Multi Agents in A city traffic Simulation High Level Design

ALon ReicHer, Liad gam

‏2023

תוכן

[Preface 4](#_Toc142574000)

[Notes for current version 4](#_Toc142574001)

[Simulation Concept – Model Description 5](#_Toc142574002)

[Environment 5](#_Toc142574003)

[Agent 5](#_Toc142574004)

[GUI 5](#_Toc142574005)

[Simulation Parameters 7](#_Toc142574006)

[Imported Data 7](#_Toc142574007)

[Fixed parameters 7](#_Toc142574008)

[Configurable parameters 7](#_Toc142574009)

[Exported Data 7](#_Toc142574010)

[Software Architecture 8](#_Toc142574011)

[Class Diagram (Unified Modeling Language) 8](#_Toc142574012)

[Data Structures 9](#_Toc142574013)

[Class: Route (Abstract Base Class) 9](#_Toc142574014)

[Method: decide\_first\_road () 9](#_Toc142574015)

[Method: get\_next\_road () 9](#_Toc142574016)

[Method: get\_alt\_road () 9](#_Toc142574017)

[Class: Random\_route (Route) 10](#_Toc142574018)

[Method: \_\_init\_\_(self, src\_node, dst\_node, road\_network) 10](#_Toc142574019)

[Method: decide\_first\_road (self) 10](#_Toc142574020)

[Method: get\_next\_road (self) 10](#_Toc142574021)

[Method: get\_alt\_road (self) 10](#_Toc142574022)

[Class: Q\_Learning\_Route (Route) 11](#_Toc142574023)

[Method: \_\_init\_\_(self, src\_node, dst\_node, road\_network, start\_time) 11](#_Toc142574024)

[Method: decide\_first\_road (self) 11](#_Toc142574025)

[Method: get\_next\_road (self) 11](#_Toc142574026)

[Method: get\_alt\_road(self) 12](#_Toc142574027)

[Class: Shortest\_path\_route (Route) 13](#_Toc142574028)

[Method: \_\_init\_\_(self, src\_node, dst\_node, road\_network) 13](#_Toc142574029)

[Method: decide\_first\_road (self) 13](#_Toc142574030)

[Method: get\_next\_road (self) 13](#_Toc142574031)

[Method: get\_alt\_road (self) 13](#_Toc142574032)

[Class: Car 14](#_Toc142574033)

[\_\_init\_\_(self, id, source\_node, destination\_node, starting\_time, road\_network, route\_algorithm='random'): 14](#_Toc142574034)

[decide\_route\_algorithm(self, route\_algorithm, source\_node, destination\_node): 14](#_Toc142574035)

[start\_car(self): 15](#_Toc142574036)

[move\_next\_road(self, time): 15](#_Toc142574037)

[check\_if\_finished(self): 15](#_Toc142574038)

[force\_finish(self): 16](#_Toc142574039)

[update\_time\_until\_next\_road(self, road): 16](#_Toc142574040)

[update\_travel\_time(self, time): 16](#_Toc142574041)

[get\_routing\_algorithm(self): 17](#_Toc142574042)

[get\_time\_until\_next\_road(self): 17](#_Toc142574043)

[get\_xy\_source(self): 17](#_Toc142574044)

[get\_xy\_destination(self): 18](#_Toc142574045)

[get\_xy\_current(self): 18](#_Toc142574046)

[Class: Road 19](#_Toc142574047)

[\_\_init\_\_(self, id, source\_node, destination\_node, length, max\_speed, activate\_traffic\_lights): 19](#_Toc142574048)

[calculate\_time(self): 19](#_Toc142574049)

[update\_speed(self, current\_time): 20](#_Toc142574050)

[update\_road\_speed\_dict(self, new\_speed): 20](#_Toc142574051)

[update\_eta\_dict(self): 20](#_Toc142574052)

[calculate\_eta(self, speed): 21](#_Toc142574053)

[get\_eta(self, time): 21](#_Toc142574054)

[Class: Node 22](#_Toc142574055)

[\_\_init\_\_(self, id, osm\_id, x, y, traffic\_lights, street\_count): 22](#_Toc142574056)

[Class: Road\_Network 23](#_Toc142574057)

