Self-Organization of Distributed Autonomous Vehicles Fleets Models Description

23rd January 2020

The system modelling tool and the models of the agents created will be illustrated. First a brief introduction to the modelling tool will be given, followed by the description of the agents created to represent the city of Trento. Finally, the agents that embody the autonomous vehicles and the people searching for lifts will be defined.

1 Simulator used: GAMA Platform

To simulate the agents the open-source GAMA modelling and simulation platform was chosen [2] [9] [4]. This tool was developed since 2007 to respond to the increase in the adoption of agent-based modelling by researchers in different scientific and application domains. These domains often require the exploitation of large datasets. As a consequence, the models have become more complex and detailed. However, the classical approach, KISS [1] [3], is limited under these aspects due to the fact that the model that is considered is the simplest possible one and is changed to a more complex one only when it is strictly needed. Due to this simple approach, it was very useful to create toy models and in helping to establish agent-based modelling as a mainstream approach. However, it cannot cope with different data sources, due to the difference in the representation of the data. In addition to this, the existing platforms generally offer little flexibility in terms of visualisation and parametrization of the model. Some attempts were made to overcome these limitations [7] [6], yet these solutions require expertise in programming languages and external tools for visualisation.

These obstacles have been overcome by the GAMA platform. First of all, it provides an IDE and a specific domain language: the GAma Modelling Language, GAML. It is an intuitive agent-oriented programming language that enables its users to easily define agents, as species, whose behaviour is described by actions and reflexes. Behind the concepts and the operational semantics of GAML, there is a meta-model. The latter provides the mean to represent a model on the base of three main categories of abstract classes representing :

- *Entities*. In the model, the representation of entities is based on:
  - **Agent**: individual entity in the simulation. It is analogous to an object in object-oriented programming and represents the simulated entity.
  - **Population**: group of agents with the same characteristics, in terms of behaviour and structure. It manages the agents that compose it.

- *Space*. To represent the concept of space in the model, the previously mentioned set is needed with the support of the two following main classes:
  - **Geometry**: the geometrical shape representing graphically an *agent* in its *environment*. It is strongly coupled with the single and individual agent it represents.
  - **Topology**: represents the above mentioned *environment*. It provides references to collocate the geometries. It is linked with one population.
- *Time*. Time is represented through the class **Scheduler**, which is linked to the **Agent** class, and the class **Scheduling Information**, linked to the **Population** class.

The last class of the model is the **Species** class, which combines together the three layer of representation expressed by the above mentioned set of classes. The concept represent by a species is similar to a class in object oriented programming. It defines the attribute, characteristic and functions of the entity it represents. Yet, differently from objects, species include also a specification of the topology and scheduling of the population it gives origin to. In addition, it allows to define hierarchies of agents. As it happens, the above mentioned meta-model "explicitly supports the development of lulti-level multi-agent based models" [4] through a *containment* relationship between species. A nested species, a *micro-species*, can indeed be defined inside a species, which will so become a *macro-species*. Hence, it is possible for a species to host a population of micro-species. There may be several reasons behind the necessity of a multi-level representation of agents. For instance, there may be the need to consider entities that are part of organisations and stay at different level. For example, in biology it may be necessary to represent molecules, cells and tissues while also simulating the underlying hierarchy. In this project, this concept was used when representing people that enter into a car by defining a micro-species inside the autonomous vehicles species.

Another important feature that helps in creating complex models using a simple syntax is the possibility to enrich the agents with *skills*. Skills are built-in modules that, when declared in a species, enhance the representation of an agent with a set of skill-related attributes and actions. Since the object of this study was in the mobility domain, the skills which were adopted were:

- *Movement skill*: it implements actions and provides the related variables in order to simulate the movement of agents in an open space or along a graph. It was employed by the people species and the passenger micro-species.
- *Road skill* [8]: was created to define the roads, providing variables and actions to register the agents on each road. The road species employs this skill.
- *Road Node skill* [8]: it helps to define agents that can simulate intersection of roads and traffic signals. This skill was declared for the intersection species.
- *Advanced Driving skill* [8]: it implements actions and provides the related variables in order to simulate agents capable of driving along a driving graph composed of *Roads* and *Road Nodes*, also allowing to specify and tune the drivers behaviour, such as the choice of changing line. The cars and autonomous vehicles species have this skill.

