

Traffic simulation with the GAMA platform

Patrick Taillandier

▶ **To cite this version:**

Patrick Taillandier. Traffic simulation with the GAMA platform. International Workshop on Agents in Traffic and Transportation, May 2014, France. 8 p. hal-01055567

HAL Id: hal-01055567
https://hal.science/hal-01055567

Submitted on 13 Aug 2014

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Traffic simulation with the GAMA platform

Patrick Taillandier
UMR CNRS IDEES, University of Rouen
7 rue Thomas Becket
Mont Saint Aignan, France
patrick.taillandier@univ-rouen.fr

ABSTRACT

These last years have seen the multiplication of traffic agent-based frameworks (MATSim, SUMO...). If these frameworks are well-fitted for the study of normal traffic conditions, it is often complex to adapt them - in particular for non-computer scientists - for more specific application contexts such as the study of impacts of uncommon events (e.g. car accidents, technological hazards). In this paper, we present a new open-source (GPL) tool, integrated into the GAMA modeling and simulation platform, allowing to easily define new microscopic traffic simulations, easily tunable, with a detailed representation of the driver operational behaviors. In particular, it allows to take into account the road infrastructures and traffic signals, the change of lanes of the drivers and their respects of norms. Moreover, the tool allows to run simulations at city level with tens of thousands of driver agents. We illustrate the use of this plug-in through an example for the traffic simulation of the Rouen city (France).

Categories and Subject Descriptors

I.2 [**Artificial Intelligence**]: Distributed Artificial Intelligence

General Terms

Design

Keywords

Agent-based Modeling, Traffic Simulation, GAMA Platform

1. INTRODUCTION

Traffic simulations have proved their interests for urban planners. Many models have been developed these last years. These models are often grouped according to their levels of representation: macroscopic [9], mesoscopic [17], microscopic [12] and nanoscopic [4].

A modeling approach that is particularly well-fitted for micro-simulation is the agent-based modeling. It allows to consider the heterogeneity of the driver behaviors and to take into account the global impact of local processes. This modeling approach is in particular adapted to the study, at fine scale (spatial and temporal), of the impacts of uncommon events such as car accidents or technological hazards (see for example [16]). In this context, being able to simulate the traffic in a realistic way while taking into account the road infrastructure (crossing, traffic signals...), the properties of the cars (length, max speed...) and the personality of the drivers (tendency to respect the norms) is mandatory.

Even if there are nowadays many frameworks dedicated to the development of agent-based traffic models, many models are still developed from scratch or with a generic platform (e.g. [6, 3, 16]). Indeed, if the existing frameworks are most of the time well-fitted to the simulation of normal traffic conditions, they cannot be easily tuned by domain experts that are often not computer scientists.

In this paper, we present a new plug-in integrated in the open-source GAMA modeling and simulation platform [5] dedicated to the development of fine scale traffic simulations. GAMA provides modelers - which are not, most of the time, computer scientists - with tools to develop highly complex models. In particular, it offers a complete modeling language (GAML: GAma Modeling Language) and an integrated development environment that allows modelers to quickly and easily build models. Indeed, the GAML language is as simple to use and to understand as the Netlogo modeling language [15] and do not requires high level programming skills. The plug-in developed allows GAMA user to easily define traffic simulation at fine scale, with a detailed representation of the driver operational behaviors.

The paper is organized as follows: Section 2 presents the related works, in particular the existing agent-based traffic simulators and frameworks. Section 3 is dedicated to the presentation of the driving GAMA plug-in. Section 4 gives an example of a model developed with the plug-in. At last, Section 5 concludes and presents some perspectives.

2. RELATED WORKS

Many open source traffic simulation frameworks have been developed these last years.

One of the most famous is MATSim [2] (Multi Agent Transport Simulation Toolkit). MATSim is an open-source (GPL) Java application that consists of several modules which can be combined. MATSim proposes many advance features dedicated to traffic simulations that can be enrich by users through the definition of new modules in JAVA.

Another famous open-source framework is SUMO [10]. SUMO is a suite of applications which help modelers to prepare and to perform traffic simulations. Like MATSim, it

proposes many advance features dedicated to traffic simulations that can be enrich using C++.

AgentPolis [7] is another open source framework dedicated to traffic simulations. In comparison to the two previously cited frameworks, AgentPolis adopts a fully agent-based modeling approach. Drivers are represented as autonomous agents with asynchronous control modules and the ability to interact freely with the environment and other agents. AgentPolis is implemented in JAVA and can be enrich using the JAVA programming language.

These three frameworks are powerful and propose many advance features. However, for modelers without high level programming skills, adapting these platforms to specific application contexts is out of reach as they require to write code in JAVA or C++.

Concerning the generic modeling and simulation platforms, only few propose tools that can be used to develop traffic simulations. One of them is Repast Symphony [13] that proposes interesting features concerning GIS loading and graphs. However, using this platform to develop complex models require to write code in JAVA. Note that for simple models, modelers can use the Relogo modeling language.

