0.1 API Module: Traffic.py

This module instantiates a simulation instance and exposes all necessary/desired functions for interacting with the simulation once it is running:

0.1.1 class TrafficManager:

```
__init__(self, network_config) -> None
Establishes an instance of TrafficManager to run on the given
network structure.
Attributes:
    graph:  Network object that the TrafficManager runs on.
    timestamp:  Simulation timestamp.

add_car(self, car)
API function:  place car (dictionary object) onto the network's
waiting queue.
Please note that the current iteration of add_car only supports
placement directly onto Edges (identified by "start_edge").

get_all_paths_A_to_B(self, start_edge_ID, end_edge_ID)
API function:  Given a start and end Edge id, return a list of
all valid paths that do not repeat Edges.

get_node_edges_in_out(self, node_ID)
API function:  lists the IDs of inbound and outbound edges for
a particular node.
This information can also be found via 'get_snapshot()'.

get_path_distance(self, path)
API function:  Given the ordered list of Edges as "path",
evaluate the total distance it would take to travel.
This function assumes that the entirety of each Edge is
traveled.

get_path_minimum_time(self, path)
API function:  Given the ordered list of Edges as "path",
evaluate the minimum time it would take to travel (in ticks)
given each Edge's speed limit.
Minimum time is calculated assuming a car is able to travel
```

the maximum speed per edge unencumbered.
This function assumes that the entirety of each Edge is traveled and includes any Node-crossing time penalties.
Note: time cost does NOT include Node-crossing time out of the final edge as the Car is expected to exit the Network before the Edge's end.

get_shortest_path_A_to_B(self, all_paths_list)
API function: Given all_paths_list (a list of paths from A to B as calculated using self.get_all_paths_A_to_B()),
returns the path with the shortest total distance in terms of length.

get_snapshot(self)
API function: outputs list of nodes, edge attributes, car attributes.
Output is formatted in such a way that it can be used as input for a new simulation.

get_theoretical_fastest_path_A_to_B(self, all_paths_list)
API function: Given all_paths_list (a list of paths from A to B as calculated using self.get_all_paths_A_to_B()),
returns the path with the minimum total travel time (assuming no congestion).

get_timestamp(self)
API function: returns (sequential) state number.

pause_car(self, car_id)
API function: forcefully halts the Car associated with 'car_id' until a 'resume_car' call is received.
No cars behind it may pass.

remove_car(self, car_id)
API function: removes the Car associated with 'car_id' from the simulation.
This is done by forcing it into Car.status = 'Removed from simulation'.

resume_car(self, car_id)

```
API function:  allows the Car associated with 'car_id' to
resume moving.
```

tick(self)
```
API function:  advance state of network by one unit of time.
```

0.2 Hidden Simulation Module: traffic_network.py

This module creates and runs an instance of a Network object when a simulation is created via starting a **TrafficManager** instance. The main item here is **class Network**, which contains, creates, and collects two dependent components (**class Node** and **class Edge**.

The functions here should be hidden from the user. Instead, the user should call for changes using the**TrafficManager** API. Internal functions belonging to the **traffic_network** module can be seen below:

0.2.1 class Network:

__init__(self, TrafficManagerPointer, config) -> None
```
Contains all functions and attributes pertaining to the (road)
network as a whole.
Attributes:
    TrafficManager_pointer:  Identifies which TrafficManger
    simulation is associated with this network
    node_ID_to_node:  Dictionary mapping Node IDs to Node objects.
    edge_ID_to_edge:  Dictionary mapping Edge IDs to Edge objects.
    car_ID_to_car:  Dictionary mapping Car IDs to Car objects.
    global_tick:  Tick index, aligns with TrafficManager tick
```

add_car(self, car)
```
Places Car (object) on the waiting queue for its specified
start_edge.
Validity checks have been passed up to the TrafficManager level
as a part of "add_car(car)",
ensuring that any Cars received are valid OR can be made valid
using the DEFAULT_car_values_config.json file.
```

add_edge(self, edge)
Imports edge(s) from given edge dictionary.
If both the start and end nodes are already in the network, then the edge will be added.
Any Edge attribute values not given in the edge object (imported) will instead be assigned from the imported defaults file:
edge_default_config.
Note: there are no default values for id, start_node_id, nor end_node_id as these are an Edge's unique identifiers.

add_node(self, node)
Imports node(s) from given node dictionary and adds them to the network.

all_paths_depth_first_search(self, current_edge_ID, end_edge_ID, visited_list=[], valid_paths=[])
Given a start and end Edge id, return a list of all valid paths that do not repeat Edges.

check_valid_car(self, car)
Returns a detailed Exception if the given car does not conform to expected input structure.

choose_path(self, all_paths_list, metric)
Given a list of paths from A to B (ex: as calculated using self.all_paths_depth_first_search()),
returns the "best" path with regards to input metric.
Currently supported input metrics:

- **'Fastest'**: best path = minimum total travel time (assuming no congestion).
- **'Shortest'**: best path = shortest total distance in terms of length.
- **'Random'**: pay no heed to metics, choose an available path at random.