These skills will be further discussed in the sections describing the agents that employ them.

In terms of visualisation, it provides an online visualisation tool, with which the user can interact in order to have a feedback during the simulation. Even so, this graphic aspect and its underlying model are kept separate. In the graphic representation different species are kept on different layers allowing users a high degree of flexibility in the choice of what is to be represent graphically. Moreover, GAMA allows to import digital representations of landscapes, by integrating Geographic Information System (GIS) data, and create geometries from it. On the base of the latter, it is possible to automatically instantiate agents, which will be embodied by this geometry, hence adding the possibility of creating a realistic model of a given geographic area. In this project, this function was used to model realistically the roads, intersections and buildings in a selected area of Trento, in which the system will be simulated.

2 Modelling of the city of Trento: Agents created from GIS data

The simulation was built on top of real data, about roads and buildings, of the city of Trento. These data were extracted from OpenStreetMap. OpenStreetMap is a collaborative project that aims at creating free editable maps of the world with contributions made by users. Inspired by the success of Wikipedia, this initiative was founded in 2014 and its initial intention was mapping the United Kingdom. Its community has now grown to count over 2 million registered users, that can collect data from scratch. There are different ways to collect data, from manual survey to aerial photography, including a variety of other free sources. These data are uploaded and entered into a database and then edited to add attribute information to the map.

These maps are then available for free for various applications and can be downloaded, provided that the copyright is rightfully attributed to the OpenStreetMap contributors. For these project, the area of Trento represented in the Figure 1 below was considered.

The data downloaded [1] was used to create agents to model buildings, roads and intersections, which correspond to real streets and buildings. Moreover, roads and intersections were the base for the generation of a road network which was then used for the movement of the agents.

In the following subsections more details will be given about the above mentioned agents and the road network.

2.1 Buildings

In the OpenStreetMap data, the buildings are spatially represented as points or polygons that may have different characteristics and can be of different types. These information is stored as a map that has as key the tag `building` and as value the assigned type of building. Nevertheless, there are cases in which it may be not be possible to assign a specific type to a given building. For instance, it is not always simple to discern whether a building is an office or a residential building, and therefore the string `yes` is commonly used to at least identify the shape on the map as a

[1] © OpenStreetMap contributors

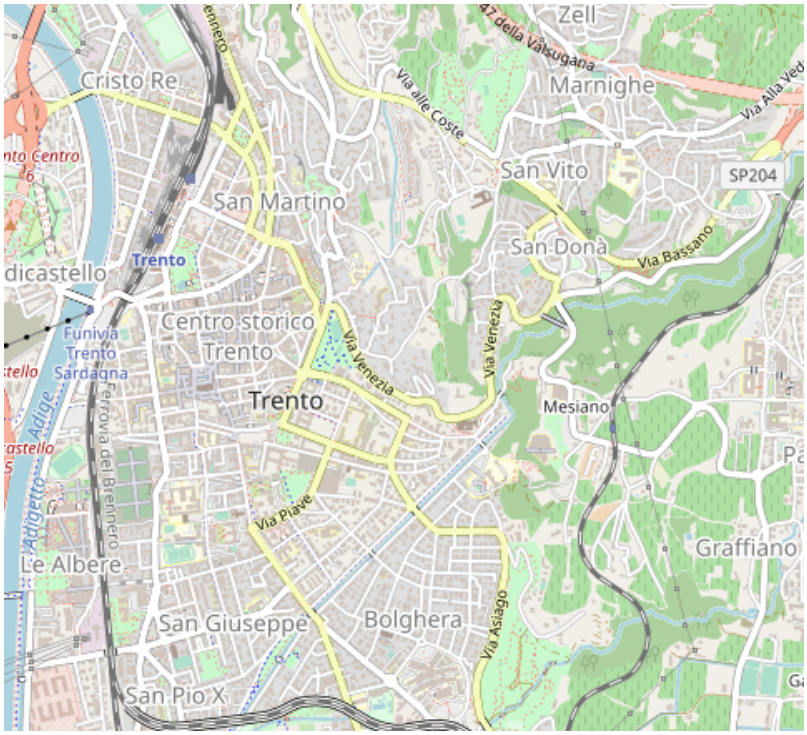


Figure 1: The OpenStreetMap map of the select area of Trento

building. In addition, since the data is crowd-sourced, not all buildings are properly mapped to a string acknowledged and employed by the whole community. An example of this was found when parsing the data about Trento where some values in the map were numbers or Italian words, such as `casa_di_riposo`, and other were too specific, such as `FBK_Fondazione_Bruno_Kessler`.