The GAMA platform [5] provides as well different features that can be used by modelers to develop traffic models. In particular, GAMA allows to simply load GIS data (shape files, OSM data...), to define graphs from polyline geometries, to compute shortest paths and to move agents on a polyline networks. If these features are well-suited for the development of traffic simulations at large time scale (see for example the MIRO project [3] - time scale: 10 minutes per step), they do not allow to simply account the driver behaviors at fine scale: his/her change of lanes, the effect of traffic signals...

In this context, we have developed a new driving plug-in for the GAMA platform. The goal is to offer to modelers a tool that is at the same time easy to use for all application contexts and that allows to build realist traffic simulations.

3. DRIVING GAMA PLUG-IN

3.1 Presentation of the plug-in

The developed tool is integrated in the GAMA platform as a plug-in. It provides modelers with new GAML instructions allowing to support the definition of traffic simulation. GAML is an agent-oriented language, in which modelers define species of agents, i.e. archetype of agents, their characteristics (variables), behaviors and aspects. The behaviors of agents are defined through actions and reflexes. An action is a block of instructions executed when called. A reflex is a block of instructions executed at each simulation step when its optional attached condition is true. An aspect represents how an agent can be displayed. The richness of GAML comes from the numerous optimized operators it integrates. In particular, GAMA provides modelers with a native integration of GIS data and allows to easily load shapefiles, OSM data and to use databases. It integrates as well many graph operators.

Concerning our tool, we chose to represent all the road infrastructures (road, traffic signals) as agents. The main interest of this is to give the modelers the possibility to simply add dynamics to these infrastructures: e.g. to add a deterioration dynamic to roads.

Our tool takes the form of three GAMA skills. A skill is a built-in module that provides a set of related built-in variables and built-in actions (programmed in JAVA) to the species of agents that declare them. In particular, we define 3 new skills:

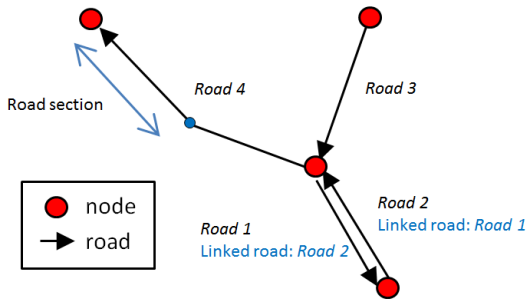


Figure 1: Roads and nodes

- *Advanced driving skill*: dedicated to the definition of the driver species. It provides the driver agents with variables and actions allowing to move an agent on a graph network and to tune its behavior.
- *Road skill*: dedicated to the definition of roads. It provides the road agents with variables and actions allowing to registers agents on the road.
- *RoadNode Skill*: dedicated to the definition of node. It provides the node agents with variables allowing to take into account the intersection of roads and the traffic signals.

3.2 Structure of the network: road and roadNode skills

A key issue for our tool is to be versatile enough to be usable with most of classic road GIS data, in particular OSM data. We choose then to use a classic format for the roads and nodes (See Figure 1). Each road is a polyline composed of road sections (segments). Each road has a target node and a source node. Each node knows all its input and output roads. A road is considered as directed. For bidirectional roads, 2 roads have to be defined corresponding to both directions. Each road will be the *linked_road* of the other. Note that for some GIS data, only one road is defined for bidirectional roads, and the nodes are not explicitly defined. In this case, it is very easy, using the GAML language, to create the reverse roads and the corresponding nodes (it only requires few lines of GAML).

A lane can be composed of several lanes (Figure 2) and the vehicles will be able to change at any time its lane. Another property of the road that will be taken into account is the maximal authorized speed on it. Note that even if the user of the plug-in has no information about these values for some of the roads (the OSM data are often incomplete), it is very easy using the GAML language to fill the missing value by a default value. It is also possible to change these values dynamically during the simulation (for example, to take into account that after an accident, a lane of a road is closed or that the speed of a road is decreased by the authorities).

The *road skill* provides the road agents with several variables that will define the road properties:

- *lanes*: integer, number of lanes.

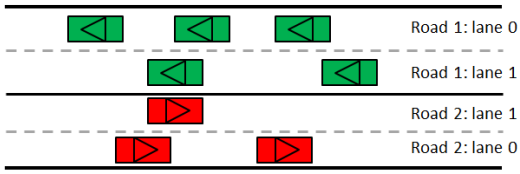


Figure 2: Roads and lanes

- *maxspeed*: float point value; maximal authorized speed on the road.
- *linked_road*: road agent; reverse road (if there is one).
- *source_node*: node agent; source node of the road.
- *target_node*: node agent; target node of the road.

It provides as well the road agents with one read only variable:

- *agents_on*: list of list (of driver agents); for each lane, the list of driver agents on the road.