[\_\_init\_\_(self, graph\_path, activate\_traffic\_lights=False): 23](#_Toc142574058)

[set\_nodes\_array(self): 23](#_Toc142574059)

[set\_roads\_array(self, activate\_traffic\_lights): 23](#_Toc142574060)

[set\_adjacency\_roads(self): 24](#_Toc142574061)

[remove\_blocked\_roads(self): 24](#_Toc142574062)

[block\_road(self, road\_id): 24](#_Toc142574063)

[unblock\_road(self, road\_id): 25](#_Toc142574064)

[unblock\_all\_roads(self): 25](#_Toc142574065)

[set\_roads\_speeds\_from\_dict(self, roads\_speeds: dict, current\_time: str, activate\_traffic\_lights: bool): 25](#_Toc142574066)

[update\_roads\_speeds(self, current\_time: str): 26](#_Toc142574067)

[get\_xy\_from\_node\_id(self, node\_id: int): 26](#_Toc142574068)

[get\_xy\_from\_osm\_id(self, osm\_id: int): 26](#_Toc142574069)

[get\_node\_from\_osm\_id(self, osm\_id: int): 27](#_Toc142574070)

[get\_road\_from\_src\_dst(self, src\_id, dst\_id): 27](#_Toc142574071)

[get\_shortest\_path(self, src\_id, dst\_id): 27](#_Toc142574072)

[check\_if\_path\_is\_blocked(self, path): 28](#_Toc142574073)

[Class: Car\_Manger 29](#_Toc142574074)

[\_\_init\_\_(self): 29](#_Toc142574075)

[add\_update\_to\_dictionary(self, time, car\_id, x, y, node\_id): 29](#_Toc142574076)

[add\_car(self, car, time): 29](#_Toc142574077)

[sort\_cars\_in\_simulation(self): 30](#_Toc142574078)

[calc\_nearest\_update\_time(self, time: datetime.datetime): 30](#_Toc142574079)

[find\_earliest\_waiting\_car(self): 30](#_Toc142574080)

[update\_cars(self, timeStamp: int, current\_datetime: datetime.datetime): 31](#_Toc142574081)

[is\_car\_stuck(self, car): 31](#_Toc142574082)

[is\_car\_finished(self, car): 31](#_Toc142574083)

[clear(self): 32](#_Toc142574084)

[get\_all\_cars\_ids(self): 32](#_Toc142574085)

[Class: Simulation\_Manager 33](#_Toc142574086)

[\_\_init\_\_(self, graph, time\_limit: int, activate\_traffic\_lights=False, start\_time=datetime.datetime(year=2023, month=6, day=29, hour=8, minute=0, second=0)): 33](#_Toc142574087)

[block\_road(self, road\_id): 33](#_Toc142574088)

[unblock\_road(self, road\_id): 34](#_Toc142574089)

[unblock\_all\_roads(self): 34](#_Toc142574090)

[update\_simulation\_clock(self, time: int): 34](#_Toc142574091)

[read\_road\_speeds(self, datetime\_obj: datetime.datetime): 34](#_Toc142574092)

[set\_up\_simulation(self, cars: list): 35](#_Toc142574093)

[start\_simulation(self): 35](#_Toc142574094)

[end\_simulation(self, simulation\_number): 35](#_Toc142574095)

[run\_full\_simulation(self, cars, number\_of\_simulations=1): 36](#_Toc142574096)

[write\_simulation\_results(self, copy\_cars: list, i: int): 36](#_Toc142574097)

[get\_simulation\_routes(self, cars: list, simulation\_number: int): 36](#_Toc142574098)

[get\_key\_from\_value(self, dictionary, value): 37](#_Toc142574099)

[transform\_node\_id\_route\_to\_osm\_id\_route(self, route): 37](#_Toc142574100)

[Class: Simulation\_Results\_Manager 38](#_Toc142574101)

[\_\_init\_\_(self, graph\_name): 38](#_Toc142574102)

[save\_results\_to\_JSON (self, results): 38](#_Toc142574103)

[read\_results\_from\_JSON (self): 38](#_Toc142574104)

[Timing Diagram 39](#_Toc142574105)

[Bibliography 39](#_Toc142574106)

|  |  |  |  |
| --- | --- | --- | --- |
| **Version** | **Date** | **Author** | **Description** |
|  | 4.8.23 |  |  |
| 0.1 |  |  | Creation |

## Preface

### Notes for current version

# Simulation Concept – Model Description

## Environment

The environment is made up of one-way roads, each road starts at an intersection and ends at another intersection. Each road will include its length, the current speed on it, the roads you can pass to from it. As soon as an agent reaches the road, the agent will calculate with the help of the road data the time it will take him to complete the trip on this road.