For these reasons, not all buildings contained in the selected area were considered when creating the agents, but the choice was limited to the values reported in the Table 1 below. Moreover, the buildings labelled with `yes` were excluded from the rendering in order not to overload the simulator, but the types reported in the table under *miscellaneous* were considered in order to have enough buildings. The buildings created were then separated into two categories, *residential* and *industrial*. The residential buildings were straightforward to identify and their types were also reported under the category of residential by the guidelines of OpenStreetMap, with the addition of the above mentioned `casa_di_riposo`. For the industrial buildings, the choice was arduous. Fewer buildings were mapped to types considered as commercial or industrial by the guidelines of OpenStreetMap. Therefore, were also considered as commercial buildings some that belong to other main categories, such as civic buildings or amenities. Finally, the buildings that were not clearly mapped into types, those labelled with the values reported under Miscellaneous in the table, were divided in an equal percentage between residential and commercial buildings in the final map.

Residential	
residential apartments cabin dormitory house hut detached casa_di_riposo	
Commercial - Industrial	
Religious	cathedral chapel church
Commercial or industrial	FBK_Fondazione_Bruno_Kessler commercial industrial kiosk manufacture office retail
Civic / Amenity	civic government gym hospital kindergarten school train_station university
Other buildings	bunker greenhouse hangar
Miscellanous	
unclassifed undefined 0 1 2 3 4 5 6 7 8 9 10 13 14 40	

Table 1: Values of the tag building used

2.2 Roads

As with what was done with the buildings, roads agents were also created starting from the GIS representation of the roads in the area. In OpenStreetMap, roads and their features are identified by the key `highway`. Roads are generally represented as a polygon, a line, composed by different segments. There are different values that are commonly used in OpenStreetMap to identify different type of roads. Among them, in this project the following were taken into consideration:

- `motorway`: representing highways.
- `primary`: representing the second most important roads, after highways, that generally connect larger towns.
- `secondary`: representing roads that connect towns.
- `tertiary`: roads that generally link smaller towns and villages.
- `residential`: roads that serve as access to housings.
- `unclassified and road`: roads, streets or motorways of *unknown* type.

All these roads come with different attributes which describe their characteristics, such as whether is a road designated for bicycles, the driving side or the number of lanes. The properties which were select due to their relevance in the modelling are the following:

- `lanes`: the number of lanes for the direction taken into account.
- `oneway`: whether the road is a one way road or if there are two directions. In the second case a *linked road* is created which will have the same characteristics of the road considered, but opposite direction. More details about one way roads and the generation of linked roads will be given in the subsection 2.4).
- `junction`: used to identify the roundabouts to avoid the erroneously generation of a linked road.

Being Trento an Italian city, the attribute `driving_side` was not considered, since it is assumed that drivers drive on the right side of the road.

In addition to the attributes derived from the information contained in OpenStreetMap, the road has *Road skill* and therefore has the attribute and actions that are reported in the Table 2 below.

Variables	
agents_on	list containing for each lane on the road the list of agents in each segment.
all_agents	the list of agents on the road.
lanes	the number of lanes of the road.
linked_road	when the roads are not one way roads, it corresponds to the opposite direction road.
maxspeed	the maximal speed on the road.
source_node and *target_node*	respectively the source node and the target node of the road.
Actions	
register	is used automatically when an agent enters the road to register its presence on the road. It may register the agent on the linked road in some cases
unregister	is used automatically to remove an agent from the agents on the road when it changes road.