The *roadNode skill* provides the road agents with several variables that will define the road properties:

- *roads_in*: list of road agents; the list of road agents that have this node for target node.
- *roads_out*: list of road agents; the list of road agents that have this node for source node.
- *stop*: list of list of road agents; list of stop signals, and for each stop signal, the list of concerned roads.

It provides as well the road agents with one read only variable:

- *block*: dictionary (map): key: driver agent, value: list of road agents; the list of driver agents blocking the node, and for each agent, the list of concerned roads.

3.3 Advanced driving skill

Concerning the driver agents, we propose a driving model based on the one proposed by [16]. In the model proposed by [16], each driver agent has a planned trajectory that consists in a succession of edges. When the driver agent enters a new edge, it first chooses its lane according to the traffic density, with a bias for the rightmost lane. The movement on an edge is inspired by the Intelligent Driver Model [8]. A difference with our driving model is that in our model the drivers have the possibility to change their lane at any time (and not only when entering a new edge). In addition, we have defined more variables for the driver agents in order to give more possibilities for the modelers to tune the driver behavior.

The *advanced driving skill* provides the driver agents with several variables that will define the car properties and the personality of the driver:

- *final_target*: point; final location that the agent wants to reach (its goal).
- *vehicle_length*: float point value; length of the vehicle.
- *max_acceleration*: float point value; maximal acceleration of the vehicle.
- *max_speed*: float point value; maximal speed of the vehicle.
- *right_side_driving*: boolean; do drivers drive on the right side of the road?
- *speed_coef*: float point value; coefficient that defines if the driver will try to drive above or below the speed limits.
- *security_distance_coeff*: float point value; coefficient for the security distance. The security distance will depend on the driver speed and on this coefficient.
- *proba_lane_change_up*: float point value; probability to change lane to a upper lane if necessary (and if possible).
- *proba_lane_change_down*: float point value; probability to change lane to a lower lane if necessary (and if possible).
- *proba_use_linked_road*: float point value; probability to take the reverse road if necessary (if there is a reverse road).
- *proba_respect_priorities*: float point value; probability to respect left/right (according to the driving side) priority at intersections.
- *proba_respect_stops*: list of float point values; probabilities to respect each type of stop signals (traffic light, stop sign...).
- *proba_block_node*: float point value; probability to accept to block the intersecting roads to enter a new road.

It provides as well the driver agents with several read only variables:

- *speed*: float point value; speed expected according to the road *max_value*, the car properties, the personality of the driver and its *real_speed* (see Equation 1 for more details).
- *real_speed*: float point value; real speed of the car (that takes into account the other drivers and the traffic signals).
- *current_path*: path (list of roads to follow); the path that the agent is currently following.
- *current_target*: point; the next target to reach (sub-goal). It corresponds to a node.
- *targets*: list of points; list of locations (sub-goals) to reach the final target.
- *current_index*: integer; the index of the current goal the agent has to reach.
- *on_linked_road*: boolean; is the agent on the linked road?

Of course, the values of these variables can be modified at any time during the simulation. For example, the probability to take a reverse road (*proba_use_linked_road*) can be increased if the driver is stucked for several minutes behind a slow vehicle.

In addition, the *advanced driving skill* provides the driver agents with several actions:

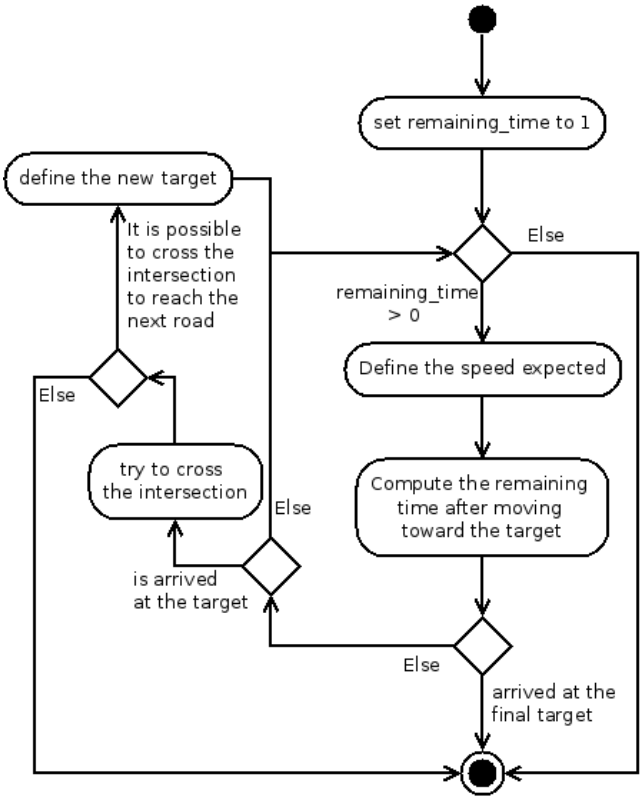


Figure 3: Drive action

- *compute_path*: arguments: a graph and a target node. This action computes from a graph the shortest path to reach a given node.
- *drive*: no argument. This action moves the driver on its current path according to the traffic condition and the driver properties (vehicle properties and driver personality).

