My Project

Generated by Doxygen 1.9.1

Chapter 1

Hierarchical Index

1.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

Class Index

2.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

Class Documentation

3.1 AddedEdge Struct Reference

Struct to store all necessary info about edge.

`#include <GraphAdjuster.h>`

Public Member Functions

- **AddedEdge** (vertex_t first, vertex_t second, char type, double length)

Public Attributes

- vertex_t **FirstVertex**
- vertex_t **SecondVertex**
- char **Type**
- double **Length**

3.1.1 Detailed Description

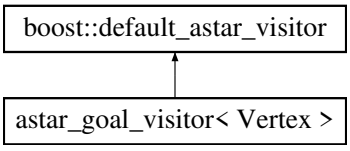
Struct to store all necessary info about edge.

The documentation for this struct was generated from the following file:

- GraphAdjuster.h

3.2 astar_goal_visitor< Vertex > Class Template Reference

Inheritance diagram for astar_goal_visitor< Vertex >:



Public Member Functions

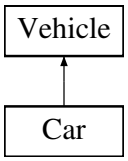
- **astar_goal_visitor** (Vertex goal)
- template<class Graph >
  void **examine_vertex** (Vertex u, Graph &g)

The documentation for this class was generated from the following file:

- Map.h

3.3 Car Class Reference

Inheritance diagram for Car:



Public Member Functions

- Car (double startTime, double waiting, MapPosition start, MapPosition end, double consumption, double batteryLevel, double relativeSpeed)

  Initializes car object.
- void StartReturning (double actualTime)

  Appropriately changes car setup to returning to start. Changes start and end position and sets flags of the car to returning.
- bool IsReturning ()
- double GetWaitingTime ()

Additional Inherited Members

3.3.1 Constructor & Destructor Documentation

3.3.1.1 Car()

```
Car::Car (
            double startTime,
            double waitingTime,
            MapPosition start,
            MapPosition end,
            double consumption,
            double batteryLevel,
            double relativeSpeed )
```

Initializes car object.

Parameters

startTime	Time of the car departure.
waitingTime	Waiting time for start returning (after reaching end position).
start	Start position.
end	End position.
consumption	Car consumption (battery percentages per minute).
batteryLevel	Starting battery level.
relativeSpeed	Speed of the car (in kilometers per minute (1/60 $*$ km/hr)).

3.3.2 Member Function Documentation

3.3.2.1 GetWaitingTime()

```
double Car::GetWaitingTime ( )
```

Returns

Returns waiting time of the car in the end to start returnning.

3.3.2.2 IsReturning()

```
bool Car::IsReturning ( )
```

Returns

Returns `true` if car is returning (already has been in the original finish).

3.3.2.3 StartReturning()

```
void Car::StartReturning (
            double actualTime )
```

Appropriately changes car setup to returning to start. Changes start and end position and sets flags of the car to returning.

The documentation for this class was generated from the following files:

- Car.h
- Car.cpp

3.4 ChargingStation Class Reference

Class representing one charging station.

`#include <ChargingStation.h>`

Public Member Functions

- ChargingStation ()

  Initializes charging station.
- ChargingStation (int32_t stationID, MapPosition position, int32_t capacity, int32_t closestCityID, double chargingWaitingTime, double estimatedChargingLevel)

  Initializes charging station.
- void AddCustomer (int32_t carID)

  Adds customer into the waiting line.
- void RemoveCustomer ()

  Updates information about customers when some customer left the charging station.
- int NextCustomer ()

  Returns ID of the next customer waiting in line for charging and removes it from the waiting queue.
- double GetChargingTime (double batteryLevel)

  Computes waiting time needed for the vehicle to be fully charged.
- double GetExpectedWaitingTime ()

  Estimates waiting time of the vehicle which just aproached charging station until it starts charging.

Public Attributes

- int32_t **StationID_**
- MapPosition **Position_**
- int32_t **CityID_**
- int32_t **Capacity_**
- int32_t **NumCustomers_**

3.4.1 Detailed Description

Class representing one charging station.

3.4.2 Constructor & Destructor Documentation

3.4.2.1 ChargingStation() [1/2]

`ChargingStation::ChargingStation ( )`

Initializes charging station.

3.4.2.2 ChargingStation() [2/2]

```
ChargingStation::ChargingStation (
            int32_t stationID,
            MapPosition position,
            int32_t capacity,
            int32_t closestCityID,
            double chargingWaitingTime,
            double estimatedChargingLevel )
```

Initializes charging station.

Parameters

stationID	ID of the station.
position	Station position.
capacity	Station capacity.
closestCityID	Closest city center from the station.
chargingWaitingTime	Waiting time of the complete charge of the battery.
estimatedChargingLevel	Estimated charing level of the customers.

3.4.3 Member Function Documentation

3.4.3.1 AddCustomer()

```
void ChargingStation::AddCustomer (
            int32_t carID )
```

Adds customer into the waiting line.

Parameters

carID	ID of the customers car.

3.4.3.2 GetChargingTime()

```
double ChargingStation::GetChargingTime (
            double batteryLevel )
```

Computes waiting time needed for the vehicle to be fully charged.

Parameters

vehicle	Vehicle object to compute waiting time (for possible different types of the vehicle extension).

Returns

Returns computed waiting time.

3.4.3.3 GetExpectedWaitingTime()

```
double ChargingStation::GetExpectedWaitingTime ( )
```

Estimates waiting time of the vehicle which just aproached charging station until it starts charging.

Returns

Return estimated waiting time in the queue for the charging.

3.4.3.4 NextCustomer()

```
int ChargingStation::NextCustomer ( )
```

Returns ID of the next customer waiting in line for charging and removes it from the waiting queue.

Returns

Returns ID of the next waiting customer or -1 if no next customer available.

3.4.3.5 RemoveCustomer()

```
void ChargingStation::RemoveCustomer ( )
```

Updates information about customers when some customer left the charging station.

The documentation for this class was generated from the following files:

- ChargingStation.h
- ChargingStation.cpp

3.5 City Struct Reference

Struct to store city coordinates.

```
#include <GraphAdjuster.h>
```

Public Member Functions

- **City** (double lat, double lon)

Public Attributes

- double **Lat**
- double **Lon**

3.5.1 Detailed Description

Struct to store city coordinates.

The documentation for this struct was generated from the following file:

- GraphAdjuster.h

3.6 ComponentsDistance Struct Reference

Struct to store necessary info about the pair of vertices.

`#include <GraphAdjuster.h>`

Public Member Functions

- **ComponentsDistance** (double distance, vertex_t firstID, vertex_t secondID)

Public Attributes

- double **Distance**
- vertex_t **FirstVertexID**
- vertex_t **SecondVertexID**

3.6.1 Detailed Description

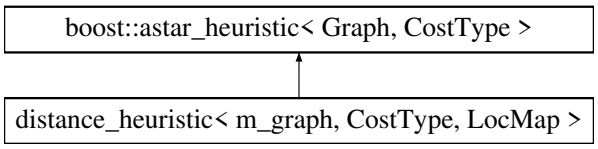
Struct to store necessary info about the pair of vertices.

The documentation for this struct was generated from the following file:

- GraphAdjuster.h

3.7 distance_heuristic< m_graph, CostType, LocMap > Class Template Reference

Inheritance diagram for distance_heuristic< m_graph, CostType, LocMap >:



Public Member Functions

- **distance_heuristic** (LocMap l, vertex_t goal)
- CostType **operator()** (vertex_t u)

The documentation for this class was generated from the following file:

- Map.h

3.8 Edge Class Reference

Class for edge representation (road between the junctions).

`#include <Edge.h>`

Public Member Functions

- Edge ()

  Initializes the edge.
- void ResetSimulation ()

  Resets all simulation information.
- void AddType (char type)

  Sets proper edge type based on the input.
- void AddType (PermitedRoadTypes type)

  Sets proper edge type based on the input.
- void SetLength (double length, double segmentLength)

  Sets length of the edge in kilometers, prepares segmentation of the edge (for battery levels and car generations) and sets capacity of the road.
- double GetLength ()
- int32_t GetNumSegments ()
- PermitedRoadTypes GetRoadType ()
- char GetRoadTypeChar ()
- std::vector< BateryPair > & GetBatteryLevels ()
- void UpdateTransitTime (bool increase)

  Updates transition time based on the change of the traffic (vehicle either left or enter the edge).
- bool UpdateSegmentData (std::pair< double, double > batteryPair, int32_t startSegment, bool segment↩ Increase, int32_t endSegment)

  Updates segments information about the battery levels.

Public Attributes

- double **TransitTime**

Static Public Attributes

- static double **CapacityOverflowConstant** = 1

3.8.1 Detailed Description

Class for edge representation (road between the junctions).

3.8.2 Constructor & Destructor Documentation

3.8.2.1 Edge()

```
Edge::Edge ( )
```

Initializes the edge.

3.8.3 Member Function Documentation

3.8.3.1 AddType() [1/2]

```
void Edge::AddType (
            char type )
```

Sets proper edge type based on the input.

Parameters

type	Type of the road (supported types: `m` - motorway, `t` - trunk, `p` - primary, `o` - other).

3.8.3.2 AddType() [2/2]

```
void Edge::AddType (
            PermitedRoadTypes type )
```

Sets proper edge type based on the input.

Parameters

type	Type of the road.

3.8.3.3 GetBatteryLevels()

```
std::vector< BateryPair > & Edge::GetBatteryLevels ( )
```

Returns

Returns reference to contatiner of battery levels data of the edge.