Future versions may include metrics like:

- **'Fastest_now'**: best path = minimum travel time after accounting for current Network congestion.

get_Network_pointer(self)

Returns tick index (which aligns with TrafficManager tick).

get_edge_id(self, edge_id)
Uses Network.edge_ID_to_edge dictionary to map an Edge ID to its corresponding Edge object.

get_global_tick(self)
Returns a pointer to this Network instance.
Useful for directly generating cars OR keeping track managing multiple simulations at once.

get_node_from_id(self, node_id)
Uses Network.node_ID_to_node dictionary to map a Node IDs to its corresponding Node object.

get_snapshot(self)
Outputs dictionary containing snapshot data for all nodes and edges in the network.

path_cost_distance(self, path_list)
Given path_list, evaluate the total distance it would take to travel.
This function assumes that the entirety of each Edge is traveled.

path_cost_minimum_time(self, path_list)
Given path_list, evaluate the the minimum time it would take to travel (in ticks) given each Edge's max_speed.
Minimum time is calculated assuming a car is able to travel the maximum speed per edge unencumbered.
This function assumes that the entirety of each Edge is traveled and includes any Node-crossing time penalties.
Note: time cost does NOT include Node-crossing time out of the final edge as the Car is expected to exit the Network before the Edge's end.

remove_edge(self, edge)
Placeholder for future software version:
Will remove an Edge and all of its associated Cars from the Network.

remove_node(self, node)
Placeholder for future software version:
Will remove a Node and all of its associated inbound/outbound Edges from the Network.

restore_tick_potential(self)
Resets the tick_potential to its maximum value for all Cars on the Network.

tick(self)
Shuffles the order in which Node ticks will be processed with each global tick to ensure no node is favored.
Note: global tick != Node tick. Global tick is the unit of time until the next state of the simulation,
while Node tick the proportion of that time that its components can move uninterrupted.
Node ticks will occur until the sum of their durations reaches that of a global tick/no further movement is possible.

0.2.2 class Node:

__init__(self, Network_reference, id, intersection_cost, stoplight_pattern, stoplight_duration, stoplight_delay) -> None
Contains all functions and attributes pertaining to a network intersection (Node).
Attributes:

- **id**: Unique ID associated with this Node object.
- **inbound_edge_ID_to_edge**: Dictionary mapping inbound Edge IDs to Edge objects.
- **outbound_edge_ID_to_edge**: Dictionary mapping outbound Edge IDs to Edge objects.
- **intersection_time_cost**: Value representing time in ticks required to cross intersection. 0 <= value < 1.
- **stoplight_pattern**: Ordered list of sets of simeltaneous Edges eligible for car exiting. Pattern cycles through sets. (Will be implemented in future versions of the software).
- **stoplight_pattern_current_index**: Index representing which set of stoplight_pattern the Node is currently on. (Will be implemented in future versions of the software).

stoplight_duration: Number of ticks that the stoplight_pattern stays on its current Edge set. (Will be implemented in future versions of the software).
stoplight_delay: Number of ticks between change of stoplight_pattern Edge sets. (Will be implemented in future versions of the software).
node_tick_number: Used in stoplight changes, increments by one with each global TrafficManager tick.

add_to_inbound(self, edge)
Used when adding an Edge to the Network when
Edge.end_node == self.id .
Adds the Edge ID and a mapping to its corresponding Edge object to the inbound_edge_ID_to_edge dictionary.

add_to_outbound(self, edge)
Used when adding an Edge to the Network when
Edge.start_node == self.id .
Adds the Edge ID and a mapping to its corresponding Edge object to the outbound_edge_ID_to_edge dictionary.

get_inbound_exit_candidates(self)
Checks all inbound edges of a Node.
Any edge that has a Car at the end position of its length is considered a candidate to advance on to the next Edge in its path.

get_intersection_time_cost(self)
Returns self.intersection_time_cost, the time penalty it takes to cross a Node.
Note: this value may be 0.

get_node_ID(self)
Returns self.id.
Used when calling value from outside the Node class.