the *drive* action works as follow (Figure 3): while the agent has the time to move ($remaining_time > 0$), it first defines the speed expected. This speed is computed from the *max_speed* of the road, the current *real_speed*, the *max_speed*, the *max_acceleration* and the *speed_coef* of the driver (see Equation 1). Then, the agent moves toward the current target and compute the remaining time. During the movement, the agents can change lanes (see below). If the agent reaches its final target, it stops; if it reaches its current target (that is not the final target), it tests if it can cross the intersection to reach the next road of the current path. If it is possible, it defines its new target (target node of the next road) and continues to move.

$$speed_{driver} = Min(max_speed_{driver}, Min(real_speed_{driver} + max_acceleration_{driver}, max_speed_{road} * speed_coef_{driver})) \quad (1)$$

The function that defines if the agent crosses or not the intersection to continue to move works as follow (Figure 4): first, it tests if the road is blocked by a driver at the intersection (if the road is blocked, the agent does not cross the

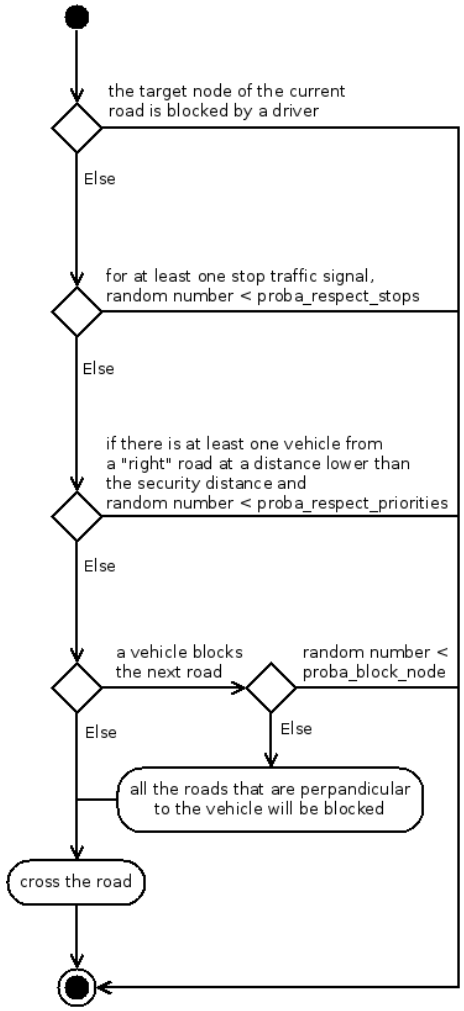


Figure 4: Crossing of an intersection (case where *right_side_driving* is true)

intersection). Then, if there is at least one stop signal at the intersection (traffic signal, stop sign...), for each of these signals, the agent tests its probability to respect or not the signal (note that the agent has a specific probability to respect each type of signals). If there is no stopping signal or if the agent does not respect it, the agent checks if there is at least one vehicle coming from a right (or left if the agent drives on the left side) road at a distance lower than its security distance. If there is one, it tests its probability to respect this priority. If there is no vehicle from the right roads or if it chooses to do not respect the right priority, it tests if it is possible to cross the intersection to its target road without blocking the intersection (i.e. if there is enough space in the target road). If it can cross the intersection, it crosses it; otherwise, it tests its probability to block the node: if the agent decides nevertheless to cross the intersection, then the perpendicular roads will be blocked at the intersection level (these roads will be unblocked when the agent is going to move).

Concerning the movement of the driver agents on the current road (Figure 5), the agent moves from a section of the road (i.e. segment composing the polyline) to another sec-