3.8.3.4 GetLength()

```
double Edge::GetLength ( )
```

Returns

Returns length of the edge.

3.8.3.5 GetNumSegments()

```
int32_t Edge::GetNumSegments ( )
```

Returns

Returns number of segments of the edge.

3.8.3.6 GetRoadType()

```
PermitedRoadTypes Edge::GetRoadType ( )
```

Returns

Returns type of the road.

3.8.3.7 GetRoadTypeChar()

```
char Edge::GetRoadTypeChar ( )
```

Returns

Returns type of the road in char representation (`m` - motorway, `t` - trunk, `p` - primary, `o` - other).

3.8.3.8 ResetSimulation()

```
void Edge::ResetSimulation ( )
```

Resets all simulation information.

3.8.3.9 SetLength()

```
void Edge::SetLength (
            double length,
            double segmentLength )
```

Sets length of the edge in kilometers, prepares segmentation of the edge (for battery levels and car generations) and sets capacity of the road.

Parameters

length	Length of the edge in kilometers.

3.8.3.10 UpdateSegmentData()

```
bool Edge::UpdateSegmentData (
            std::pair< double, double > batteryPair,
            int32_t startSegment,
            bool segmentIncrease,
            int32_t endSegment )
```

Updates segments information about the battery levels.

Parameters

batteryPair	Pair of start and end battery levels.
startSegment	Index of the start segment.
segmentIncrease	Flag if vehicle moving in increasing order of the segment indices `true` or decreasing order `false`.
endSegment	Index of the end segment (if vehicle movement ends in the middle of the edge), (if -1, then automaticaly choosen the appropriate end of the edge).

Returns

Returns `true` if update was successful, else `false`.

3.8.3.11 UpdateTransitTime()

```
void Edge::UpdateTransitTime (
            bool increase )
```

Updates transition time based on the change of the traffic (vehicle either left or enter the edge).

Parameters

increase	If vehicle entered the edge `true`, else if vehicle left `false`.

The documentation for this class was generated from the following files:

- Edge.h
- Edge.cpp

3.9 found_goal Struct Reference

The documentation for this struct was generated from the following file:

- Map.h

3.10 GeneticParameters Struct Reference

Parameters for the genetic algorithm.

`#include <OptimizerParameters.h>`

Public Member Functions

- **GeneticParameters** (int32_t populationSize, int32_t numGenerations, int32_t numBestSelection, double tournamentSelectionTreshold, double mutationTreshold, double memberSizeVariance)

Public Attributes

- int32_t **PopulationSize_**
- int32_t **NumGenerations_**
- int32_t **NumBestSelection_**
- double **TournamentSelectionTreshold_**
- double **MutationTreshold_**
- double **MemberSizeVariance_**

3.10.1 Detailed Description

Parameters for the genetic algorithm.

The documentation for this struct was generated from the following file:

- OptimizerParameters.h

3.11 GraphAdjuster Class Reference

Class to do preprocessing of the graph.

`#include <GraphAdjuster.h>`

Public Member Functions

- void AddCityCoordinate (double lat, double lon)

  Adds City coordinates to the proper container.
- std::vector< City > & GetCitiesCoordinates ()
- std::vector< AddedEdge > GetAddedEdges ()

  Moves added edges container to the user. From that moment it is not permited to use any function from `GraphAdjuster` which uses container `this->addedEdges_`. Typical usage is right before destroying `this` object to retrieve information about added edges.
- std::vector< std::vector< double > > GetCitiesDistances ()

  Moves container of the distances between each city pair to the user. After calling this function it is not permited to use any function from `GraphAdjuster` which uses container `this->citiesDistances_`. Typical usage is right before destroying `this` object to retrieve information about cities distances.
- void ComputeCitiesDistances ()

  Computes distances between all pairs of the cities.
- bool MergeTwoComponents (Map &map, int32_t numComponents, std::vector< vertex_t > &&vertices↩ Components)

  Finds closest component from the component with index 0 and merges them. Distance is computed with the closest cities of each component and distances of the closest vertices from the cities center. Merges components with the new edge of type `OTHER` between the closest vertices of the components with distance: `distance = cities↩ _distance + vertex1_city_distance + vertex2_city_distance`
- bool MergeDegreeTwoVertices (Map &map)

  For every vertex of degree 2 conected with edges of the same road type, merges its edges (to avoid current vertex) and finally deletes the vertex.
- bool MergeDegreeTwoVertex (vertex_t vertex_to_merge, Map &map)

  Tries to merge given vertex (if degree two and the same road type).

3.11.1 Detailed Description

Class to do preprocessing of the graph.

3.11.2 Member Function Documentation

3.11.2.1 AddCityCoordinate()

```
void GraphAdjuster::AddCityCoordinate (
            double lat,
            double lon )
```

Adds City coordinates to the proper container.

Parameters

lat	Latitude of the city.
lon	Longitude of the city.

3.11.2.2 ComputeCitiesDistances()

```
void GraphAdjuster::ComputeCitiesDistances ( )
```

Computes distances between all pairs of the cities.

3.11.2.3 GetAddedEdges()

```
std::vector< AddedEdge > GraphAdjuster::GetAddedEdges ( )
```

Moves added edges container to the user. From that moment it is not permited to use any function from `GraphAdjuster` which uses container `this->addedEdges_`. Typical usage is right before destroying `this` object to retrieve information about added edges.

Returns

Return container of all added edges to the graph in order to merge all graph components.

3.11.2.4 GetCitiesCoordinates()

```
std::vector< City > & GraphAdjuster::GetCitiesCoordinates ( )
```

Returns

Returns reference to coordinates of all cities.

3.11.2.5 GetCitiesDistances()

```
std::vector< std::vector< double > > GraphAdjuster::GetCitiesDistances ( )
```

Moves container of the distances between each city pair to the user. After calling this function it is not permited to use any function from `GraphAdjuster` which uses container `this->citiesDistances_`. Typical usage is right before destroying `this` object to retrieve information about cities distances.

Returns

Returns container of the distanes between each pair of cities.

3.11.2.6 MergeDegreeTwoVertex()

```
bool GraphAdjuster::MergeDegreeTwoVertex (
            vertex_t vertex_to_merge,
            Map & map )
```

Tries to merge given vertex (if degree two and the same road type).

Parameters

vertex_to_merge	Vertex id to be merged.
map	Map object where to do the graph changes.

Returns

Returns `true` if vertex should be deleted (network was reconected), `false` do not delete the node.

3.11.2.7 MergeDegreeTwoVertices()

```
bool GraphAdjuster::MergeDegreeTwoVertices (
            Map & map )
```

For every vertex of degree 2 conected with edges of the same road type, merges its edges (to avoid current vertex) and finally deletes the vertex.

Parameters

map	Map object where to do the graph changes.

3.11.2.8 MergeTwoComponents()

```
bool GraphAdjuster::MergeTwoComponents (
            Map & map,
            int32_t numComponents,
            std::vector< vertex_t > && verticesComponents )
```

Finds closest component from the component with index 0 and merges them. Distance is computed with the closest cities of each component and distances of the closest vertices from the cities center. Merges components with the new edge of type `OTHER` between the closest vertices of the components with distance: `distance = cities_distance + vertex1_city_distance + vertex2_city_distance`

Parameters

graph	Graph to work with.
numComponents	Number of components of the graph.
verticesComponents	Vector of IDs of each vertex from the component.

Returns

Returns `true` if merging was sucessfull, else `false`.

The documentation for this class was generated from the following files:

- GraphAdjuster.h
- GraphAdjuster.cpp

3.12 GreedyParameters Struct Reference

Parameters for the greedy algorithm.

`#include <OptimizerParameters.h>`

Public Member Functions

- **GreedyParameters** (int32_t maxIterations, int32_t numThrowAway)

Public Attributes

- int32_t **MaxIterations_**
- int32_t **NumThrowAway_**

3.12.1 Detailed Description

Parameters for the greedy algorithm.

The documentation for this struct was generated from the following file:

- OptimizerParameters.h

3.13 KMeansParameters Struct Reference

Parameters for the K-Means optimalization.

`#include <OptimizerParameters.h>`

Public Member Functions

- **KMeansParameters** (int32_t numIterationsOneRun, int32_t numGenerations)

Public Attributes

- int32_t **NumIterationsOneRun_**
- int32_t **NumGenerations_**

3.13.1 Detailed Description

Parameters for the K-Means optimalization.

The documentation for this struct was generated from the following file:

- OptimizerParameters.h

3.14 location Struct Reference

Public Attributes

- double **Latitude**
- double **Longitude**

The documentation for this struct was generated from the following file:

- Node.h

3.15 Map Class Reference

Class to store and do basic operation with map objects.

`#include <Map.h>`

Public Member Functions

- Map (int32_t numClosestStations, double segmentLength)

  Initializes map object.
- void InitHeuristicLocations ()

  Initializes vector of locations for each vertex (for a-star heuristic function).
- void ResetSimulation ()

  Resets simulation information in the object (deletes charging stations and info about them and resets edges information about its battery levels).
- void AddNode (int32_t newID, double latitude, double longitude, int32_t cityID, double distance, int32_t oldID=-1)

  Adds node on the graph and sets its properties.
- void AddEdge (int32_t firstID, int32_t secondID, char type, double length)

  Adds edge on the map and sets its properties.
- void AddEdge (vertex_t firstID, vertex_t secondID, PermitedRoadTypes type, double length)

  Adds edge on the map and sets its properties.
- void AddCity (int32_t cityID, int32_t population)

  Adds population of the given city into the desired storage. If city already initialized, then rewrites the info. If cities with lower cityID not defined, defines it with zero population.
- void AddChargingStation (ChargingStation chargingStation)

  Adds given charging station object to the container of all charging stations of the map.
- void SetCitiesDistances (Double2DMatrix citiesDistances)

Stores cities distances information into proper variable.