Table 2: Summary of the actions and variables of the built-in *Road* skill

The roads also have a `speed_coefficient` attribute, that helps to identify the amount of traffic on the roads. The traffic was computed on the base of the length of the road and the cars, autonomous vehicles and normal cars, on the street with the following formula

$$speed\_coef = \frac{agents}{capacity} \tag{1}$$

with

$$capacity = 1 + \frac{|road| \times |lanes|}{length_{vehicles}} \tag{2}$$

2.3 Intersections

The intersections in the model are also agents originated from GIS data extracted from OpenStreetMap. In OpenStreetMap, crossings, traffic signals, traffic roundabouts, stops and give way signals, but also bus stops and street lamps, are represented with points and are tagged with the string `highway` as roads. The values considered to identify useful intersections are the following:

- `crossing`: represents a generic area where pedestrians can cross the street. In turn, it can have different attributes in order to distinguish the different crossings. Among them, it was

important for our simulation to track the instances representing traffic signals, namely those in which crossing had as attribute *traffic_signals*, in order to create semaphores.

- `give_way`: represent the give way signal and therefore the drivers or autonomous vehicles should give way to the cars on the other roads.
- `mini_roundabout`: represent a small roundabout
- `stop`: represent the stop signal and therefore the drivers should stop before proceeding to the next road.
- `traffic_signals`: represent an intersection regulated by semaphores.

On the base of the GIS data with the above characteristic, traffic signals and other relevant intersection for our scenario were represented with the species `intersection` following the code created by Patric Taillandier[2].

This species was created with the *RoadNode skill* and has the following built-in characteristics.

`block`	defines the list of agents blocking the node, and for each agent, the list of concerned roads
`priority_roads`	the list of priority roads
`roads_in` and `roads_out`	the lists of input and outputs roads, namely the roads that respectively start and end at the intersection considered
`stop`	define for each type of stop, the list of concerned roads

Table 3: Summary of the variables of the *RoadNode* skill

In addition to these built-in variables, the intersection species has the variable `is_traffic_signal` in order to easily check whether or not the intersection should behave as a traffic signal. In the first case, the agent is initialised via the `initialize` action. By launching the action `compute_crossing` it checks the angles between the roads that end at the intersection in order to separate them into two groups: one group will containing the roads for which, at a given time, the light will be green, whereas the other group will contain the roads that will have a red light. It also has two actions, `to_green` and `to_red` that change the traffic light, respectively, from red to green and the other way around, also updating the `stop` list. Finally, the species has the reflex `dynamic_reflex` to automatically change the colour of the traffic light after a given interval employing the actions mentioned before, by checking on counter value and the current colour of the traffic light.

2.4 From roads and intersections to the graph representing the road network

In our simulation, people and cars are require to move over a graph based on the roads and intersections. GAMA has two built-in functions, `as_edge_graph` and `as_driving_graph`, to create graphs.

[2] `GitHubrepository` for Road Traffic advanced model. author: Patrick Taillandier

The former is used to create a simple undirected graph from the roads, automatically generating nodes when there are overlapping ending points between graphs. This first graph is used for the movements of the people species. The latter is needed to generate a graph that can be used with the *Advanced Driving skill*: it has the roads as edges and the intersections as nodes. For this graph to be correctly generated, it was necessary to create the linked roads for the bidirectional roads, namely, on the base of the roads obtained from the GIS data, for each bidirectional road among them a linked road having the inverse direction was created. An exception was made in the case of the roads identified by the attribute junction. In these cases, it generally was not explicitly specified whether the road considered was a one way road or not, hence it was useful to also consider this attribute as a parameter in deciding whether or not to create its linked road. In addition, for most roads the attribute oneway was not specified, therefore, they were assumed to be bidirectional too and hence the corresponding linked roads were created. Apart from roads, it was also necessary to create some supplementary intersections to signal the end point or start point of a road when such an intersection did not appear in the GIS data. In the Figure 2 below, taken from [8], it can be seen as the graph generated using the intersections as nodes and the roads and linked roads as edges.

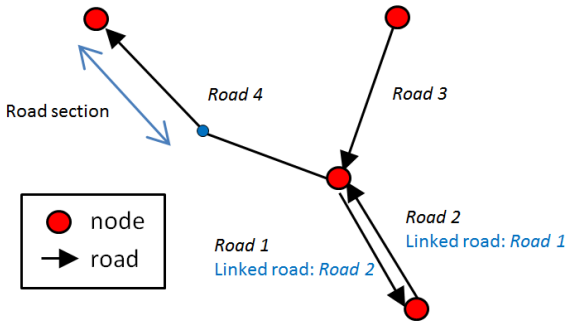


Figure 2: Graphic representation of the graph generated from roads and nodes with the as_driving_graph function

However, after generating the graph, it appeared that some roads were linked among themselves but were separated from the rest of the graph as is shown in the Figure 3. For this reason the roads were eliminated, or killed.