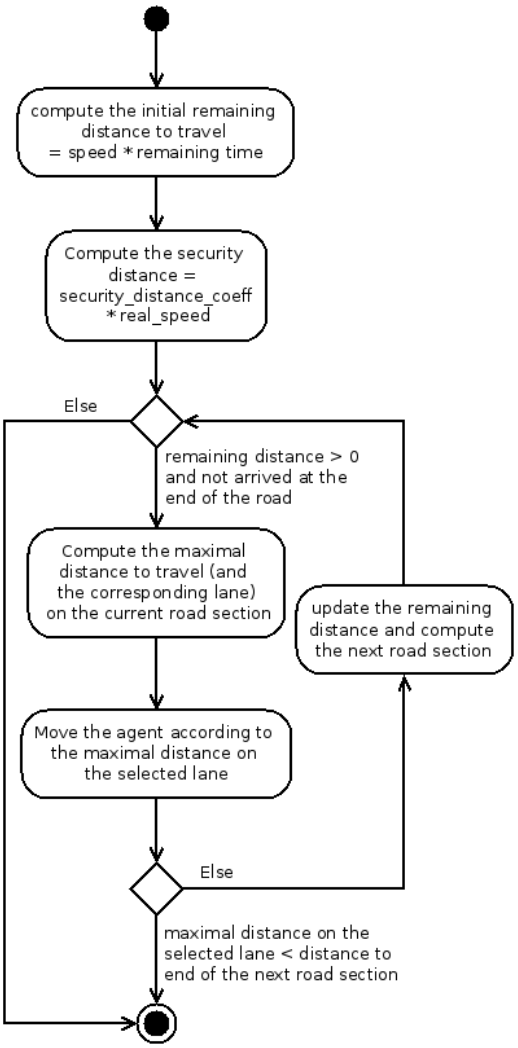


Figure 5: Move on the current road

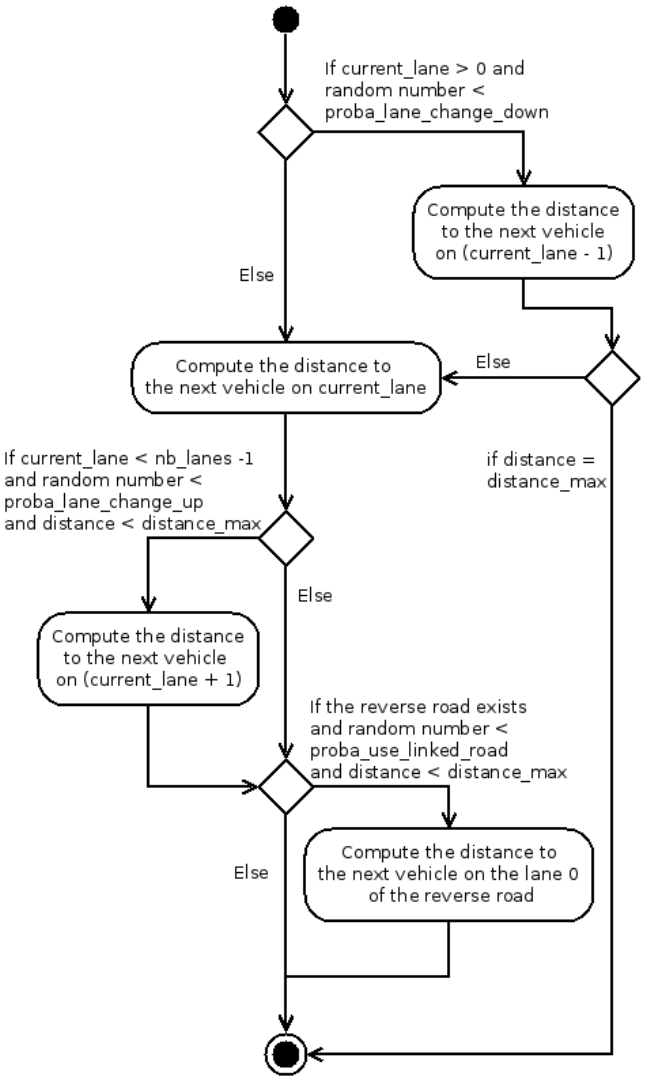


Figure 6: Define the maximal distance possible to travel and the corresponding lane (case where *right_side_driving* is true)

tion according to the maximal distance that the agent can moves (that will depend on the remaining time). For each road section, the agent first computes the maximal distance it can travel according the remaining time and its speed. Then, the agent computes its security distance according to its speed and its *security_distance_coeff*. While its remaining distance is not null, the agent computes the maximal distance it can travel (and the corresponding lane), then it moves according to this distance (and update its current lane if necessary). If the agent is not blocked by another vehicle and can reach the end of the road section, it updates its current road section and continues to move.

The computation of the maximal distance an agent can move on a road section consists in computing for each possible lane the maximal distance the agent can move. First, if there is a lower lane, the agent tests the probability to change its lane to a lower one. If it decides to test the lower lane, the agent computes the distance to the next vehicle on this lane and memorizes it. If this distance corresponds to the maximal distance it can travel, it chooses this lane; otherwise it computes the distance to the next vehicle on its current lane and memorizes it if it is higher than the current memorized maximal distance. Then if the memorized distance is lower than the maximal distance the agent can travel and if there is an upper lane, the agents tests the probability to change its lane to a upper one. If it decides to test the upper lane, the agent computes the distance to the next vehicle on this lane and memorizes it if it is higher than the current memorized maximal distance. At last, if the memorized distance is still lower than the maximal distance it can travel, if the agent is on the highest lane and if there is a reverse road, the agent tests the probability to use the reverse road (linked road). If it decides to use the reverse road, the agent computes the distance to the next vehicle on the lane 0 of this road and memorizes the distance if it is higher than the current memorized maximal distance.