- const Graph & GetConstGraph ()
- Graph & GetGraph ()
- const Node & GetNode (int32_t nodeID)
- std::vector< std::unique_ptr< ChargingStation > > & GetChargingStations ()
- std::unique_ptr< ChargingStation > & GetChargingStation (int32_t stationID)
- double GetCityPairDistance (int32_t firstCityID, int32_t secondCityID)
- const std::vector< int32_t > & GetCitiesPopulation ()
- const std::vector< vertex_t > & GetCityNodes (int32_t cityID)
- std::unique_ptr< ChargingStation > & GetLastChargingStation ()
- std::map< edge_t, std::vector< BateryPair > & > GetAllSegments ()
- bool ExistNode (int32_t nodeID)

  Checks whether node with given ID exists.
- double ComputeSphericalDistance (double firstLatitude, double firstLongitude, double secondLatitude, double secondLongitude)

  Computes distance between 2 cities based on longitude and latitude (using the 'Haversine' formula).
- void FindCityNodes ()

  Finds all nodes which belongs to the city and stores them into proper container.
- void FindCitiesNearestChargingStations ()

  Finds `this->numClosestStations_` nearest stations for each city in the map and stores them in ascending order (by distance from city) in the proper container. For path searching through charging stations.
- MapPosition FindClosestPositionToCoordinates (double latitude, double longitude)

  Finds the closest vertex from the given coordinates, then its neighbor closest to the given coordinates and on the edge connecting these two vertices randomly generates segment closer to the closest vertex. It (partly randomly) approximates the position of the given coordinates (for K-Means algorithm and finding centroids there).
- LengthPathPair FindVehiclePath (vertex_t start, vertex_t end, bool goCharging)

  Finds estimated shortest path between two given vertices. If we want to go charging, then finds estimated shorted path through some charging station (returns only path to the charging station, rest of the path should be computed after the vehicle is charged).
- LengthPathPair FindShortestPath (const vertex_t &start, const vertex_t &goal)

  Finds shortest path between two choosen vertices using A-Star algorithm.
- std::vector< double > DijkstraFindDistances (vertex_t start)

  Performs Dijkstra algorithm to find distance of each vertex from the start vertex.
- void TestComponents (std::tuple< int32_t, std::vector< vertex_t >> &numComponentsTuple)

  Finds components of the graph.
- void SimulateVehiclePassedThroughEdge (edge_t edge, std::pair< double, double > batteryPair, int32_t startSegment, bool segmentIncrease, int32_t endSegment)

  Simulates vehicle passage through the given edge (updates battery usage info).

3.15.1 Detailed Description

Class to store and do basic operation with map objects.

3.15.2 Constructor & Destructor Documentation

3.15.2.1 Map()

```
Map::Map (
            int32_t numClosestStations,
            double segmentLength )
```

Initializes map object.

Parameters

numClosestStations	Number of closest charging station to conside while planning the vehicle route.
segmentLength	Length of the segment.

3.15.3 Member Function Documentation

3.15.3.1 AddChargingStation()

```
void Map::AddChargingStation (
            ChargingStation chargingStation )
```

Adds given charging station object to the container of all charging stations of the map.

Parameters

chargingStation	Object representing charging station to be added.

3.15.3.2 AddCity()

```
void Map::AddCity (
            int32_t cityID,
            int32_t population )
```

Adds population of the given city into the desired storage. If city already initialized, then rewrites the info. If cities with lower cityID not defined, defines it with zero population.

Parameters

cityID	ID of the city to add.
population	Population of the city.

3.15.3.3 AddEdge() [1/2]

```
void Map::AddEdge (
            int32_t firstID,
            int32_t secondID,
            char type,
            double length )
```

Adds edge on the map and sets its properties.

Parameters

firstID	ID of the first node of the edge.
secondID	ID of the second node of the edge.
type	Type of the edge (`m` - motorway, `t` - trunk, `p` - primary, `o` - other).
length	Length of the edge (in kilometers).

3.15.3.4 AddEdge() [2/2]

```
void Map::AddEdge (
            vertex_t firstID,
            vertex_t secondID,
            PermitedRoadTypes type,
            double length )
```

Adds edge on the map and sets its properties.

Parameters

firstID	ID of the first node of the edge.
secondID	ID of the second node of the edge.
type	Road type of the edge.
length	Length of the edge (in kilometers).

3.15.3.5 AddNode()

```
void Map::AddNode (
            int32_t newID,
            double latitude,
            double longitude,
            int32_t cityID,
            double distance,
            int32_t oldID = -1 )
```

Adds node on the graph and sets its properties.

Parameters

newID	Current ID of the node.
latitude	Latitude of the node in the real world.
longitude	longitude of the node in the real world.
cityID	ID of the nearest city of the node.
distance	Distance between the node and its nearest city.
oldID	Original ID of the node (from the source map), if not specified `-1` by default.

3.15.3.6 ComputeSphericalDistance()

```
double Map::ComputeSphericalDistance (
            double firstLatitude,
            double firstLongitude,
            double secondLatitude,
            double secondLongitude )
```

Computes distance between 2 cities based on longitude and latitude (using the 'Haversine' formula).

Parameters

firstCity	First city coordinates.
secondCity	Second city coordinates.

Returns

Returns distance between 2 cities in kilometers.

3.15.3.7 DijkstraFindDistances()

```
std::vector< double > Map::DijkstraFindDistances (
            vertex_t start )
```

Performs Dijkstra algorithm to find distance of each vertex from the start vertex.

Parameters

start	Start vertex to compute distance from.

Returns

Returns container of distances of each vertex from the `start` vertex.

3.15.3.8 ExistNode()

```
bool Map::ExistNode (
            int32_t nodeID )
```

Checks whether node with given ID exists.

Parameters

nodeID	Node ID to check.

Returns

Returns `true` if node exists, else `false`.

3.15.3.9 FindCitiesNearestChargingStations()

```
void Map::FindCitiesNearestChargingStations ( )
```

Finds `this->numClosestStations_` nearest stations for each city in the map and stores them in ascending order (by distance from city) in the proper container. For path searching through charging stations.

3.15.3.10 FindCityNodes()

```
void Map::FindCityNodes ( )
```

Finds all nodes which belongs to the city and stores them into proper container.

3.15.3.11 FindClosestPositionToCoordinates()

```
MapPosition Map::FindClosestPositionToCoordinates (
            double latitude,
            double longitude )
```

Finds the closest vertex from the given coordinates, then its neighbor closest to the given coordinates and on the edge connecting these two vertices randomly generates segment closer to the closest vertex. It (partly randomly) approximates the position of the given coordinates (for K-Means algorithm and finding centroids there).

Parameters

latitude	Latitude of the searched position.
longitude	Longitude of the searched position.

Returns

Returns the approximately nearest position on the map from the given position.

3.15.3.12 FindShortestPath()

```
LengthPathPair Map::FindShortestPath (
            const vertex_t & start,
            const vertex_t & goal )
```

Finds shortest path between two choosen vertices using A-Star algorithm.

Parameters

start	Start vertex.
goal	Final vertex.

Returns

Returns pair of estimated path length and reversed path of the vertices from `start` to `goal`.

3.15.3.13 FindVehiclePath()

```
LengthPathPair Map::FindVehiclePath (
            vertex_t start,
            vertex_t end,
            bool goCharging )
```

Finds estimated shortest path between two given vertices. If we want to go charging, then finds estimated shorted path through some charging station (returns only path to the charging station, rest of the path should be computed after the vehicle is charged).

Parameters

start	Start vertex of the path.
end	Final vertex of the path.
goCharging	Flag if car should go charging then `true`, else `false`.

Returns

Returns the pair of estimated relative path duration and the container of the reversed path.

3.15.3.14 GetAllSegments()

```
std::map< edge_t, std::vector< BateryPair > & > Map::GetAllSegments ( )
```

Returns

Returns map of edge identifier and its corresponding battery levels data.

3.15.3.15 GetChargingStation()

```
std::unique_ptr< ChargingStation > & Map::GetChargingStation (
            int32_t stationID )
```

Parameters

stationID	ID of the wanted charging station.

Returns

Returns reference to the charging station with given ID.

3.15.3.16 GetChargingStations()

```
std::vector< std::unique_ptr< ChargingStation > > & Map::GetChargingStations ( )
```

Returns

Returns reference to the container of all charging stations.

3.15.3.17 GetCitiesPopulation()

```
const std::vector< int32_t > & Map::GetCitiesPopulation ( )
```

Returns

Returns const reference to each city population container.

3.15.3.18 GetCityNodes()

```
const std::vector< vertex_t > & Map::GetCityNodes (
            int32_t cityID )
```

Parameters

cityID	City ID to get vertices from.

Returns

Returns constant reference to the container of all city nodes for the given city.

3.15.3.19 GetCityPairDistance()

```
double Map::GetCityPairDistance (
            int32_t firstCityID,
            int32_t secondCityID )
```

Parameters

firstCityID	ID of the first city.
secondCityID	ID of the second city.

Returns

Returns distance between two cities.

3.15.3.20 GetConstGraph()

```
const Graph & Map::GetConstGraph ( )
```

Returns

Returns constant reference to graph.