Finally, both the undirected graph and the driving graph allow to specify weights for the edges. On the base of the weights the agents will move more or less slowly and, if they have knowledge of the weights of the edges, this knowledge will affect their choices when choosing a given road over another. The weight of each road was computed as follows:

$$w = \left(\frac{|road|}{speed\_coefficient} \right) \tag{3}$$

Figure 3: Example of groups of roads separated from the rest of the graph

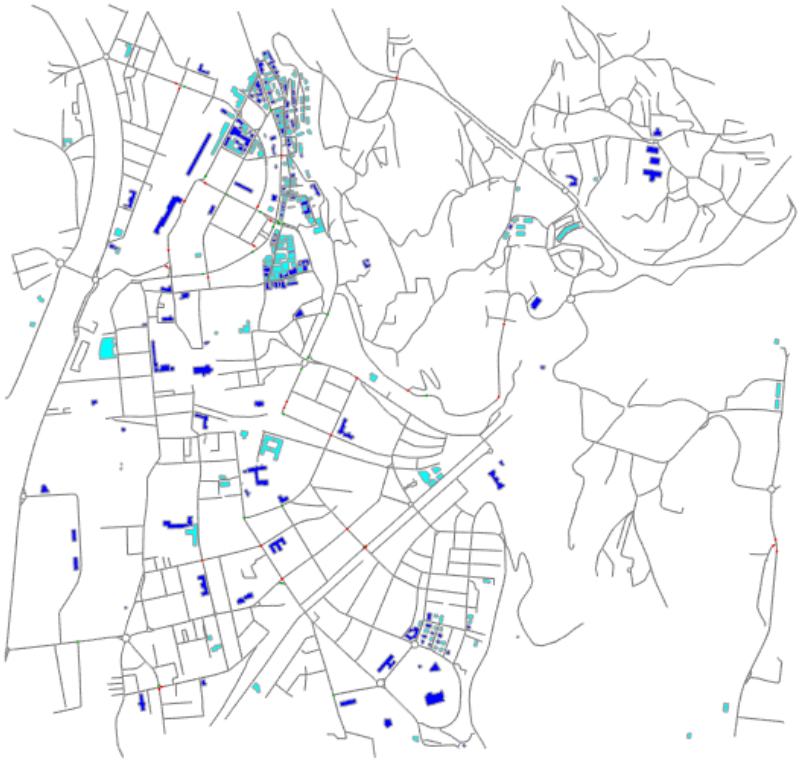


Figure 4: Example of a map generated with roads, intersections and buildings agents

3 Modelling the agents of the system

Along with the above mentioned agents, also three other types of agents were modelled: one that simulates people that are looking for a lift to go to work or to go home, one to simulate autonomous cars that offer lifts and finally another type for normal cars that are generated in order to always have

a certain amount of traffic on the roads. In the following subsections, a more detailed description of these agents will be given.

3.1 People

The people in the system are people that have to reach their homes or working places and who are therefore searching for a lift to reach their working place before a given hour, starting from their home, or a lift at a given hour to return to their home after work. Such entities are modelled in the species *people*. In order to model their behaviour, each agent is initialised with a home, *living place*, and a *working place*, a starting and an ending hour for work and a list of the intersection at a given distance – it was chosen a 50 meter radius – from its location or at least the closest one. This information will be used by the autonomous vehicles in order to recognise people that are looking for a lift and is updated while the agents are searching for a lift.

Moreover, this species has the *moving* skill that defines a minimal set of behaviours that allow agents to move on different topologies. In particular, this skill adds a set of variables to the definition of the agent to specify the starting point (*location*, which is the current geographical position of the agent in the simulation) and destination of the agent (*destination*), the path between these two points (*current_path*) and information about the agent location on the path (*current_edge*, which correspond to the current road occupied by the agent) and predefined and real speed of movement on the path. Moreover, this skill provides four actions:

- **follow**: the agent moves on a given path, according to a given speed and, potentially, movement weights.
- **goto**: with this action a path is computed from the current location of the agent to a given destination. It may be on a topology or a graph that restrict the movements. Once again, it is possible to specify movement weights and the speed to follow.
- **move**: this action moves the agent in an open space and requires to specify the boundaries within which to move.
- **wander**: the agents performing this action wander towards a random location at a given speed and without given boundaries. It may also be constrained by a a graph to follow instead of boundaries. In this last case, it is possible to specify a map containing the probability of choosing an edge in the graph as next edge to follow.