## Agent

An agent is a car driving on the roads on the way to a predetermined destination. The agent will be able to use different ways to determine the mode of movement. It can be randomly oriented, according to the shortest distance from the target, or a reinforcement-based learning algorithm (Q-learning).

An agent's state is defined by the current time (time and day of the week), current node and destination node. An agent will take information from the relevant table for the day of the week from a table updated for that day and the time slice of the previous week, according to its current location, the current time slice, and the target vertex.

For example: a car in edge no.1 that wants to reach edge no.3, at 10:00 on Sunday, will check last week's table - on Sunday at 10:00, when the current curve is 1 and the destination is curve 3, which next curve (continuation of curve 1) will give the maximum return.

In the simulation we will have a Q table from each source vertex to each destination vertex on the simulation graph, the return for each arc is negative, and is defined in direct proportion to the time it takes to cross it.

An agent will also contribute to filling the distance table: when an agent uses the shortest path algorithm - if the data for his trip is not in the distance table, he will update the distance table according to his origin and destination. Otherwise, the relevant data for the agent already exists in the table, and he will take it from there.

● In the event that an agent uses the shortest path algorithm, and encounters a block in an arc he wanted to enter - he will choose an arc from among the other optional arcs, and will continue to act according to the same principle.

● For all algorithms - if an agent cannot move forward due to blocking of all arcs, it will wait/stop completely

## GUI

At the end of the simulation, the map of the area will be printed with two main features:

1. Tracking the travel path of specific agents on the map.

2. Color the edges colors based on road's speeds.

# Simulation Parameters

### Imported Data

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Explanation** | **Calculated value** |
| G | Graph of the roads, as imported with the OSM API. | --- |
| Speed\_Json | Data (in a JSON format) about cars speed on the roads in the imported OSM graph. |  |
|  |  |  |

### Fixed parameters

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Explanation** | **Value** |
| INTERVAL\_TIME | The time intervals of the speed data. | 10 (minutes) |
|  |  |  |

### Configurable parameters

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Explanation** | **Valid values** |
| location | The simulated location. A city/ part of a city. | City names\coordinates |
| routing\_algorithm | The algorithm which is used by agents to determine the next road/edge of their route. | "random"/"shortest path"/"q learning" |
| Speed\_noise\_dist | The distribution used to generate synthetic movement data from the real data. The generation method will apply noise to the real data, using gaussian distribution or uniform distribution between -10 to 10 (with no negative speeds). | "gauss"/"uni" |

### Exported Data

|  |  |  |
| --- | --- | --- |
| **Symbol** | **Explanation** | **Calculated value** |
| route | Routes of the agents that finished their navigation | route |
| Q\_table | Calculated Q table |  |

# Software Architecture

## תמונה שמכילה תרשים התיאור נוצר באופן אוטומטי Class Diagram (Unified Modeling Language)[[1]](#footnote-1)



## Data Structures

# Class: Route (Abstract Base Class)

This abstract base class defines the interface and common methods for different routing strategies.

### Method: decide\_first\_road ()

**Functionality:**

* This method is abstract and should be implemented by subclasses.
* It is responsible for deciding the first road that a car should take.
* Returns the selected first road.

### Method: get\_next\_road ()

**Functionality:**

* This method is abstract and should be implemented by subclasses.
* It determines the next road that the car should take based on the current road and other parameters.
* Returns the selected next road.

### Method: get\_alt\_road ()

**Functionality:**

* This method is abstract and should be implemented by subclasses.
* It determines an alternative road when the chosen road is blocked or unavailable.
* Returns the selected alternative road, or None if no other road is available.

# Class: Random\_route (Route)

This class represents a random routing strategy for car simulation.

