. R. S. Sutton, A. G. Barto et al. Introduction to reinforcement learning. MIT press Cambridge, 1998, vol. 135.
3. Monireh Abdoos, Nasser Mozayani, and Ana LC Bazzan. 2013. Holonic multi-agent system for traffic signals control. Engineering Applications of Artificial Intelligence 26, 5 (2013), 1575–1587.
4. Baher Abdulhai, Rob Pringle, and Grigoris J Karakoulas. 2003. Reinforcement learning for true adaptive traffic signal control. Journal of Transportation Engineering 129, 3 (2003), 278–285.
5. Andrea Vidali, Luca Crociani, Giuseppe Vizzari, Stefania Bandini, "A Deep Reinforcement Learning Approach to Adaptive Traffic Lights Management".
6. Wade Gendersa , Saiedeh Razavib, "Using a Deep Reinforcement Learning Agent for Traffic Signal Control".
7. Hua Wei, Guanjie Zheng, Huaxiu Yao, Zhenhui Li, "IntelliLight: A Reinforcement Learning Approach for Intelligent Traffic Light Control"
8. Arel C. Liu1 T. Urbanik A.G. Kohls, "Reinforcement learning-based multi-agent system for network traffic signal control"

Appendix C: Record of Change History

#	Date	Document Version No.	Change Description	Reason for Change
1.	**24/09/2021**	**1.0**	NA	**Initial document**
2.				

Appendix D: Traceability Matrix

Project Requirement Specification Reference Section No. and Name.	**DESIGN / HLD Reference Section No. and Name.**	**LLD Reference Section No. Name**
4. External Interface requirements	8. External interfaces	-
2. Product perspective	3. Design considerations	2. Design constraints, assumptions and dependencies.
3. Functional requirements	-	3. Design Description