get_node_inbound(self)
Returns the keys to the dictionary self.inbound_edge_ID_to_edge, list of all inbound Edge IDs.
Used when calling from outside the Node class.

get_node_outbound(self)
Returns the keys to the dictionary self.outbound_edge_ID_to_edge, list of all outbound Edge IDs.
Used when calling from outside the Node class.

get_snapshot(self)
Outputs dictionary of Node attributes.

tick(self)
Facilitates Edge ticks and movement of Car objects from one Edge to another.
If a Car that is eligible to cross the Node has type "Dynamic", then its path is recalculated upon crossing.
Each Node tick shuffles the order in which Edges tick to ensure no particular Edge is favored.

update_stoplight_attributes(self)
Toggles which Edges allow cars to exit by cycling through sets in stoplight_pattern.
Returns the set of which inbound Edge set in stoplight_pattern is currently active, or NULL set (denoting red lights for all edges).
Communication to prevent cars from leaving inbound Edges during the node tick will be established in future versions.

0.2.3 class Edge:

__init__(self, id, start_node_id, end_node_id, edge_length, max_speed, max_capacity) -> None
Contains all functions and attributes pertaining to a road segment (Edge).
Attributes:

id: Unique ID associated with this Edge object.
start_node_id: Node from which this Edge originates (this Edge is an outbound_edge for start_node).
end_node_id: Node from which this Edge terminates (this Edge is an inbound_edge for end_node).
start_node: Node object represented by start_node_id.
end_node: Node object represented by end_node_id.

edge_length: Physical length of the Edge (ex: meter length of a road). Default value can be found and adjusted at edge_default_config["edge_length"]

max_speed: (optional) Unit speed limit of the road. Without obstructions, this is the maximum distance a Car can move on this Edge in one tick. Default value can be found and adjusted at edge_default_config["max_speed"]

max_capacity: (optional) Maximum number of Car objects allowed on the Edge (max length of current_cars). Default value can be found and adjusted at edge_default_config["max_capacity"]

edge_car_ID_to_car: Dictionary containing all Car objects associated with the Edge; maps Car IDs to Car objects.

current_cars: List of IDs of all Cars currently on the Edge.

waiting_cars: List of IDs for Cars that are trying to enter the Network at this Edge.

processed_cars: List capturing IDs of Cars that have already been processed on the current tick. Becomes current_cars at the end of the Edge tick.

completed_cars: List of IDs of any Cars that have completed their route on this Edge in the duration of the simulation.

Note: some attributes have been given default values in the case that the user did not provide them.

add_car_to_wait_queue(self, car)
Adds Car object to the waiting queue and links Car to Edge on Car ID.

get_current_cars(self)
Returns self.current_cars, the list of Car IDs for all cars currently on the Edge.
Used when calling value from outside the Edge class.

get_edge_ID(self)
Returns self.id.
Used when calling value from outside the Edge class.

get_end_node(self)
Returns self.end_node Object.
Used when calling value from outside the Edge class.

get_end_node_id(self)
Returns self.end_node_id.
Used when calling value from outside the Edge class.

get_length(self)
Returns self.edge_length (length of road segment, typically in meters).
Used when calling value from outside the Edge class.

get_max_capacity(self)
Returns self.max_capacity.
Used when calling value from outside the Edge class.

get_max_speed(self)
Returns self.max_speed (speed limit of road segment, typically in meters/sec).
Used when calling value from outside the Edge class.

get_snapshot(self)
Outputs dictionary of Edge attributes, including lists of Cars that are: currently on the Edge, waiting to enter the Edge, or completed their trip on this Edge.

get_start_node(self)
Returns self.start_node Object.
Used when calling value from outside the Edge class.

get_start_node_id(self)
Returns self.start_node_id.
Used when calling value from outside the Edge class.

move_existing_car_to_edge(self, car)
Adds Car object to the 'processed-cars' list and links Car to (new) Edge on Car ID.

set_current_cars(self, new_list)
Replaces the list contents of self.current_cars with new_list.
Used when updating value from outside the Edge class.

set_end_node(self, node_ptr)
Associates (end) Node pointer with Edge object.
Used when adding an Edge to the Network.

set_start_node(self, node_ptr)
Associates (start) Node pointer with Edge object.
Used when adding an Edge to the Network.

tick(self)
Facilitates the movement of Car objects traversing this Edge.
There are three types of movement:
car entry: a Car from the waiting_car list will be placed on the Edge if and when space becomes available.
car exiting: a Car will exit the Network if and when it reaches its end_pos_meter in the process of its movement IF self.id = Car.end_edge.
car movement: a Car with status mobile = True will advance as far as possible (maximum potential distance, edge end, or until obstructed by another car).