3.15.3.21 GetGraph()

```
Graph & Map::GetGraph ( )
```

Returns

Returns reference to graph of the map.

3.15.3.22 GetLastChargingStation()

```
std::unique_ptr< ChargingStation > & Map::GetLastChargingStation ( )
```

Returns

Returns reference to charging station which was used in the last route planning through charging station (typicaly used to get information about position of the station for vehicle object).

3.15.3.23 GetNode()

```
const Node & Map::GetNode (
            int32_t nodeID )
```

Parameters

nodeID	ID of the node to get reference.

Returns

Returns const reference to the node with given ID.

3.15.3.24 InitHeuristicLocations()

```
void Map::InitHeuristicLocations ( )
```

Initializes vector of locations for each vertex (for a-star heuristic function).

3.15.3.25 ResetSimulation()

```
void Map::ResetSimulation ( )
```

Resets simulation information in the object (deletes charging stations and info about them and resets edges information about its battery levels).

3.15.3.26 SetCitiesDistances()

```
void Map::SetCitiesDistances (
            Double2DMatrix citiesDistances )
```

Stores cities distances information into proper variable.

Parameters

citiesDistances	Matrix of distances between each pair of the city (to be stored in the proper variable).

3.15.3.27 SimulateVehiclePassedThroughEdge()

```
void Map::SimulateVehiclePassedThroughEdge (
            edge_t edge,
            std::pair< double, double > batteryPair,
            int32_t startSegment,
```

```
            bool segmentIncrease,
            int32_t endSegment )
```

Simulates vehicle passage through the given edge (updates battery usage info).

Parameters

edge	Identifier of the edge to be updated.
batteryPair	Pair of in and out battery level.
startSegment	Starting segment of the vehicle.
segmentIncrease	Flag whether vehicle increasing in segments (`true` if so, else `false`).
endSegment	End Segment of the path (if `-1` then automaticaly finds appropriate end of the edge).

3.15.3.28 TestComponents()

```
void Map::TestComponents (
            std::tuple< int32_t, std::vector< vertex_t >> & numComponentsTuple )
```

Finds components of the graph.

Parameters

numComponentsTuple	Tuple where to store number of components (first parameter) and vector of component IDs for each vertex.

The documentation for this class was generated from the following files:

- Map.h
- Map.cpp

3.16 MapPosition Class Reference

Class to store info about the position in the map (edge and its segment).

`#include <MapPosition.h>`

Public Member Functions

- MapPosition ()

  Intitilizes object.
- MapPosition (vertex_t firstVertex, vertex_t secondVertex, int32_t segment, Graph &graph)

  Intitilizes object.
- void ChangeParameters (vertex_t closer, vertex_t further, int32_t segment, Graph &graph)

  Changes parameters of the object and computes other (if specified).
- double GetDistanceFromCloserVertex (Graph &graph, double segmentLength)

  Computes distance from the closer vertex. Distance is only approximate (there could be difference of 1 segment length). We assume this function will be used only as a rough estimate of the distance from closer vertex.

Public Attributes

- edge_t **EdgeID**
- vertex_t **CloserVertexID**
- vertex_t **FurtherVertexID**
- int32_t **EdgeSegmentID**

3.16.1 Detailed Description

Class to store info about the position in the map (edge and its segment).

3.16.2 Constructor & Destructor Documentation

3.16.2.1 MapPosition() [1/2]

```
MapPosition::MapPosition ( )
```

Intitilizes object.

3.16.2.2 MapPosition() [2/2]

```
MapPosition::MapPosition (
            vertex_t firstVertex,
            vertex_t secondVertex,
            int32_t segment,
            Graph & graph )
```

Intitilizes object.

Parameters

firstVertex	ID of the first vertex of the edge.
secondVertex	ID of the second vertex of the edge.
segment	ID of the segment on the edge.
graph	Graph object where the position is (to get edge ID).

3.16.3 Member Function Documentation

3.16.3.1 ChangeParameters()

```
void MapPosition::ChangeParameters (
            vertex_t closer,
            vertex_t further,
            int32_t segment,
            Graph & graph )
```

Changes parameters of the object and computes other (if specified).

Parameters

closer	New closer vertex ID.
further	New further vertex ID.
segment	New segment ID (if -1 then let it precompute by the function (the last or the first segment)).
graph	Graph object to get all necessary info.

3.16.3.2 GetDistanceFromCloserVertex()

```
double MapPosition::GetDistanceFromCloserVertex (
            Graph & graph,
            double segmentLength )
```

Computes distance from the closer vertex. Distance is only approximate (there could be difference of 1 segment length). We assume this function will be used only as a rough estimate of the distance from closer vertex.

Parameters

graph	Graph object to get the distance from.
segmentLength	Length of one segment of the edge.

Returns

Returns estimated distance of the position from the closer vertex.

The documentation for this class was generated from the following files:

- MapPosition.h
- MapPosition.cpp

3.17 MapReader Class Reference

Public Member Functions

- MapReader ()

  Initializes the reader.

- bool LoadMap (Map &map, SimulationParameters &simulationParameters)

  Loads map from the file streams from given parameters in the appropriate format.
- bool LoadNodesCities (Map &map, std::istream &nodeCityStream)

  Loads node-city pairs from the input.
- bool LoadEdges (Map &map, std::istream &edgeStream)

  Loads edges from the input.
- bool LoadCities (Map &map, std::istream &cityStream)

  Loads cities from the input.
- bool LoadChargingStations (Map &map, std::istream &chargingStationsStream)

  Loads charging stations.
- bool WriteAddedEdges (std::ostream &addedEdgesStream)

  Writes added edges used for components merging. In format: `first_vertex second_vertex length type`
- bool WritePreparedCombinedNodes (Map &map, std::ostream &preparedNodesStream)

  Writes preprocessed combined nodes (after removing degree 2 vertices from the original graph). In format: `node_ID latitude longitude nearest_city_ID city_distance`
- bool WritePreparedEdges (Map &map, std::ostream &preparedEdgesStream)

  Writes preprocessed edges (after removing degree 2 vertices from the original graph). In format: `node_id_1 node_id_2 length road_type`
- bool WriteStations (Map &map, std::ostream &stationsStream)

  Writes all charging stations representations into the given stream. In format: `station_ID closer_vertex further_vertex segment capacity city_ID waiting_time estimated_charging`
- GraphAdjuster & GetGraphAdjuster ()

3.17.1 Constructor & Destructor Documentation

3.17.1.1 MapReader()

```
MapReader::MapReader ( )
```

Initializes the reader.

3.17.2 Member Function Documentation

3.17.2.1 GetGraphAdjuster()

```
GraphAdjuster & MapReader::GetGraphAdjuster ( )
```

Returns

Returns reference to currently used `GraphAdjuster` object.

3.17.2.2 LoadChargingStations()

```
bool MapReader::LoadChargingStations (
            Map & map,
            std::istream & chargingStationsStream )
```

Loads charging stations.

Parameters

map	Object to store info about the stations.
chargingStationsStream	Input stream.

Returns

Returns `true` if loading was successful, else `false`.

3.17.2.3 LoadCities()

```
bool MapReader::LoadCities (
            Map & map,
            std::istream & cityStream )
```

Loads cities from the input.

Parameters

map	Object to store info about the cities.
cityStream	Input stream.

Returns

Returns `true` if loading was successful, else `false`.

3.17.2.4 LoadEdges()

```
bool MapReader::LoadEdges (
            Map & map,
            std::istream & edgeStream )
```

Loads edges from the input.

Parameters

map	Object to store info about the edges.
edgeStream	Input stream.

Returns

Returns `true` if loading was successful, else `false`.

3.17.2.5 LoadMap()

```
bool MapReader::LoadMap (
            Map & map,
            SimulationParameters & simulationParameters )
```

Loads map from the file streams from given parameters in the appropriate format.

Parameters

map	Object of the map to store the loaded information.
simulationParameters	Object which clusters all simulation parameters.

Returns

Return `true` if loading was successful, else `false` (error happened).

3.17.2.6 LoadNodesCities()

```
bool MapReader::LoadNodesCities (
            Map & map,
            std::istream & nodeCityStream )
```

Loads node-city pairs from the input.

Parameters

map	Object to store info about node-city pair.
nodeCityStream	Input stream.

Returns

Returns `true` if loading was successful, else `false`.

3.17.2.7 WriteAddedEdges()

```
bool MapReader::WriteAddedEdges (
            std::ostream & addedEdgesStream )
```

Writes added edges used for components merging. In format: `first_vertex second_vertex length type`

Parameters

addedEdgesStream	Stream to write info about added adges.

Returns

Returns `true` if writing was successful, else `false`.

3.17.2.8 WritePreparedCombinedNodes()

```
bool MapReader::WritePreparedCombinedNodes (
            Map & map,
            std::ostream & preparedNodesStream )
```

Writes preprocessed combined nodes (after removing degree 2 vertices from the original graph). In format: `node_ID latitude longitude nearest_city_ID city_distance`

Parameters

map	Map object to get graph info.
preparedNodesStream	Stream to write node-city info.

Returns

Returns `true` if writing was successful, else `false`.

3.17.2.9 WritePreparedEdges()

```
bool MapReader::WritePreparedEdges (
            Map & map,
            std::ostream & preparedEdgesStream )
```

Writes preprocessed edges (after removing degree 2 vertices from the original graph). In format: `node_id_1 node_id_2 length road_type`

Parameters

map	Map object to get graph info.
preparedEdgesStream	Stream to write edges info.

Returns

Returns `true` if writing was successful, else `false`.