In the modelling of the people, it was chosen to create two behaviours or reflexes, `search_path` and `move`, of which the latter exploit the above mentioned follow action. The first behaviour was created in order to simulate the search of a path in the road graph from the current location of the person to his or hers destination. This is analogous to what happens in real life: nowadays, before commuting to work it is common to search the best option on Google maps, since there may be traffic and deciding to take the usual route may lead to being late. Information about the time required (*time_to_cover*) to reach the destination, the distance the agent will cover (*distance_to_cover*) are stored and, using the latter as a base, the cost for the agent to move alone is computed. The second reflex was created to model the commuting of the person from home to work and the other way around. It keeps track of the time that was actually needed by the agent to arrive to its destination and the distance it covered. It also updates a general statistic of the agent about the distance it has

cover alone (*dist_covered_alone*). In other words, the two behaviours were kept separated in order for the agents to wait a given amount of time before actually moving from their current location towards their target to allow the research of an available autonomous vehicle nearby. In addiction to this behaviours, the agents were modelled as a finite state machine with eight states represented in the Figure 5 and described below.

- **resting**: the agent is at home resting. At a given hour before the starting of its shift, the agent will switch its state to the state *search_lift_to_work*.
- **working**: the agent is working. When its time for the agent to finish working, it will change to the *search_lift_to_home*.
- **search_lift_to_work**: before the shift of the agent starts, it will set its target to the working place location, search for a route to reach it and then wait for an established amount of time for an available autonomous car nearby that could pick it up. After this interval of time has passed, the agent changes its state to *go_work*
- **search_lift_to_home**: after the working hours have ended, the agent will set its target to its house location and search for the best route to return home. Then it will wait a given amount of time for an potential lift from an autonomous car that is passing nearby. After this interval of time has passed, the agent changes its state to *go_home*
- **go_work**: after the given amount of time established to wait for a car, the agent moves towards its working place. It will change its state to *working* when its location correspond to its working place and its the beginning of its shift.
- **go_home**: after the given amount of time established to wait for a car, the agent moves towards its home. It will change its state to *resting*as soon as its position correspond to its house.
- **wait_for_lift**: if a car can give a lift to the passenger, but it is still far away, the agent will wait in the state *wait_for_lift* for the car to arrive.

Finally, two variables, *late* and *actual_time_in* are used to track whether the agent was late at work, in which case the latter will contain the time the agent managed to start working.

3.2 Cars

The driving model used for the vehicles present in the simulation is based on the one proposed by [10] and re-elaborated by [8] on the GAMA platform tool. The agents can compute a new path given an origin and a destination, which may also be chosen randomly, on the road network. This trajectory will be composed by a succession of edges. In the model presented in [10], the movements on an edge are inspired by the Intelligent Driver Model presented in [5], with the addition of the possibility for the drivers to change lane at any time, both when entering a new edge or while they are already on it, and also there are several variables apt to define properties of the car, such as the maximum reachable speed or its length, or the personality of the driver, such as following the general rules for priority at crossroads, for stop signals or speed limits. A summary of those used in this project can be found in the following Table 4.

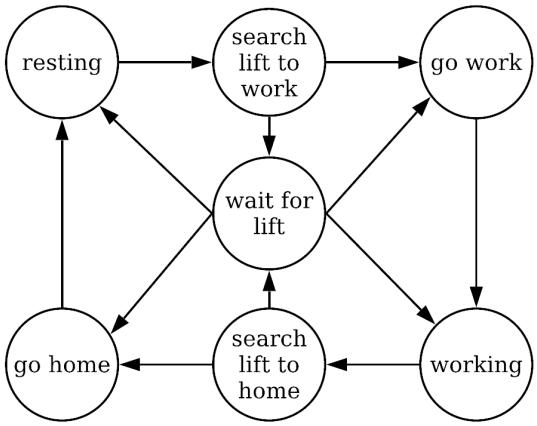


Figure 5: Finite state machine representing the states and related transitions for the species people