3.4 Discussion

As presented above, the plug-in allows to simplify the work of modelers for the definition of traffic simulations with the GAMA platform. Of course, the plug-in does not

make GAMA as rich as the existing frameworks for the development of such simulations. In particular, it proposes no tools for the pre-processing of data and do not propose any features concerning the definition of the construction of the driver daily activities. However, our tool is perfectly adapted to modelers that are not computer scientists and that want to quickly create a specific traffic model (or at least a prototype) that is not possible to create using directly the existing framework. The success of generic and simple platforms such as Netlogo [15] or GAMA have proved the interest of researchers from many research fields (geographers, sociologists...) for this kind of tools.

4. APPLICATION EXAMPLE

We illustrate the use of our plug-in for a simple model concerning the simulation of the traffic of the city of Rouen (France, Normandie). This city of 111553 inhabitants is built on the two sides of the Seine River. Five bridges allow to cross the river. These bridges are particularly critical for the traffic in Rouen. For instance, a truck (transporting fuel) accident has caused the closing of the Mathilde bridge since the 29th October of 2012. This bridge, which was the most used to cross the Seine (80 000 vehicles per day), should remain closed until summer 2014. This accident had (and still have) an important impact on the traffic as it has led to the multiplication of traffic jams.

As the goal of this model is just to illustrate the use of the new driving plug-in, we did not use real data to define the driver origin and destination: we affected to each driver agent a random initial location (one of the node) and a random final target (one of the node). When a driver agent reaches its destination, it just chooses a new random final target. In the same way, we did not define any specific behavior to avoid traffic jam for the driver agents: once they compute their path (all the driver agents use for that the same road graph with the same weights), they never re-compute it even if they are stucked in a traffic jam. Concerning the traffic signals, we just consider the traffic lights (without any pre-processing: we consider the raw OSM data). One step of the simulation represents 1 second. At last, in order to clarify the explanation of the model, we chose to do not present the parts of the GAML code that concern the simulation visualization. The complete model is available on the GAMA SVN (is downladable from the GAMA website [1]).

Figure 7 shows the total area (road and node shapefiles) that we choose to take into account in the simulation. This area is composed of 8000 roads and 6000 nodes. We used the OSM data (converted as shapefiles) of Rouen. A preprocess has been applied on the data in order to create a node shapefile from the road shapefile: a node is created at the extremity of each road (when several roads intersect each other, only one node is created at the intersection). Note that a GAMA model, available on the GAMA SVN, allows to directly pre-process the OSM data and to create the node and road shapefiles from them.

The following code shows the definition of species to represent the road infrastructure:

```
species road skills: [skill_road] {
  string oneway;
}
```



Figure 7: Total area simulated: in black the roads and in yellow the nodes

```
species node skills: [skill_road_node] {
  bool is_traffic_signal;
  int time_to_change <- 100;
  int counter <- rnd (time_to_change) ;

  reflex dynamic when: is_traffic_signal {
    counter <- counter + 1;
    if (counter >= time_to_change) {
      counter <- 0;
      stop[0] <-empty(stop[0])? roads_in : [];
    }
  }
}
```

In order to use our driving plug-in, we just have to add the *skill_road_node* to the *node* species and the *skill_road* to the *road* species. In addition, we added to the road species a variable called *oneway* that will be initialized from the OSM data and that represents the traffic direction (see the OSM map features for more details). Concerning the node, we defined 3 new attributes:

- *is_traffic_signal*: boolean; is the node a traffic light?
- *time_to_change*: integer; represents for the traffic lights the time to pass from the red light to the green light (and vice versa).
- *counter*: integer; number of simulation steps since the last change of light color (used by the traffic light nodes).

In addition, we defined for the *node* species a reflex (behavior) called *dynamic* that will be activated only for traffic light nodes and that will increment the *counter* value. If this counter is higher than *time_to_change*, this variable is set to 0, and the node change the value of the *stop* variable: if the traffic light was green (i.e. there is no road concerns by this stop sign), the list of block roads is set by all the roads that

enter the node; if the traffic light was red (i.e. there is at least one road concerns by this stop sign), the list of block roads is set to an empty list.

The following code shows the definition of driver species:

```
species driver skills: [advanced_driving] {
  reflex time_to_go when: final_target = nil {
    current_path <- compute_path(
        graph: road_network, target: one_of(node));
  }
  reflex move when: final_target != nil {
    do drive;
  }
}
```

In order to use our driving plug-in, we just have to add the *advanced_driving* to the *driver* species. For this species, we defined two reflexes:

- *time_to_go*: activated when the agent has no final target. In this reflex, the agent will randomly choose one of the nodes as its final target, and computed the path to reach this target using the *road_network* graph. Note that it will have been possible to take into account the knowledge that each agent has concerning the road network by defining a new variable of type map (dictionary) containing for each road a given weight that will reflect the driver knowledge concerning the network (for example, the known traffic jams, its favorite roads....) and to use this map for the path computation.