3.17.2.10 WriteStations()

```
bool MapReader::WriteStations (
            Map & map,
            std::ostream & stationsStream )
```

Writes all charging stations representations into the given stream. In format: `station_ID closer_vertex further_vertex segment capacity city_ID waiting_time estimated_charging`

Parameters

map	Map object to load charging stations from.
stationsStream	Stream to write charging stations configurations.

Returns

Returns `true` if writing was successfull, else `false`.

The documentation for this class was generated from the following files:

- MapReader.h
- MapReader.cpp

3.18 ModelRepresentation Class Reference

Class to represent one model.

`#include <ModelRepresentation.h>`

Public Member Functions

- ModelRepresentation ()
  Initializes the object.
- ModelRepresentation (double loss, std::vector< StationParameters > allStations)
  Initilizes the object.
- void SaveModelRepresentation (std::vector< std::unique_ptr< ChargingStation >> &allStations)
  Stores representation of the given model to `this` object.

Public Attributes

- double **Loss_**
- std::vector< StationParameters > **AllChargingStations_**

3.18.1 Detailed Description

Class to represent one model.

3.18.2 Constructor & Destructor Documentation

3.18.2.1 ModelRepresentation() [1/2]

```
ModelRepresentation::ModelRepresentation ( )
```

Initializes the object.

3.18.2.2 ModelRepresentation() [2/2]

```
ModelRepresentation::ModelRepresentation (
            double loss,
            std::vector< StationParameters > allStations )
```

Initilizes the object.

Parameters

loss	Loss of the model.
allStations	Vector of all charging stations of the model.

3.18.3 Member Function Documentation

3.18.3.1 SaveModelRepresentation()

```
void ModelRepresentation::SaveModelRepresentation (
            std::vector< std::unique_ptr< ChargingStation >> & allStations )
```

Stores representation of the given model to `this` object.

Parameters

allStations	Container of all charging stations of the model.

The documentation for this class was generated from the following files:

- ModelRepresentation.h
- ModelRepresentation.cpp

3.19 MyGreater Struct Reference

Public Member Functions

- bool **operator()** (const TableEvent &lhs, const TableEvent &rhs)

The documentation for this struct was generated from the following file:

- TimeTable.h

3.20 Node Class Reference

Class representing road junction.

`#include <Node.h>`

Public Member Functions

- Node ()
  Initializes the object.
- Node (int32_t newID, int32_t oldID, int32_t cityID, double cityDistance, double latitude, double longitude)
  Initializes the object.

Public Attributes

- int32_t **NewID_**
- int32_t **OldID_**
- int32_t **CityID_**
- double **CityDistance_**
- location **Location_**

3.20.1 Detailed Description

Class representing road junction.

3.20.2 Constructor & Destructor Documentation

3.20.2.1 Node() [1/2]

```
Node::Node ( )
```

Initializes the object.

3.20.2.2 Node() [2/2]

```
Node::Node (
            int32_t newID,
            int32_t oldID,
            int32_t cityID,
            double cityDistance,
            double latitude,
            double longitude )
```

Initializes the object.

Parameters

newID	Current node ID.
oldID	Old node ID (in the original map). To potentially project the node to the real map).
cityID	ID of the nearest city.
cityDistance	Distance between the node and its nearest city center.
latitude	Node latitude.
longitude	Node longitude.

The documentation for this class was generated from the following files:

- Node.h
- Node.cpp

3.21 Optimizer Class Reference

Class to optimize number of charging stations and its locations.

`#include <Optimizer.h>`

Public Member Functions

- Optimizer (SimulationParameters simulationParameters, OptimizerParameters optimizerParamters)

  Initilizes the optimizer object.
- double ModelLoss (double stationNumberParameter, double runDownParameter, double durationParameter, double batteryDifferenceParameter, double waitingTimesParameter)

  Computes loss of the current station model. Score is just linear combination of:
- void RunMultipleSimulations (int16_t numIterations, bool logs)

  Runs traffic simulation multiple times with randomly placed charing stations.
- void GreedyAlgorithm (bool logs)

  Optimizes charging station positions and its number based on the station usage and positions of the battery run down vehicles. Maintains all at least once time used stations and rest randomly generates on the positions of battery run down and also some stations just doesn't use (to optimize number of stations too).
- void GeneticAlgorithm (bool logs)

  Optimizes charging stations position and number using the genetic algorithm approach. One model is considered to be an individual. All operations works only with the loss (fitness) and all charging stations of the model. For selection we take given number of best models and rest of the population we choose using the tournament selection. We then use one point crossover (changes stations from the given index in the vector of all charging stations). And in mutation we randomly delete stations and replace them with randomly generated one or we randomly delete or add some stations (mutation in position and also in number of stations).
- void KMeansAlgorithm (bool logs)

  Optimizes the position of the charging stations using the algorithm inspired by K-Means algorithm. Randomly generates centroids finds its clusters (approximately using Dijsktra algorithm) and then approximately computes new clusters (using geographical coordinates of the nodes).

3.21.1 Detailed Description

Class to optimize number of charging stations and its locations.

3.21.2 Constructor & Destructor Documentation

3.21.2.1 Optimizer()

```
Optimizer::Optimizer (
            SimulationParameters simulationParameters,
            OptimizerParameters optimizerParameters )
```

Initilizes the optimizer object.

Parameters

simulationParameters	Parameters of the traffic simulator.
optimizerParameters	Parameters of the optimizer.

3.21.3 Member Function Documentation

3.21.3.1 GeneticAlgorithm()

```
void Optimizer::GeneticAlgorithm (
            bool logs )
```

Optimizes charging stations position and number using the genetic algorithm approach. One model is considered to be an individual. All operations works only with the loss (fitness) and all charging stations of the model. For selection we take given number of best models and rest of the population we choose using the tournament selection. We then use one point crossover (changes stations from the given index in the vector of all charging stations). And in mutation we randomly delete stations and replace them with randomly generated one or we randomly delete or add some stations (mutation in position and also in number of stations).

Parameters

logs	Whether to write simulator logs.

3.21.3.2 GreedyAlgorithm()

```
void Optimizer::GreedyAlgorithm (
            bool logs )
```

Optimizes charging station positions and its number based on the station usage and positions of the battery run down vehicles. Maintains all at least once time used stations and rest randomly generates on the positions of battery run down and also some stations just doesn't use (to optimize number of stations too).

Parameters

logs	Whether to write simulator logs.

3.21.3.3 KMeansAlgorithm()

```
void Optimizer::KMeansAlgorithm (
            bool logs )
```

Optimizes the position of the charging stations using the algorithm inspired by K-Means algorithm. Randomly generates centroids finds its clusters (approximately using Dijsktra algorithm) and then approximately computes new clusters (using geographical coordinates of the nodes).

Parameters

logs	Whether to write simulator logs.

3.21.3.4 ModelLoss()

```
double Optimizer::ModelLoss (
            double stationNumberParameter,
            double runDownParameter,
            double durationParameter,
            double batteryDifferenceParameter,
            double waitingTimesParameter )
```

Computes loss of the current station model. Score is just linear combination of:

(number_of_charging_stations, number_of_battery_run_downs, average_travel_duration, average_battery_↩ difference (difference between start and end battery level (negative - battery increase, positive - battery decrease)), average_waiting_time)

Each part of the linear combination has its apropriate multiplication contants.

Parameters

stationNumberParameter	Multiplication constant of the number of charging stations.
runDownParameter	Multiplication constant of the number of run down vehicles.
durationParameter	Multiplication constant of the average duration (in minutes).
batteryDifferenceParameter	Multiplication constant of the average battery level difference (values from -1 to 1 (negative values means battery level increases, positive - decreases)).
waitingTimesParameter	Multiplication constant for average waiting time in the charging station.

Returns

Returns loss value of the current charging stations model.

3.21.3.5 RunMultipleSimulations()

```
void Optimizer::RunMultipleSimulations (
            int16_t numIterations,
            bool logs )
```

Runs traffic simulation multiple times with randomly placed charing stations.

Parameters

numIterations	Number of simulations to run.
logs	Whether to write simulator logs.

The documentation for this class was generated from the following files:

- Optimizer.h
- Optimizer.cpp

3.22 OptimizerParameters Class Reference

Public Member Functions

- OptimizerParameters ()

  Initializes the optimizer parameters.
- OptimizerParameters (GreedyParameters greedyParameters, GeneticParameters geneticParameters, KMeansParameters kMeansParameters, double scoreDifference, double stationNumberParameter, double runDownParameter, double durationParameter, double batteryDifferenceParameter, double waitingTimes↩ Parameter)

  Initializes the optimizer parameters.

Public Attributes

- GreedyParameters **GreedyParameters_**
- GeneticParameters **GeneticParameters_**
- KMeansParameters **KMeansParameters_**
- double **ScoreDifferenceTreshold_**
- double **StationNumberParameter_**
- double **RunDownParameter_**
- double **DurationParameter_**
- double **BatteryDifferenceParameter_**
- double **WaitingTimesParameter_**

3.22.1 Constructor & Destructor Documentation

3.22.1.1 OptimizerParameters() [1/2]

```
OptimizerParameters::OptimizerParameters ( )
```

Initializes the optimizer parameters.

3.22.1.2 OptimizerParameters() [2/2]

```
OptimizerParameters::OptimizerParameters (
            GreedyParameters greedyParameters,
            GeneticParameters geneticParameters,
            KMeansParameters kMeansParameters,
            double scoreDifference,
            double stationNumberParameter,
            double runDownParameter,
            double durationParameter,
            double batteryDifferenceParameter,
            double waitingTimesParameter )
```

Initializes the optimizer parameters.