In the system, two type of cars were simulated: a common car agent and an autonomous car agent. Various instances of the former are generated in order to simulate traffic on the roads by letting them wander on the roads without a real objective. For this reason, the common cars, specified by the species cars, have two reflexes: `time_to_go` and `move`. The former is used to set a random target for the agent towards which to move, the latter makes the agent move exploiting the action `drive`. In this second reflex, it is also checked whether the agent has slowed down to a speed below 5 kilometres/hour in order to account for cases in which the road is blocked and consequently decide whether or not to turn around and change route. Another reflex, `breakdown`, was implemented in order to simulate the breakdown of a car.

3.3 Autonomous Vehicles

On the other hand, the autonomous cars have the same reflexes as the common car with the addition of more complex ones, since the logic undermining their behaviour is more complex. However, these agents have a different behaviour depending on the employment of a centralised or decentralised solution in the simulation.

In both simulations, all the cars start in the *wander* state, therefore they start by wandering over the road network. In the centralised version, the cars will change their states to the *still_place* state as soon as the manager node assigns them a group of passengers. In this state, they will wait a given amount of time for other passengers, or change directly to the next state, *computing*, to compute the path to follow and the costs for the passengers. It will then loop between the states *moving* and *stop*, in which they respectively move towards the location of passengers to pick them up or drop them off and stop in order to achieve this.

On the other hand, in the decentralised version, the cars will change state from *wander* to *first_stop* when they they find a person or a group of people searching for a lift on the road, trig-

Variables	
current_path	the path that the agent is currently following
current_road	the road on which the agent is
max_acceleration	maximum acceleration for a cycle of simulation
max_speed	maximal speed of the vehicle
min_security_distance	minimal distance to keep from another driver
on_linked_road	whether or not the agent is on a bidirectional road
real_speed	the actual speed of the agent in meter/second)
security_distance_coeff	the coefficient for the computation of the the minimum distance between two drivers. The security distance will depend on the driver speed and on this coefficient.
speed	the speed of the agent in meter/second
speed_coeff	speed coefficient for the speed that the driver want to reach (according to the max speed of the road)
vehicle_length	the length of the vehicle in meters
Probabilities	
proba_block_node	to not let other drivers cross the intersection
proba_lane_change_down and proba_lane_change_up	to change lane to a lower or upper lane if necessary
proba_respect_priorities and proba_respect_stops	to respect priority laws and stop laws (one value for each type of stop)
proba_use_linked_road	to change lane to a linked road lane if necessary
Actions	
compute_path	action to compute a path to a target location according to a given graph
drive	action to drive toward the final target

Table 4: Variables and actions of the *AdvancedDriving* built-in skill

gering a complex decision mechanism. In this case, the general idea is that, a final destination is chosen among the targets of the first person of group of people found in order to set an objective for the car. Once this objective is set, the car can create a first path proposing the related costs to the first set of probable passengers. After the first set of passengers that have accepted the car offer have boarded, the agent changes its state to *moving* and, if there are still open seats, it may propose a lift to passengers found on the road, by changing its state to *stop*, updating the costs for all passengers. In the Table 5 below a summary of the car states is reported and in the figures Figure 6a and Figure 6b a representation of the car states can be found.

In both versions, when they have passengers aboard, the cars keep track of the distance and the expected time needed to reach their next destination, the time that was actually necessary to reach their destinations, the passengers on board and all the stops they have made to either take on board or drop off passengers.

States	Centralised	Decentralised
`wander`	the car wanders on the roads network without a clear objective	
`still_place`	the car stops and waits for a given interval of time to have other passengers assigned to itself	
`computing`	the car computes the path and the related costs	
`first_stop`		the car has found its passenger or group of passengers while wandering as has stopped to pick them up
`moving`	the car is moving on the graph either reaching a passenger location or the destination of a passenger already on board	
`stop`	the car has stopped to either pick up or drop off a passenger assigned to it	the car has stopped to either pick up a passenger on the road when there are other passengers aboard, or to drop off a passenger

Table 5: States of the autonomous vehicles in the centralised and decentralised simulations and their meanings

References

[1] Robert Axelrod. *The complexity of cooperation: Agent-based models of competition and collaboration*, volume 1. Princeton University Press, 1997.