- *move*: activated when the agent has a final target. In this reflex, the agent will drive in direction of its final target.

We describe in the following code how we initialize the simulation:

```
init {
  create node from: file("nodes.shp") with:[
    is_traffic_signal::read("type")="traffic_signals"];

  create road from: file("roads.shp")
    with:[lanes::int(read("lanes")),
    maxspeed::float(read("maxspeed")),
    oneway::string(read("oneway"))]
    {
      switch oneway {
        match "no" {
          create road {
            lanes <- myself.lanes;
            shape <- polyline(reverse
              (myself.shape.points));
            maxspeed <- myself.maxspeed;
            linked_road <- myself;
            myself.linked_road <- self;
          }
        }
        match "-1" {
          shape <- polyline(reverse(shape.points));
        }
      }
    }
  }

  map general_speed_map <-  road as_map
    (each::(each.shape.perimeter/(each.maxspeed)));

  road_network <-  (as_driving_graph(road, node))
     with_weights general_speed_map;

  create driver number: 100000 {
    location <- one_of(node).location;
    vehicle_length <- 3.0;
    max_acceleration <- 0.5 + rnd(500) / 1000;
    speed_coeff <- 1.2 - (rnd(400) / 1000);
    right_side_driving <- true;
    proba_lane_change_up <- rnd(500) / 500;
    proba_lane_change_down <- 0.5+ (rnd(250) / 500);
    security_distance_coeff <- 3 - rnd(2000) / 1000);
    proba_respect_priorities <- 1.0 - rnd(200/1000);
    proba_respect_stops <- [1.0 - rnd(2) / 1000];
    proba_block_node <- rnd(3) / 1000;
    proba_use_linked_road <- rnd(10) / 1000;
  }
}
```

In this code, we create the node agents from the node shapefile (while reading the attributes contained in the shapefile), then we create in the same way the road agents. However, for the road agents, we use the *oneway* variable to define if we should or not reverse their geometry (*oneway* = "-1") or create a reverse road (*oneway* = "no"). Then, from the road and node agents, we create a graph (while taking into account the *maxspeed* of the road for the weights of the edges). This graph is the one that will be used by all agents to compute their path to their final target. Finally, we create 10000 driver agents. At initialization, they are randomly placed on the nodes; their vehicle has a length of 3m; the maximal acceleration of their vehicle is randomly drawn between 0.5 and 1; the speed coefficient of the driver is randomly drawn between 0.8 and 1.2; they are driving on the right side of the road; their probability of changing lane for a upper lane is randomly drawn between 0 and 1.0; their probability of changing lane for a lower lane is randomly drawn between 0.5 and 1.0; the security distance coefficient is randomly drawn between 1 and 3; their probability to respect priorities is randomly drawn between 0.8 and 1; their probability to respect light signal is randomly drawn between 0.998 and 1; their probability to block a node is randomly drawn between 0 and 0.003; their probability to use the reverse road is randomly drawn between 0 and 0.01;

We carried out a simulation of 1000 simulation steps (1000 seconds) on a i7 computer using only one of the computer cores. The duration of the simulation (without taking into account the time taken by the displaying of the simulation) was 1 second per step if we take into account the time spent by the shortest path computation by the Dijkstra algorithm or 0.3 second per step if we do not. Figure 8 shows a snapshot of the simulation. We can observe the emergence of traffic jams, driver agents stopping at a red traffic light and using the different lanes of the roads.

5. CONCLUSION

In this paper, we presented a new plug-in for the GAMA platform dedicated to the development of traffic simulations. This plug-in allows to define new traffic simulations with a detailed representation of the driver operational behaviors.

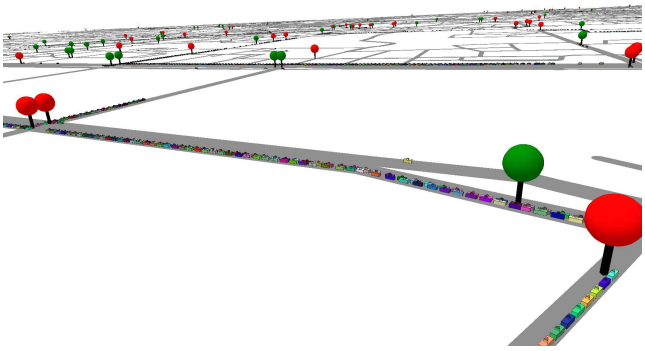


Figure 8: Snapshot of the simulation (simulation step: 1000): The driver agents are represented by the rectangles with a triangle on top; the traffic lights are represented by the sticks with a red/green sphere on top

In particular, it allows to take into account the road infrastructures and traffic signals, the change of lanes of the drivers and their respect of norms. We illustrated the use of our plug-in by a simple model concerning the simulation of the traffic of the city of Rouen.