### Method: \_\_init\_\_(self, src\_node, dst\_node, road\_network)

**Input:**

* src\_node (int): The source node ID.
* dst\_node (int): The destination node ID.
* road\_network (Road\_Network): The road network object.

### Method: decide\_first\_road (self)

**Functionality:**

* Selects the first road randomly from roads connected to the source node.

**Output:**

* (Road): the selected first road.

### Method: get\_next\_road (self)

**Functionality:**

* Selects the next road randomly from the list of adjacent roads.
* Ensures that the selected road is not blocked and has adjacent roads.

**Output:**

* (Road): the selected road.

### Method: get\_alt\_road (self)

**Input and Functionality:**

Try to find an alternative road to the current road if it's blocked.

**Output:**

* (Road): the alternative road, or None.

# Class: Q\_Learning\_Route (Route)

This class represents a Q-learning based routing strategy for car simulation.

### Method: \_\_init\_\_(self, src\_node, dst\_node, road\_network, start\_time)

**Input:**

* src\_node (int): The source node ID.
* dst\_node (int): The destination node ID.
* road\_network (Road\_Network): The road network object.
* start\_time (datetime.datetime): The starting time of the simulation.

**Functionality:**

* Initializes the Q-learning route with source, destination, and other parameters.
* Creates an instance of the QLearning class to perform Q-learning.
* Loads or trains the Q-table based on previous experience or initializes a new one.
* Defines the agent's path using Q-learning.

### Method: decide\_first\_road (self)

**Functionality:**

* Chooses the first road using Q-learning based on the source node.

**Output:**

* (Road): the selected first road.

### Method: get\_next\_road (self)

**Functionality:**

* This method uses Q-learning to select the next road based on the current road's destination node.

**Output:**

* (Road): the selected road.

### Method: get\_alt\_road(self)

**Input and Functionality:**

Try to find an alternative road to the current road if it's blocked, based on the q table of the agent.

**Output:**

* (Road): the alternative road, or None.

# Class: Shortest\_path\_route (Route)

This class represents a shortest path-based routing strategy for car simulation.

### Method: \_\_init\_\_(self, src\_node, dst\_node, road\_network)

**Input:**

* src\_node (int): The source node ID.
* dst\_node (int): The destination node ID.
* road\_network (Road\_Network): The road network object.

### Method: decide\_first\_road (self)

**Functionality:**

* Determines the first road using a shortest path algorithm.

**Output:**

* (Road): the selected first road.

### Method: get\_next\_road (self)

**Functionality:**

* Determines the next road using a shortest path algorithm.

**Output:**

* (Road): the selected road.

### Method: get\_alt\_road (self)

**Input and Functionality:**

* This method calculates an alternative road based on minimum distance from adjacent roads.

**Output:**

* (Road): the alternative road, or None.

# Class: Car

This class represents a car within a road network simulation.

### \_\_init\_\_(self, id, source\_node, destination\_node, starting\_time, road\_network, route\_algorithm='random', num\_episodes = 2000, use\_existing\_q\_table = True):

Input:

* id (int): The unique identifier for the car.
* source\_node (int): The source node id which the car starts its journey.
* destination\_node (int): The destination node id where the car aims to arrive.
* starting\_time (datetime): The time when the car's journey begins.
* road\_network (Road\_Network): The road network within which the car operates.
* route\_algorithm (str: optional): The algorithm the car will use to decide its route (default: 'random').
* num\_episodes (int): \*only matters if the car uses Q Learning\*. The number of episodes the agent will do in its training.
* use\_existing\_q\_table (bool): \*only matters if the car uses Q Learning\*. Determines if the Q Learning agent will use past experience of the same path, if exists.

Functionality:

* Initializes a new instance of the Car class with the provided attributes.
* Sets up various attributes to manage the car's state, route, and movement within the simulation.
* Sets up attributes for the q learning route.

Output:

• None

### decide\_route\_algorithm(self, route\_algorithm, source\_node, destination\_node):

Input:

* route\_algorithm (str): The string representing the route algorithm.
* source\_node (int): Source node ID.
* destination\_node (int): Destination node ID.

Functionality:

* Decides the route algorithm based on the provided string.
* Returns the selected route algorithm instance.