Parameters

greedyParameters	Parameters for greedy optimization.
geneticParameters	Parameters for genetic optimization.
kMeansParameters	Parameters for K-Means optimization.
scoreDifference	Treshold of score difference to continue in optimization (in some algorithms).
stationNumberParameter	Loss parameter for number of stations.
runDownParameter	Loss parameter for run down vehicles.
durationParameter	Loss parameter for travel duration.
batteryDifferenceParameter	Loss parameter for battery difference.
waitingTimesParameter	<Loss parameter for waiting times in stations./param>

The documentation for this class was generated from the following files:

- OptimizerParameters.h
- OptimizerParameters.cpp

3.23 SimulationParameters Class Reference

Class to manage program arguments.

```
#include <SimulationParameters.h>
```

Public Member Functions

- SimulationParameters ()

  Initializes simulation parameters.

Public Attributes

- bool **SavePrepared**
- int32_t **SimulationTime**
- int32_t **NumClosestStations**
- int32_t **NumStations**
- int32_t **StationCapacity**
- double **SegmentLength**
- double **CarConsumption**
- double **ExponentialLambdaCities**
- double **ExponentialLambdaDepartures**
- double **EndCityRatio**
- double **BatteryTresholdLambda**
- double **CarBatteryMean**
- double **CarBatteryDeviation**
- double **CarStartBatteryBottomLimit**
- double **ChargingTreshold**
- double **NotChargingTreshold**
- double **BatteryTolerance**
- double **CarVelocity**
- double **ChargingWaitingTime**
- double **MeanChargingLevel**
- std::string **EdgesFile**
- std::string **NodeCityFile**
- std::string **CitiesFile**
- std::string **AddedEdgesFile**
- std::string **PreparedNodesFile**
- std::string **PreparedEdgesFile**

3.23.1 Detailed Description

Class to manage program arguments.

3.23.2 Constructor & Destructor Documentation

3.23.2.1 SimulationParameters()

```
SimulationParameters::SimulationParameters ( )
```

Initializes simulation parameters.

The documentation for this class was generated from the following files:

- SimulationParameters.h
- SimulationParameters.cpp

3.24 StationParameters Class Reference

Struct to store information about the charging station.

`#include <StationParameters.h>`

Public Member Functions

- StationParameters ()

  Initializes station parameters.
- StationParameters (int32_t stationID, MapPosition position, int32_t capacity, int32_t closestCityID, double chargingWaitingTime, double estimatedChargingLevel)

  Initializes station parameters.

Public Attributes

- int32_t **StationID_**
- MapPosition **Position_**
- int32_t **Capacity_**
- int32_t **ClosestCityID_**
- double **ChargingWaitingTime_**
- double **EstimatedChargingLevel_**

3.24.1 Detailed Description

Struct to store information about the charging station.

3.24.2 Constructor & Destructor Documentation

3.24.2.1 StationParameters() [1/2]

```
StationParameters::StationParameters ( )
```

Initializes station parameters.

3.24.2.2 StationParameters() [2/2]

```
StationParameters::StationParameters (
            int32_t stationID,
            MapPosition position,
            int32_t capacity,
            int32_t closestCityID,
            double chargingWaitingTime,
            double estimatedChargingLevel )
```

Initializes station parameters.

Parameters

stationID	ID of the station.
position	Position of the station.
capacity	Capacity of the station.
closestCityID	ID of the nearest city of the station.
chargingWaitingTime	Waiting time for full charge.
estimatedChargingLevel	Expected level to be charged on the station.

The documentation for this class was generated from the following files:

- StationParameters.h
- StationParameters.cpp

3.25 TableEvent Class Reference

Class to store info about the simulation event (for discrete simulation).

`#include <TableEvent.h>`

Public Member Functions

- TableEvent (int64_t carID, double actionTime, Actions action)
  Initializes the event object.

Public Attributes

- double **ActionTime_**
- int64_t **CarID_**
- Actions **Action_**

3.25.1 Detailed Description

Class to store info about the simulation event (for discrete simulation).

3.25.2 Constructor & Destructor Documentation

3.25.2.1 TableEvent()

```
TableEvent::TableEvent (
            int64_t carID,
            double actionTime,
            Actions action )
```

Initializes the event object.

Parameters

carID	ID of the vehicle of the event.
actionTime	Time of the event.
action	Type of the action.

The documentation for this class was generated from the following files:

- TableEvent.h
- TableEvent.cpp

3.26 TimeTable Class Reference

Class to manage all high level simulation logic.

`#include <TimeTable.h>`

Public Member Functions

- TimeTable (SimulationParameters &simulationParameters)

  Initializes the timetable.
- bool LoadMap (SimulationParameters &simulationParameters)

  Loads map of the simulator from the given source and sets proper simulator parameters.
- void RandomlyGenerateChargingStations (int32_t numStations, SimulationParameters &simulationParameters)

  Randomly generates charging stations.
- void ResetSimulation ()

  Resets all simulation information.
- void RunSimulation (int32_t numStations, SimulationParameters &simulationParameters, bool logs)

  Executes the simulation.
- void AddEvent (TableEvent tableEvent)

  Adds event to the timetable.
- const TableEvent & GetNextEvent ()
- void RunRestSimulation (SimulationParameters &simulationParameters, bool logs)

  Executes simulation after generation of first 2 cars.
- std::vector< double > & GetTravelDurations ()
- std::vector< double > & GetBatteryDifferences ()
- std::vector< std::vector< double > > & GetChargingLevels ()
- std::vector< std::pair< int32_t, double > > & GetWaitingTimes ()
- std::map< edge_t, std::vector< BateryPair > & > GetAllMapSegmentsInfo ()
- TrafficSimulator & GetTrafficSimulator ()
- std::map< int32_t, std::vector< MapPosition > > & GetRunDownPositions ()

Public Attributes

- double **ActualTime_**
- int32_t **CarsBatteryRunDown_** = 0

3.26.1 Detailed Description

Class to manage all high level simulation logic.

3.26.2 Constructor & Destructor Documentation

3.26.2.1 TimeTable()

```
TimeTable::TimeTable (
            SimulationParameters & simulationParameters )
```

Initializes the timetable.

Parameters

simulationParameters	Parameters of the simulation.

3.26.3 Member Function Documentation

3.26.3.1 AddEvent()

```
void TimeTable::AddEvent (
            TableEvent tableEvent )
```

Adds event to the timetable.

Parameters

tableEvent	Event to be added.

3.26.3.2 GetAllMapSegmentsInfo()

```
std::map< edge_t, std::vector< BateryPair > & > TimeTable::GetAllMapSegmentsInfo ( )
```

Returns

Returns map with key edge identifier and value reference to vector of battery info for each segment of the corresponding edge.

3.26.3.3 GetBatteryDifferences()

```
std::vector< double > & TimeTable::GetBatteryDifferences ( )
```

Returns

Returns reference to vector of all battery differences between start and end of the route.

3.26.3.4 GetChargingLevels()

```
std::vector< std::vector< double > > & TimeTable::GetChargingLevels ( )
```

Returns

Returns reference to vector of all charging level for each charging station.

3.26.3.5 GetNextEvent()

```
const TableEvent & TimeTable::GetNextEvent ( )
```

Returns

Returns const reference to next event to be done.

3.26.3.6 GetRunDownPositions()

```
std::map< int, std::vector< MapPosition > > & TimeTable::GetRunDownPositions ( )
```

Returns

Returns reference to all battery run down positions of each city.

3.26.3.7 GetTrafficSimulator()

```
TrafficSimulator & TimeTable::GetTrafficSimulator ( )
```

Returns

Returns reference to `TrafficSimulator` object.

3.26.3.8 GetTravelDurations()

```
std::vector< double > & TimeTable::GetTravelDurations ( )
```

Returns

Returns reference to vector of all finished travel durations of the simulation.

3.26.3.9 GetWaitingTimes()

```
std::vector< std::pair< int32_t, double > > & TimeTable::GetWaitingTimes ( )
```

Returns

Returns reference to vector of number of customers and sum of their waiting times for each charging station.

3.26.3.10 LoadMap()

```
bool TimeTable::LoadMap (
            SimulationParameters & simulationParameters )
```

Loads map of the simulator from the given source and sets proper simulator parameters.

Parameters

simulationParameters	Parameters of the simulation.

Returns

Returns `true` if loading was sucessfull, else `false`.

3.26.3.11 RandomlyGenerateChargingStations()

```
void TimeTable::RandomlyGenerateChargingStations (
            int32_t numStations,
            SimulationParameters & simulationPararemetrs )
```

Randomly generates charging stations.

Parameters

numStations	Number of station to be generated.
simulationParameters	Parameters of the simulation.

3.26.3.12 ResetSimulation()

```
void TimeTable::ResetSimulation ( )
```

Resets all simulation information.

3.26.3.13 RunRestSimulation()

```
void TimeTable::RunRestSimulation (
            SimulationParameters & simulationParameters,
            bool logs )
```

Executes simulation after generation of first 2 cars.

Parameters

simulationParameters	Parameters of the simulation.
logs	Whether to write simulation logs (for debugging).

3.26.3.14 RunSimulation()

```
void TimeTable::RunSimulation (
            int32_t numStations,
            SimulationParameters & simulationParameters,
            bool logs )
```

Executes the simulation.

Parameters

numStations	Number of charging stations.
simulationParameters	Parameters of the simulation.
logs	Whether to write simulation logs (for debugging).