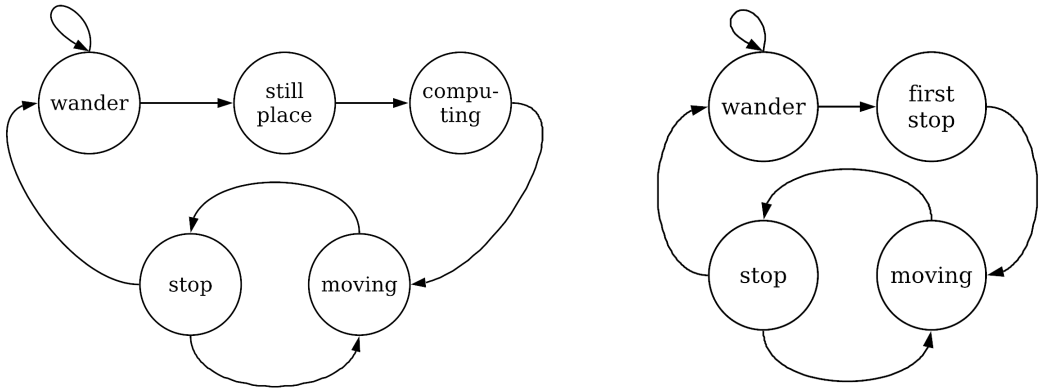
[2] Alexis Drogoul, Edouard Amouroux, Philippe Caillou, Benoit Gaudou, Arnaud Grignard, Nicolas Marilleau, Patrick Taillandier, Maroussia Vavasseur, Duc-An Vo, and Jean-Daniel Zucker. Gama: multi-level and complex environment for agent-based models and simulations. In *Proceedings of the 2013 international conference on Autonomous agents and multi-agent systems*, pages 1361–1362. International Foundation for Autonomous Agents and Multiagent Systems, 2013.

[3] Bruce Edmonds and Scott Moss. From kiss to kids–an 'anti-simplistic'modelling approach. In *International workshop on multi-agent systems and agent-based simulation*, pages 130–144. Springer, 2004.

[4] Arnaud Grignard, Patrick Taillandier, Benoit Gaudou, Duc An Vo, Nghi Quang Huynh, and Alexis Drogoul. Gama 1.6: Advancing the art of complex agent-based modeling and simulation. In Guido Boella, Edith Elkind, Bastin Tony Roy Savarimuthu, Frank Dignum, and Martin K. Purvis, editors, *PRIMA 2013: Principles and Practice of Multi-Agent Systems*, pages 117–131, Berlin, Heidelberg, 2013. Springer Berlin Heidelberg.

[5] Arne Kesting, Martin Treiber, and Dirk Helbing. General lane-changing model mobil for car-following models. *Transportation Research Record*, 1999(1):86–94, 2007.

[6] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. Mason: A multiagent simulation environment. *Simulation*, 81(7):517–527, 2005.



(a) states in the centralised simulation

(b) states in the decentralised simulation

Figure 6: The finite states machine representing the states transition of an autonomous vehicle in the centralised and decentralised simulations, respectively on the left and on the right

[7] Michael J North, Nicholson T Collier, Jonathan Ozik, Eric R Tatara, Charles M Macal, Mark Bragen, and Pam Sydelko. Complex adaptive systems modeling with repast simphony. *Complex adaptive systems modeling*, 1(1):3, 2013.

[8] Patrick Taillandier. Traffic simulation with the GAMA platform. In *International Workshop on Agents in Traffic and Transportation*, page 8 p., France, May 2014.

[9] Patrick Taillandier, Duc-An Vo, Edouard Amouroux, and Alexis Drogoul. Gama: a simulation platform that integrates geographical information data, agent-based modeling and multi-scale control. In *International Conference on Principles and Practice of Multi-Agent Systems*, pages 242–258. Springer, 2010.

[10] Pierrick Tranouez, Eric Daudé, and Patrice Langlois. A multiagent urban traffic simulation. *arXiv preprint arXiv:1201.5472*, 2012.