Output:

* (Route): The selected route algorithm.

### start\_car(self):

Functionality:

* Moves the car to the first road based on the starting node and the car's route object. Updates the car's time until the next road.

Output:

* (Road): The first road the car will travel to.

### move\_next\_road(self, time):

Input:

* Time (float): The time interval.

Functionality:

* Moves the car to the next road based on the route's next node.
* Updates the car's time until the next road.
* Returns the next road the car will travel to, or "blocked" if the road is blocked.

Output:

* (Road or str): The next road the car will travel to, or "blocked" if the road is blocked.

### check\_if\_finished(self):

Functionality:

* Checks if the car has reached its destination.

Output:

* (bool): True if the car has reached its destination, False otherwise.

### force\_finish(self):

Functionality:

* Used on we force the car to finish
* Appends the last road to the past\_roads list.

Output:

* None

### update\_time\_until\_next\_road(self, road):

Input:

* road (Road): the road that the car is driving in.

Functionality:

* Updates the time until the car finishes the current road and reaches the next road.

Output:

* None

### update\_travel\_time(self, time):

Input:

* time (float): The time interval.

Functionality:

* Updates the total travel time and the time until the next road.

Output:

* (datetime.timedelta): the car's total travel time

### get\_routing\_algorithm(self):

Functionality:

* Gets the routing algorithm the car uses.

Output:

* str: The routing algorithm name.

### get\_time\_until\_next\_road(self):

Functionality:

* Gets the time until the car reaches the next road.
* If car is blocked then return 600 seconds.

Output:

* (int): The time in seconds.

### get\_xy\_source(self):

Functionality:

* Gets the x, y coordinates of the source node.

Output:

* tuple: (x, y) coordinates.

### get\_xy\_destination(self):

Functionality:

* Gets the x, y coordinates of the destination node.

Output:

* tuple: (x, y) coordinates.

### get\_xy\_current(self):

Functionality:

* Gets the x, y coordinates of the current node.

Output:

* tuple: (x, y) coordinates.

# Class: Road

Represents a road segment in a road network.

### \_\_init\_\_(self, id, source\_node, destination\_node, length, max\_speed, activate\_traffic\_lights, rain\_intensity = 0):

Input:

* id (int): The unique identifier for the road.
* source\_node (Node): The source node from which the road originates (class Node).
* destination\_node (Node): The destination node where the road ends (class Node).
* Length (int): The length of the road in meters.
* max\_speed (int): The maximum speed allowed on the road in km/h.
* activate\_traffic\_lights (bool): A boolean indicating whether traffic lights are active on the road.
* rain\_intensity (int): optinal, represents the intensity of the rain where 0 = no rain, 1 = little bit of rain, 2 = moderate rain, 3 = heavy rain.

Functionality:

* Initializes a new instance of the Road class with the provided attributes.
* Sets up various attributes to represent the road's characteristics and traffic conditions.

Output:

* None

### calculate\_time(self):

Functionality:

* Calculates the estimated time it takes to travel the road, based on self.current\_speed.
* If activate\_traffic\_lights is true then, adds a random time based on the road's destination node number of intersecting roads. Supposed to simulate real life traffic signals time.

Output:

* (float): The estimated time in seconds.

### update\_speed(self, current\_time, traffic\_white\_noise):

Input:

* current\_time (datetime): The current time.
* traffic\_white\_noise (bool): adds a little white noise to the known speed of last week to get a little bit of diversity.

Functionality:

* Updates the current speed of the road based on the provided time, traffic\_white\_noise and rain\_intensity.
* Recalculates the estimated time of arrival (ETA).

Output:

* (float): The updated estimated time of arrival (ETA) in seconds.

### update\_road\_speed\_dict(self, new\_speed):

Input:

* new\_speed (dict): A dictionary containing times and corresponding road speeds.

Functionality:

* Updates the road speed dictionary and recalculates ETA values.

Output:

* None

### update\_eta\_dict(self):

Functionality:

* Updates the estimated time of arrival (ETA) dictionary based on last week's road speeds.

Output:

* None

### calculate\_eta(self, speed):

Input:

* Speed (int): The speed of travel on the road in km/h.