In comparison to existing traffic simulation frameworks, the advantage of our tool is to enable modelers to easily define models adapted to their application context. Indeed, the use of the GAML language enables modelers without high-level programming skills to develop their own models or at least prototypes.

If the plug-in allows yet to simulate tens of thousands of driver agents, we plan to improve its efficiency by using High Performance Computing and in particular distribution on GPU to enable to carry out large scale simulation with millions of driver agents.

In addition, we plan to enrich the driving skill in order to make the driver agents more cognitive, in particular concerning their choice of path and their adaptation to the their current context. For this, we plan to give the driver agents a BDI architecture that can be based on [14, 11].

As last, we plan as well to develop new tools to help people to prepare their data. The goal will be to offer the possibility from incomplete OSM data (OSM are often incomplete) to automatically fill the missing attributes, and to create a consistent network (with its infrastructure and traffic signals). A particular attention will be brought on traffic signals and traffic lights.

6. REFERENCES

[1] Gama website: https://code.google.com/p/gama-platform, January 2014.

[2] M. Balmer, M. Rieser, K. Meister, D. Charypar, N. Lefebvre, K. Nagel, and K. Axhausen. Matsim-t: Architecture and simulation times. *Multi-Agent Systems for Traffic and Transportation Engineering*, pages 57–78, 2009.

[3] A. Banos, N. Marilleau, and M. Team. Improving individual accessibility to the city: an agent-based modelling approach. In *ECSS*, 2012.

[4] N. Daiheng. 2dsim: A prototype of nanoscopic traffic simulation. In *Intelligent Vehicles Symposium*, pages 47–52, 2003.

[5] A. Grignard, P. Taillandier, B. Gaudou, D. Vo, N. Huynh, and A. Drogoul. Gama 1.6: Advancing the art of complex agent-based modeling and simulation. In *PRIMA 2013: Principles and Practice of Multi-Agent Systems*, volume 8291 of *Lecture Notes in Computer Science*, pages 117–131, 2013.

[6] M. Horn. Multi-modal and demand-responsive passenger transport systems: a modelling framework with embedded control systems. *Transportation Research Part A: Policy and Practice*, 36(2):167–188, 2002.

[7] M. Jakob and Z. Moler. Modular framework for simulation modelling of interaction-rich transport systems. In *Proceedings of the 16th IEEE Intelligent Transportation Systems Conference (ITSC 2013)*, 2013.

[8] A. Kesting, M. Treiber, and D. Helbing. General lane-changing model mobil for car-following models. *Journal of the Transportation Research Board*, 1999:86–94, 2007.

[9] A. Kotsialos, M. Papageorgioy, C. Diakaki, Y. Pavlis, and F. Middleham. Traffic flow modeling of large-scale motor- way networks using the macroscopic modeling tool metanet. *IEEE Transactions on Intelligent Transportation Systems*, 3(4):282–292, 2002.

[10] D. Krajzewicz, J. Erdmann, M. Behrisch, and L. Bieker. Recent development and applications of SUMO - Simulation of Urban MObility. *International Journal On Advances in Systems and Measurements*, 5(3&4):128–138, 2012.

[11] V. M. Le, B. Gaudou, P. Taillandier, and D. A. Vo. A new bdi architecture to formalize cognitive agent behaviors into simulations. In *Advanced Methods and Technologies for Agent and Multi-Agent Systems (KES-AMSTA)*, volume 252 of *Frontiers in Artificial Intelligence and Applications*, pages 395–403. IOS Press, 2013.

[12] J. E. Miller, D. J. Hunt, J. E. Abraham, and S. P. A. Microsimulating urban systems. *Computers, Environment and Urban Systems*, 28:9–44, 2004.

[13] M. North, N. Collier, J. Ozik, E. Tatara, C. Macal, M. Bragen, and P. Sydelko. Complex adaptive systems modeling with repast simphony. *Complex Adaptive Systems Modeling*, 1(1):3, 2013.

[14] P. Taillandier, O. Therond, and B. Gaudou. A new bdi agent architecture based on the belief theory. application to the modelling of cropping plan decision-making. In *International Environmental Modelling and Software Society (iEMSs)*, 2012.

[15] S. Tisue and U. Wilensky. Netlogo: A simple environment for modeling complexity. In *International Conference on Complex Systems*, pages 16–21, 2004.

[16] P. Tranouez, E. Daudé, and P. Langlois. A multiagent urban traffic simulation. *Journal of Nonlinear Systems and Applications*, 3(2):98–106, 2012.

[17] K. Waldeer. Numerical investigation of a mesoscopic vehicular traffic flow model based on a stochastic acceler- ation process. *Transport Theory and Statistical Physics*, 33(1):31–46, 2004.