The documentation for this class was generated from the following files:

- TimeTable.h
- TimeTable.cpp

3.27 TrafficSimulator Class Reference

Class for managing simulator operations.

`#include <TrafficSimulator.h>`

Public Member Functions

- TrafficSimulator (int32_t numClosestStations, double segmentLength)

  Intitilizes the object.
- bool LoadMap (SimulationParameters &simulationParameters)

  Load map and stores all other necessary precomputed data (precomputed distances between each pair of cities).
- void ResetSimulation ()

  Reset simulation parameters.
- void GenerateChargingStations (int32_t numStations, SimulationParameters &simulationParameters)

  Randomly generates charging stations (based on city probability and its distance from the center of the city) and stores it.
- ChargingStation GenerateChargingStation (int32_t stationID, SimulationParameters simulationParameters)

  Randomly generates charging station.
- int64_t GenerateCar (double startTime, SimulationParameters &simulationParameters)

  Generates car and adds it to the vector of all currently used cars.
- double GenerateNextDepartureTime ()

  Generates next departure or waiting time.
- double GenerateBatteryTreshold ()

  Generates random additional part to battery treshold (for random part of the charging decision).
- void DeleteCar (int64_t CarID)

  Deletes the car from the container of all currently used cars.
- Map & GetMap ()
- CarIterator GetCarIterator (int64_t carID)

  Finds queried car and returns its reference as `CarIterator` value.

3.27.1 Detailed Description

Class for managing simulator operations.

3.27.2 Constructor & Destructor Documentation

3.27.2.1 TrafficSimulator()

```
TrafficSimulator::TrafficSimulator (
            int32_t numClosestStations,
            double segmentLength )
```

Intitilizes the object.

Parameters

numClosestStations	Number of closest charging stations to consider during optimal path finding.
segmentLength	Length of the edge segment.

3.27.3 Member Function Documentation

3.27.3.1 DeleteCar()

```
void TrafficSimulator::DeleteCar (
            int64_t carID )
```

Deletes the car from the container of all currently used cars.

Parameters

carID	ID of the car to delete.

3.27.3.2 GenerateBatteryTreshold()

```
double TrafficSimulator::GenerateBatteryTreshold ( )
```

Generates random additional part to battery treshold (for random part of the charging decision).

Returns

Returns randomly additional battery level above the bottom treshold.

3.27.3.3 GenerateCar()

```
int64_t TrafficSimulator::GenerateCar (
            double startTime,
            SimulationParameters & simulationParameters )
```

Generates car and adds it to the vector of all currently used cars.

Returns

Returns ID of the new car.

3.27.3.4 GenerateChargingStation()

```
ChargingStation TrafficSimulator::GenerateChargingStation (
            int32_t stationID,
            SimulationParameters simulationParameters )
```

Randomly generates charging station.

Parameters

stationID	ID of the station.
simulationParameters	Parameters of the simulation.

Returns

3.27.3.5 GenerateChargingStations()

```
void TrafficSimulator::GenerateChargingStations (
            int32_t numStations,
            SimulationParameters & simulationParameters )
```

Randomly generates charging stations (based on city probability and its distance from the center of the city) and stores it.

Parameters

numStations	Number of stations to generate.
simulationParameters	Parameters of the simulation.

3.27.3.6 GenerateNextDepartureTime()

```
double TrafficSimulator::GenerateNextDepartureTime ( )
```

Generates next departure or waiting time.

Returns

Returns exponentialy randomly generated time.

3.27.3.7 GetCarIterator()

```
CarIterator TrafficSimulator::GetCarIterator (
            int64_t carID )
```

Finds queried car and returns its reference as `CarIterator` value.

Parameters

carID	ID of the car to find if `-1`, then return end of the container (we want to propagate information that this object doesn't exist).

Returns

Returns iterator to the given car into the container of all currently used cars or `this->allCars_.end()` if this object doesn't exist.

3.27.3.8 GetMap()

```
Map & TrafficSimulator::GetMap ( )
```

Returns

Returns reference to map.

3.27.3.9 LoadMap()

```
bool TrafficSimulator::LoadMap (
            SimulationParameters & simulationParameters )
```

Load map and stores all other necessary precomputed data (precomputed distances between each pair of cities).

Parameters

simulationParameters	Parameters of the simulation.

Returns

Returns `true` if loading was successful, else `false`.

3.27.3.10 ResetSimulation()

```
void TrafficSimulator::ResetSimulation ( )
```

Reset simulation parameters.

The documentation for this class was generated from the following files:

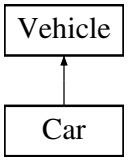
- TrafficSimulator.h
- TrafficSimulator.cpp

3.28 Vehicle Class Reference

Parent class of all vehicle objects for the simulation.

`#include <Vehicle.h>`

Inheritance diagram for Vehicle:



Public Member Functions

- Vehicle (double startTime, MapPosition start, MapPosition end, double consumption, double batteryLevel)

  Initializes object.
- double GetBatteryLevel ()
- const MapPosition & GetPosition ()
- const MapPosition & GetStationPosition ()
- const MapPosition & GetEndPosition ()
- double GetNextNodeTransitTime (Map &map, int32_t segment, bool increasing, int32_t secondSegment)

  Computes expected transition time for current traffic.
- double GetExpectedPathTransitTime (double pathLength)
- double GetExpectedEdgeTransitBatteryLevel (Map &map, int32_t segment, bool increasing, int32_t second↩ Segment)

  Computes expected battery level, if crossing the edge from given segment. Works also if we want to get to given segment, but in `incresing` should be flag for the direction from the `segment` to end of the edge.
- double GetExpectedPathBatteryLevel (double estimatedTime)

  Computes estimated battery level after reaching the path end in estimated time.
- int32_t GetChargingStationID ()
- int32_t GetStartingSegment (bool isDecreasing, Edge &edge)

  Computes starting segment ID (when not starting inside the edge). Decides whether start from the begin or the end side of the edge.
- bool IsIncreasing (vertex_t start, vertex_t end)
- bool ChangedCity (Map &map)

  Finds out whether car changed the city.
- bool GoingToStation ()
- double GetStartTime ()
- double GetStartBatteryLevel ()
- double GetStartLineWaitingTime ()
- int32_t GetPathSize ()
- bool TriedCharging ()
- bool IsInsideEdge ()
- bool IsFinishing ()
- void StartGoingCharging ()

  Sets proper flags to start going to the charging station.
- void SetStartLineWaitingTime (double startTime)

  Sets start time of the waiting in the charging station queue.
- void SetChargingTry (bool value)

  Sets whether vehicle tried to go to the charging station.

- bool GoCharging (vertex_t start, vertex_t end, Map &map, double batteryTreshold, SimulationParameters &simulationParameters)

  Decides whether go charging or not (based on the battery level, using randomness and estimated battery level in the end (if enough -> don't go charging).
- bool GoCharging (double batteryTreshold, SimulationParameters &simulationParameters)

  Decides whether go charging or not (based on the battery level, using randomness and estimated battery level in the end (if enough -> don't go charging). Works without need to compute the path.
- bool BatteryLevelToGoCharging (double bottomTreshold, double upperTreshold, double additionalTreshold)

  Decides whether it is time to charge the battery or not (starts moving to the charging station) only based on the battery level (doesn't count with the expected battery level in the end).
- void UpdateVehiclePath (vertex_t start, vertex_t end, Map &map, double batteryTreshold, bool removeFirst, SimulationParameters &simulationParameters)

  Updates vehicle path and decides whether it is neccesary to go to the charging station. If so then finds the path to the best estimated charging station.
- void ChargeBattery (double chargeLevel)

  Increases the battery level (maximal capacity is 1) and stops charging.
- bool MoveFirstSegment (Map &map, bool logs)

  Moves the vehicle from the starting position to the closest node (either starting or leaving the charging station).
- bool MoveToNextNode (Map &map, bool logs)

  Moves the vehicle to the next node (without the last and first move).
- bool MoveToFinalSegment (Map &map, bool logs)

  Moves the vehicle on the final segment of the path (moving to the middle the edge).
- void PopFirstPathVertex ()

  Removes first vertex of the path (only use when we want to remove current vertex of the path start).

Protected Attributes

- MapPosition **startPosition_**
- MapPosition **stationPosition_**
- MapPosition **endPosition_**
- MapPosition **currentPosition_**
- double **startTime_**
- double **startLineWaitingTime_**
- int32_t **chargingStationID_**
- vertex_t **previousVertex_**
- bool **insideEdge_**
- double **startBatteryLevel_**
- double **batteryConsumption_**
- double **batteryLevel_**
- double **relativeVelocity_**
- bool **headingToChargingStation_**
- bool **chargingTested_**
- std::vector< vertex_t > **path_**
- double **pathLength_**

3.28.1 Detailed Description

Parent class of all vehicle objects for the simulation.

3.28.2 Constructor & Destructor Documentation

3.28.2.1 Vehicle()

```
Vehicle::Vehicle (
            double startTime,
            MapPosition start,
            MapPosition end,
            double consumption,
            double batteryLevel )
```

Initializes object.

Parameters

startTime	Time of the departure.
start	Starting position.
end	Ending position.
consumption	Battery consumption.
batteryLevel	Start battery level.

3.28.3 Member Function Documentation

3.28.3.1 BatteryLevelToGoCharging()

```
bool Vehicle::BatteryLevelToGoCharging (
            double bottomTreshold,
            double upperTreshold,
            double additionalTreshold )
```

Decides whether it is time to charge the battery or not (starts moving to the charging station) only based on the battery level (doesn't count with the expected battery level in the end).