Functionality:

* Calculates the estimated time of arrival (ETA) based on the provided speed.
* This function used in update\_eta\_dict.

Output:

* float: The estimated time of arrival (ETA) in seconds.

### get\_eta(self, time):

Input:

* time (str): The time for which ETA is requested as a string. Represented by hours and minutes, i.e: "08:00".

Functionality:

* Gets the estimated time of arrival (ETA) for the specified time.

Output:

* float: The estimated time of arrival (ETA) in seconds.

### block (self):

Functionality:

* Turning on self.is\_blocked flag.
* Making the road's eta infinite.

Output:

* None

### unblock (self):

Functionality:

* Turning off self.is\_blocked flag.
* Recalculates the road's eta

Output:

* (float, int): the new eta and the new current speed.

# Class: Node

### \_\_init\_\_(self, id, osm\_id, x, y, traffic\_lights, street\_count):

Input:

* id: The unique identifier for the node.
* osm\_id: The OpenStreetMap identifier for the node.
* x: The x-coordinate of the node's location.
* y: The y-coordinate of the node's location.
* traffic\_lights: A boolean indicating whether the node has traffic lights.
* street\_count: The number of streets connected to the node.

Functionality:

* Initializes a new instance of the Node class with the provided attributes.
* Sets up various attributes to represent the node's characteristics and location.

Output:

* None

# Class: Road\_Network

This class represents a road network, containing the graph of the simulation, all the roads in the graph, and related data.

### \_\_init\_\_(self, graph\_path, activate\_traffic\_lights=False):

Input:

* graph\_path: The path to the graph data used to construct the road network.
* activate\_traffic\_lights (bool, optional): Whether to activate traffic lights for roads (default: False).

Functionality:

* Initializes a new instance of the Road\_Network class using the provided graph data and optional parameters.
* Sets up various attributes to manage the road network, nodes, roads, and adjacency information.

Output:

* None

### set\_nodes\_array(self):

Functionality:

* Initialize and populate the nodes\_array attribute with Node objects.

Output:

* None

### set\_roads\_array(self, activate\_traffic\_lights):

Input:

* activate\_traffic\_lights (bool): Whether to activate traffic lights for roads.

Functionality:

* Initialize and populate the roads\_array attribute with Road objects based on the provided graph data and traffic light activation.

Output:

* None

### set\_adjacency\_roads(self):

Input:

* None

Functionality:

* Set the adjacent\_roads attribute for each road in the roads\_array.

Output:

* None

### remove\_blocked\_roads(self):

Input:

* None

Functionality:

* Remove roads from roads\_array that have no adjacent roads.

Output:

* None

### block\_road(self, road\_id):

Input:

* road\_id (int): ID of the road to be blocked.

Functionality:

* Block a specific road by marking it as blocked.
* Output:
* None

### unblock\_road(self, road\_id):

Input:

* road\_id (int): ID of the road to be unblocked.

Functionality:

* Unblock a specific road by marking it as unblocked.

Output:

* None

### unblock\_all\_roads(self):

Functionality:

* Unblock all previously blocked roads.

Output:

* None

### set\_roads\_speeds\_from\_dict(self, roads\_speeds: dict, current\_time: str, activate\_traffic\_lights: bool):

Input:

* roads\_speeds (dict): Dictionary of road\_id: speed for different times of the day.
* current\_time (str): Current time in the simulation.
* activate\_traffic\_lights (bool): Whether to activate traffic lights for roads.

Functionality:

* Update road speeds based on the provided speeds dictionary and current time.

Output:

* None

### update\_roads\_speeds(self, current\_time: str):

Input:

* current\_time (str): Current time in the simulation.

Functionality:

* Update road speeds based on the current time.

Output:

* None

### get\_xy\_from\_node\_id(self, node\_id: int):

Input:

* node\_id (int): Node ID.

Functionality:

* Get x, y coordinates of a node based on its node ID.

Output:

* tuple: x, y coordinates.

### get\_xy\_from\_osm\_id(self, osm\_id: int):

Input:

* osm\_id (int): OSM ID of the node.

Functionality:

* Get x, y coordinates of a node based on its OSM ID.

Output:

* tuple: x, y coordinates.