0.3 Hidden Simulation Module: network_cars.py

This module creates and stores a car object. A car is created when called into existence by an API call via the **Traffic** module. While the **network_cars** module is fully dependent on the **network_traffic** module to move, car objects can exist separately. Thus, **network_cars** is imported into the **network_traffic** module to allow for object-network interaction.

Once again, the functions here should be hidden from the user. Instead, the user should call for changes and additions using the**TrafficManager** API. Internal functions belonging to the **network_traffic** module can be seen below:

0.3.1 class Car:

__init__(self, car_ID, car_length, start_edge, start_pos_meter, end_edge, end_pos_meter, path, car_type, route_preference, max_tick_potential) -> None

```
Contains all functions and attributes pertaining to an object
traversing the Network (Car).
Attributes:
    id:  Unique ID associated with this Car object.
    car_length:  Physical unit length of the Car object
    (ex: meters).  May be 0.
    start_edge:  Edge from which this Car originates
    its journey.
    start_pos_meter:  Unit position along start_edge
    from which the Car begins its journey.
    Edge origin = position 0.
    end_edge:  Edge from which this Car terminates its
    journey.
    end_pos_meter:  Unit position along end_edge at
    which the Car terminates its journey and leaves the Network.
    path:  Ordered list of Edges that the Car will
    traverse to get from start to end.
    route_preference:  Classification determining
    which type of path will be followed:
        if 'Fastest': chooses path with minimum total
        travel time (assuming no congestion).
        if 'Shortest': chooses path with shortest total
        distance in terms of length.
        if 'Random':  pays no heed to metrics and
        instead chooses an available path at random.
    car_type:  Car classification for path-following:
        if 'Static':  Car follows predetermined path
        set on addition.
        if 'Dynamic':  Car will recalculate its route
        every time it reaches a Node.
    mobile:  Car classification for mobility:
        if True:  Car is eligible to move (default).
        if False:  Car has been halted and will not move
        until further instructions given.
    route_status:  string explaining the Car's status
    with regards to path completion:
        'In progress':  The Car is eligible for movement;
        the Car is moving along its path.
        'Route Completed':  The Car has reached its
        destination and has been removed from the Network.
```

'Paused': The Car is ineligible for movement due to mobile=False.

'Removed from simulation at tick #n': The Car was removed from the simulation by external intervention. n denotes timestamp at which it was removed.

current_edge: Edge ID corresponding to the Car's current location.

current_pos_meter_car_front: Unit distance along current_edge corresponding to the Car's current location. If car_length > 0, this refers to the position of the front of the Car.

max_tick_potential: Proportion of global maximum tick time-distance that the Car is eligible to move (default = 1, full potential).

current_tick_potential: Portion of tick time-distance that the car has not (yet) utilized on this tick.

get_car_ID(self)
Returns self.id.
Used when calling value from outside the Car class.

get_car_length(self)
Returns self.car_length.
Used when calling value from outside the Car class.

get_car_type(self)
Returns self.car_type.
Value is **"Static"** (Car remains on its original path) or **"Dynamic"** (Car recalculates path at every Node crossing).
Used when calling value from outside the Car class.

get_current_edge(self)
Returns self.car_length.
Used when calling value from outside the Car class.

get_current_pos_meter_car_front(self)
Returns self.current_pos_meter_car_front.
Used when calling value from outside the Car class.

get_current_tick_potential(self)
Returns self.current_tick_potential.
Used when calling value from outside the Car class.

get_end_edge(self)
Returns self.end_edge, the Edge at which the Car finishes its route and leaves the Network.
Used when calling value from outside the Car class.

get_end_pos_meter(self)
Returns self.end_pos_meter, the position on the Edge at which the Car finishes its route and leaves the Network.
Used when calling value from outside the Car class.

get_max_tick_potential(self)
Returns self.max_tick_potential.
Used when calling value from outside the Car class.

get_mobility(self)
Returns self.mobile.
Value is **True** (Car is eligible to move) or **False**
(Car is halted or its path is complete).
Used when calling value from outside the Car class.

get_path(self)
Returns self.path, the ordered list of upcoming Edges the Car will traverse.
Used when calling value from outside the Car class.

get_route_metric(self)
Returns self.route_preference. Used when a Car's path needs to be (re)calculated.
Value is "Shortest", "Fastest", "Random", with "Random" being the default value if none specified.
Used when calling value from outside the Car class.

get_route_status(self)
Returns self.route_status.
Value is "In progress", "Route Completed", "Paused", or
"Removed from simulation at tick #n".