Parameters

batteryTreshold	Randomly generated additional battery level for decision if go charging or not (if level above then go).

Returns

Returns `true` if vehicle should go charging, else `false`.

3.28.3.2 ChangedCity()

```
bool Vehicle::ChangedCity (
            Map & map )
```

Finds out whether car changed the city.

Parameters

previous	Previous vertex.
current	Current vertex.
map	Map object to get info from.

Returns

Returns `true` if vehicle crossed the border of the city, else `false`.

3.28.3.3 ChargeBattery()

```
void Vehicle::ChargeBattery (
            double chargeLevel )
```

Increases the battery level (maximal capacity is 1) and stops charging.

Parameters

chargeLevel	Battery percentage to be charged (needs to be non-negative value).

3.28.3.4 GetBatteryLevel()

```
double Vehicle::GetBatteryLevel ( )
```

Returns

Returns current battery level.

3.28.3.5 GetChargingStationID()

```
int32_t Vehicle::GetChargingStationID ( )
```

Returns

Returns ID of the charging station where the vehicle is heading or where currently is.

3.28.3.6 GetEndPosition()

```
const MapPosition & Vehicle::GetEndPosition ( )
```

Returns

Returns end position of the vehicle.

3.28.3.7 GetExpectedEdgeTransitBatteryLevel()

```
double Vehicle::GetExpectedEdgeTransitBatteryLevel (
            Map & map,
            int32_t segment,
            bool increasing,
            int32_t secondSegment )
```

Computes expected battery level, if crossing the edge from given segment. Works also if we want to get to given segment, but in `incresing` should be flag for the direction from the `segment` to end of the edge.

Parameters

map	Map to find corresponding edge info.
segment	ID of the segment from which we are begining.
increasing	Flag if increasing in segment ID (if we want to get from end of the edge to the segment, then opposite way).
secondSegment	ID of the final segment in the edge (if movent to the end of the edge, then set attribute to `-1`)

Returns

Returns expected battery level after crossing the edge.

3.28.3.8 GetExpectedPathBatteryLevel()

```
double Vehicle::GetExpectedPathBatteryLevel (
            double estimatedTime )
```

Computes estimated battery level after reaching the path end in estimated time.

Parameters

estimatedTime	Estimated time to reach the end of the road.

Returns

Returns estimated end battery level.

3.28.3.9 GetExpectedPathTransitTime()

```
double Vehicle::GetExpectedPathTransitTime (
            double pathLength )
```

Parameters

pathLength	Lenght of the path to pass.

Returns

Returns expected transition time based on the path length.

3.28.3.10 GetNextNodeTransitTime()

```
double Vehicle::GetNextNodeTransitTime (
            Map & map,
            int32_t segment,
            bool increasing,
            int32_t secondSegment )
```

Computes expected transition time for current traffic.

Parameters

map	Map object to compute transit time from.
segment	ID of the segment from which we are begining.
increasing	Flag if increasing in segment ID (if we want to get from end of the edge to the segment, then opposite way).
secondSegment	ID of the final segment in the edge (if movent to the end of the edge, then set attribute to -1)

Returns

Returns transition time the vehicle needs to get from current node to the next node.

3.28.3.11 GetPathSize()

```
int32_t Vehicle::GetPathSize ( )
```

Returns

Returns number of vertices stored in the precomputed path.

3.28.3.12 GetPosition()

```
const MapPosition & Vehicle::GetPosition ( )
```

Returns

Returns current vehicle position.

3.28.3.13 GetStartBatteryLevel()

```
double Vehicle::GetStartBatteryLevel ( )
```

Returns

Returns battery level at the start of the path.

3.28.3.14 GetStartingSegment()

```
int32_t Vehicle::GetStartingSegment (
            bool isDecreasing,
            Edge & edge )
```

Computes starting segment ID (when not starting inside the edge). Decides whether start from the begin or the end side of the edge.

Parameters

isDecreasing	Flag indicating whether the vehicle is moving in decreasing segment IDs.
edge	The current edge object (to get number of segments).

Returns

Returns starting segment ID.

3.28.3.15 GetStartLineWaitingTime()

```
double Vehicle::GetStartLineWaitingTime ( )
```

Returns

Returns time of the start waiting to go charging.

3.28.3.16 GetStartTime()

```
double Vehicle::GetStartTime ( )
```

Returns

Returns start time of the vehicle (either of the start or when it starts returning).

3.28.3.17 GetStationPosition()

```
const MapPosition & Vehicle::GetStationPosition ( )
```

Returns

Returns charging station position.

3.28.3.18 GoCharging() [1/2]

```
bool Vehicle::GoCharging (
            double batteryTreshold,
            SimulationParameters & simulationParameters )
```

Decides whether go charging or not (based on the battery level, using randomness and estimated battery level in the end (if enough -> don't go charging). Works without need to compute the path.

Parameters

batteryTreshold	Randomly generated additional battery level for decision if go charging or not (if level above current level then go).
simulationParameters	Parameters of the simulation.

Returns

Returns whether go to the charging station `true` or not `false`.

3.28.3.19 GoCharging() [2/2]

```
bool Vehicle::GoCharging (
            vertex_t start,
            vertex_t end,
            Map & map,
            double batteryTreshold,
            SimulationParameters & simulationParameters )
```

Decides whether go charging or not (based on the battery level, using randomness and estimated battery level in the end (if enough -> don't go charging).

Parameters

start	Start vertex of the path.
end	End vertex of the path.
map	Map object to find the path from.
batteryTreshold	Randomly generated additional battery level for decision if go charging or not (if level above current level then go).
simulationParameters	Parameters of the simulation.

Returns

Returns `true` if go to the charging station, else `false`.

3.28.3.20 GoingToStation()

```
bool Vehicle::GoingToStation ( )
```

Returns

Returns `true` if vehicle is heading to the charging station, else `false`.

3.28.3.21 IsFinishing()

```
bool Vehicle::IsFinishing ( )
```

Returns

Returns `true` if vehicle is in the last step of the road or in the penultimate step, which is the further vertex (just get to the correct segment), (finish or charging station).

3.28.3.22 IsIncreasing()

```
bool Vehicle::IsIncreasing (
            vertex_t start,
            vertex_t end )
```

Parameters

start	Start vertex identifier.
end	End vertex identifier.

Returns

Returns `true` if vehicle is moving in increasing order with the segments, else if decreasing `false`

3.28.3.23 IsInsideEdge()

```
bool Vehicle::IsInsideEdge ( )
```

Returns

Returns `true` if vehicle is inside the edge (segment not last or first), else `false`.

3.28.3.24 MoveFirstSegment()

```
bool Vehicle::MoveFirstSegment (
            Map & map,
            bool logs )
```

Moves the vehicle from the starting position to the closest node (either starting or leaving the charging station).

Parameters

map	Map to get information about the route.
logs	Whether to write simulation logs (for debugging).

Returns

Returns `true` if moving was successful, else `false (battery run down).

3.28.3.25 MoveToFinalSegment()

```
bool Vehicle::MoveToFinalSegment (
            Map & map,
            bool logs )
```

Moves the vehicle on the final segment of the path (moving to the middle the edge).

Parameters

map	Map to get information about the route.
logs	Whether to write simulation logs (for debugging).

Returns

Returns `true` if moving was successful, else `false (battery run down).

3.28.3.26 MoveToNextNode()

```
bool Vehicle::MoveToNextNode (
            Map & map,
            bool logs )
```

Moves the vehicle to the next node (without the last and first move).

Parameters

map	Map to get information about the route.
logs	Whether to write simulation logs (for debugging).

Returns

Returns `true` if moving was successful, else `false (battery run down).

3.28.3.27 PopFirstPathVertex()

```
void Vehicle::PopFirstPathVertex ( )
```

Removes first vertex of the path (only use when we want to remove current vertex of the path start).

3.28.3.28 SetChargingTry()

```
void Vehicle::SetChargingTry (
            bool value )
```

Sets whether vehicle tried to go to the charging station.

Parameters

value	Flag if tried `true`, else `false`.

3.28.3.29 SetStartLineWaitingTime()

```
void Vehicle::SetStartLineWaitingTime (
            double startTime )
```

Sets start time of the waiting in the charging station queue.

Parameters

startTime	Start time of the waiting.

3.28.3.30 StartGoingCharging()

```
void Vehicle::StartGoingCharging ( )
```

Sets proper flags to start going to the charging station.

3.28.3.31 TriedCharging()

```
bool Vehicle::TriedCharging ( )
```

Returns

Returns `true` if car already tested if it should go charging, else `false`.

3.28.3.32 UpdateVehiclePath()

```
void Vehicle::UpdateVehiclePath (
            vertex_t start,
            vertex_t end,
            Map & map,
            double batteryTreshold,
            bool removeFirst,
            SimulationParameters & simulationParameters )
```

Updates vehicle path and decides whether it is neccesary to go to the charging station. If so then finds the path to the best estimated charging station.

Parameters

start	Start vertex of the path.
end	Final vertex of the path.
batteryTreshold	Randomly generated additional battery level for decision if go charging or not (if level above then go).
map	Map object to compute path from.
removeFirst	Flag if remove first (current) vertex from the path (useful when not inside edge). If remove then `true` (if path has length 1, then don't remove the vertex (we want to know that it is end of the road), else `false`.

The documentation for this class was generated from the following files:

- Vehicle.h
- Vehicle.cpp

Index