### get\_node\_from\_osm\_id(self, osm\_id: int):

Input:

* osm\_id (int): OSM ID of the node.

Functionality:

* Get a node object based on its OSM ID.

Output:

* Node object: Node corresponding to the OSM ID.

### get\_road\_from\_src\_dst(self, src\_id, dst\_id):

Input:

* src\_id (int): Source node ID.
* dst\_id (int): Destination node ID.

Functionality:

* Get a road object based on source and destination node IDs.

Output:

* Road object: Road between the source and destination nodes.

### get\_shortest\_path(self, src\_id, dst\_id):

* Input:
* src\_id (int): Source node ID.
* dst\_id (int): Destination node ID.

Functionality:

* Get the shortest path between two node IDs.

Output:

* list: List of node IDs representing the shortest path.

### check\_if\_path\_is\_blocked(self, path):

Input:

* path (list): List of node IDs representing a path.

Functionality:

* Check if a given path is blocked by any blocked roads.

Output:

* bool: True if the path is blocked, False otherwise.

# Class: Car\_Manger

This class manages cars within a road network simulation. It adds, removes, and updates cars during the simulation.

### \_\_init\_\_(self):

Input:

• None

Functionality:

* Initializes a new instance of the CarManager class.
* Sets up various attributes to manage cars in different stages of the simulation.

Output:

* None

### add\_update\_to\_dictionary(self, time, car\_id, x, y, node\_id):

Input:

* time (datetime.datetime): Time of the update.
* car\_id (int): ID of the car.
* x (float): X-coordinate of the car's position.
* y (float): Y-coordinate of the car's position.
* node\_id (int): ID of the current node.

Functionality:

* Add car update information to the dictionary.

Output:

* None

### add\_car(self, car, time):

Input:

* car (Car): Car object to be added.
* time (datetime.datetime): Current time in the simulation.

Functionality:

* Add a car to the simulation based on its starting time.

Output:

* None

### sort\_cars\_in\_simulation(self):

Input:

* None

Functionality:

* Sort cars in the simulation based on the time until the next road.
* Update the nearest update time.

Output:

* bool: False if no cars are in simulation, True otherwise.

### calc\_nearest\_update\_time(self, time: datetime.datetime):

Input:

* time (datetime.datetime): Current time in the simulation.

Functionality:

* Calculate the time of the nearest update based on waiting and simulation cars.

Output:

* int: The time of the nearest update.

### find\_earliest\_waiting\_car(self):

Input:

• None

Functionality:

* Find the earliest time that a car is waiting to enter the simulation.

Output:

* datetime.datetime or None: The earliest starting time of a waiting car, or None if no cars are waiting.

### update\_cars(self, timeStamp: int, current\_datetime: datetime.datetime):

Input:

* timeStamp (int): The simulation time step.
* current\_datetime (datetime.datetime): Current time in the simulation.

Functionality:

* Update all the cars in the simulation based on the current time step.

Output:

* None

### is\_car\_stuck(self, car):

Input:

* car (Car): The car to check.

Functionality:

* Check if a car is stuck in the simulation.

Output:

* bool: True if the car is stuck, False otherwise.

### is\_car\_finished(self, car):

Input:

* car (Car): The car to check.

Functionality:

* Check if a car has finished its journey in the simulation.

Output:

* bool: True if the car has finished, False otherwise.

### clear(self):

Input:

* None

Functionality:

* Clear all simulation data and reset the CarManager.

Output:

* None

### get\_all\_cars\_ids(self):

Input:

* None

Functionality:

* Get the IDs of all cars currently in the simulation and those blocked.

Output:

* list[int]: List of car IDs.

# Class: Simulation\_Manager

Manages the simulation by creating and updating the road network, managing cars, and printing results.

### \_\_init\_\_(self, graph, time\_limit: int, activate\_traffic\_lights=False, start\_time=datetime.datetime(year=2023, month=6, day=29, hour=8, minute=0, second=0)):

Input:

* graph: The networkx graph representing the road network.
* time\_limit (int): The maximum time the simulation will run in seconds.
* activate\_traffic\_lights (bool, optional): Indicates whether traffic lights are activated. Default is False.
* start\_time (datetime.datetime, optional): The starting datetime of the simulation. Default is June 29, 2023, 08:00:00.