Used when calling value from outside the Car class.

get_snapshot(self)
Outputs dictionary of Car attributes.

get_start_edge(self)
Returns self.start_edge, the Edge at which the Car entered
the Network.
Used when calling value from outside the Car class.

get_start_pos_meter(self)
Returns self.start_pos_meter, the position on the start Edge
at which the Car entered the Network.
Used when calling value from outside the Car class.

set_current_edge(self, edge_ID)
Replaces self.current_edge with edge_ID.
Used when updating value from outside the Car class.

set_current_pos_meter_car_front(self, new_position_meters)
Replaces self.current_pos_meter_car_front with
new_position_meters.
Used when updating value from outside the Car class.

set_current_tick_potential(self, new_potential)
Replaces self.current_tick_potential with new_potential.
Used when updating value from outside the Car class.

set_mobility(self, Boolean)
Updates the Boolean value of self.mobile to input Boolean.
Value is **True** (Car is eligible to move) or **False**
(Car is halted or its path is complete).
Used when updating value from outside the Car class.

set_path(self, new_path_list)
Replaces self.path with new_path_list,
typically removing the first entry as the car enters a new Edge,
or when calculating a new route.
Used when updating value from outside the Car class.

set_route_status(self, new_string)
Updates self.route_status to new_string.
Value should be "In progress", "Route Completed", "Paused", or
"Removed from simulation at tick #n".
Used when updating value from outside the Car class.

tick(self, old_potential)
Calculates "potential" differential;
This is the portion of a full tick movement completed by the
Car on this tick.

0.4 Optional Module: UnderlyingNetworkGenerator.py

In the absence of a pre-made graph to run the simulation on, the user can elect to use this module to generate a network. This method of Netowrk generation may be useful for some particular use cases where it is beneficial to create a graph with many Nodes and Edges sharing the same properties.

The current version of this module supports the creation of Erdős-Renyi random graphs OR complete, bidirectional graphs. This module will be improved later to support other mathematical network types.

0.4.1 class NetworkGenerator:

__init__(self) -> None
Class containing various functions for generation Network objects
for the simulation to run on.
A Network can also be provided via custom JSON file instead.

create_ER_network_default_values(self, number_nodes,
probability_joining=0.5)
Creates an Erdos Renyi Network based on the given parameters:
A each pair of nodes has a probability_joining (0 < p < 1) of
being connected in an ER Network.
As this is a directional Network, each pair will be considered
separately per direction.
This Network uses the following default values:

```
    probability_joining = 0.5     # can be overwritten by input
    node.intersection_time_cost = 0
```

Please note that this NetworkGenerator function only generates the barebone structures necessary for a Network.
All additional attributes will be loaded via "DEFAULT_edge_values_config.json" during the simulation process.

generate_complete_bidirectional_network_default_values(self, number_nodes)
Generates a complete Network consisting of number_nodes Nodes, each connected to every other Node in both directions.
This Network uses the following default value:

```
    node.intersection_time_cost = 0
```

Please note that this NetworkGenerator function only generates the barebone structures necessary for a Network.
All additional attributes will be loaded via "DEFAULT_edge_values_config.json" during the simulation process.
output_Network_dictionary(self, node_dict, edge_dict)
Returns dictionary containing all Node and Edge information for the newly generated Network.

0.4.2 class GeneratorNode:

__init__(self, id) -> None
Contains all attributes necessary for creating a network intersection (Node).
Attributes:

```
    id:  Unique ID associated with this Node object.
    intersection_time_cost:  Value representing time in ticks
    required to cross intersection.  0 <= value < 1.
```

0.4.3 class GeneratorEdge:

__init__(self, id, start_node_id, end_node_id, edge_length=None, max_speed=None, max_capacity=None) -> None
Contains all attributes necessary for creating a road segment (Edge).
Attributes generated in all NetworkGenerator functions:

```
    id:  Unique ID associated with this Edge object.
```

start_node_id: Node from which this Edge originates.

end_node_id: Node from which this Edge terminates.

Attributes generated only in probabilistic NetworkGenerator functions:

edge_length: Physical length of the Edge (ex: meter length of a road).

max_speed: (optional) Unit speed limit of the road. Without obstructions, this is the maximum distance a Car can move on this Edge in one tick.

max_capacity: (optional) Maximum number of Car objects allowed on the Edge.