Functionality:

* Initializes a new instance of the Simulation\_manager class.
* Sets up various attributes to manage the simulation, road network, and car manager.

Output:

* None

### block\_road(self, road\_id):

Input:

* road\_id (int): The ID of the road to be blocked.

Functionality:

* Block a road in the road network.

Output:

* None

### unblock\_road(self, road\_id):

Input:

* road\_id (int): The ID of the road to be unblocked.

Functionality:

* Unblock a previously blocked road.

Output:

* None

### unblock\_all\_roads(self):

Input:

* None

Functionality:

* Unblock all roads in the road network.

Output:

* None

### update\_simulation\_clock(self, time: int):

Input:

* time (int): The time interval to update the simulation clock in seconds.

Functionality:

* Update the simulation clock and road speeds.

Output:

* None

### read\_road\_speeds(self, datetime\_obj: datetime.datetime):

Input:

* datetime\_obj (datetime.datetime): The datetime for which road speeds are to be read.

Functionality:

* Read road speeds from a JSON file and update the road network.

Output:

* None

### set\_up\_simulation(self, cars: list):

Input:

* cars (list): List of Car objects to be added to the simulation.

Functionality:

* Set up the simulation by adding cars to the car manager.

Output:

* None

### start\_simulation(self):

Input:

* None

Functionality:

* Start the simulation and run it until all cars finish or time limit is reached.

Output:

* None

### end\_simulation(self, simulation\_number):

Input:

* simulation\_number (int): The index of the current simulation.

Functionality:

* Print the results of the simulation.

Output:

* None

### run\_full\_simulation(self, cars, number\_of\_simulations=1):

Input:

* cars (list): List of Car objects for the simulation.
* number\_of\_simulations (int, optional): Number of simulations to run. Default is 1.

Functionality:

* Run the full simulation process including setup, execution, and result printing.

Output:

* None

### write\_simulation\_results(self, copy\_cars: list, i: int):

Input:

* copy\_cars (list): List of Car objects used in the simulation.
* i (int): Index of the current simulation.

Functionality:

* Write simulation results to the simulation results list.

Output:

* None

### get\_simulation\_routes(self, cars: list, simulation\_number: int):

Input:

* cars (list): List of Car objects.
* simulation\_number (int): Index of the simulation.

Functionality:

* Retrieve routes passed by cars in a simulation.

Output:

* list: List of routes passed by cars in the simulation.

### get\_key\_from\_value(self, dictionary, value):

Input:

* dictionary (dict): The dictionary to search in.
* value: The value to search for.

Functionality:

* Retrieve the corresponding key from a dictionary given a value.

Output:

* key: The key corresponding to the given value.

### transform\_node\_id\_route\_to\_osm\_id\_route(self, route):

Input:

* route (list): List of node IDs representing a route.

Functionality:

* Transform a route from node IDs to OSM IDs.

Output:

* list: List of OSM IDs representing the transformed route.

# Class: Simulation\_Results\_Manager

### \_\_init\_\_(self, graph\_name):

Input:

* graph name (str): The graph name that was used for the graphml file.

Functionality:

* Initializes a new instance of the Simulation\_results\_manager class.

Output:

* None

### save\_results\_to\_JSON (self, results):

Input:

* result (dict): All the simulation results.

Functionality:

* Saves all the results in a JSON file according to the graph's name.

Output:

* None

### read\_results\_from\_JSON (self):

Input:

* None.

Functionality:

* Read simulation results from a JSON file.

Output:

* dict: dict that contains all the simulation results.

## Timing Diagram

1.call simulation manger() -> 2. Simulation manager will create road\_network with all the road in it -> 3. Random agents will be created at different times -> 4. Every agent will use Route class to navigate to its destination ->5. Simulation will stop after certain amount of time and the ending result will be shown.

## Bibliography

|  |  |
| --- | --- |
| [1] | IEEE-SA, *IEEE 802.11n-2009—Amendment 5: Enhancements for Higher Throughput,* 2009. |
| [2] | IEEE-SA, *IEEE 802.11-2007,* 2007. |

1. <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-uml/> [↑](#footnote-ref-1)