

# Final report :

## Drone Simulator



### Project membres :

BLLADI Ismail  
DRISSI SLIMANI Youness  
KENAAN Assaad  
LOUKILI Younes

## Table of contents

1	Introduction.....	4
1.1	Purpose.....	4
1.2	Definitions – Abbreviations .....	4
2	General description .....	5
2.1	Context diagram .....	5
2.2	User requirement .....	5
2.3	Development constraints .....	5
2.4	Work hypotheses.....	6
3	Functional needs .....	7
3.1	Use case.....	7
3.2	Sequence diagram .....	8
3.2.1	Communication between nearest drones.....	8
3.2.2	Charging sequence .....	9
3.2.3	Healing Sequence .....	10
3.2.4	Delivery sequence .....	11
3.3	State diagram .....	11
3.3.1	Ambulance drone .....	11
3.3.2	Delivery drone .....	12
4	Data structure specifications.....	13
4.1	Class diagram.....	13
5	External interfaces specifications.....	16
6	Performance needs .....	21
7	Development constraint.....	22
8	References.....	23

## Table of figures

Figure 1 : Context diagram .....	5
Figure 2 : User use case .....	7
Figure 3 : Communication between nearest drones .....	9
Figure 4 : Charging sequence .....	9
Figure 5 : Healing Sequence .....	10
Figure 6 : Delivery sequence .....	11
Figure 7 : Ambulance state diagram .....	11
Figure 8 : Delivery state diagram .....	12
Figure 9: Class diagram.....	13
Figure 10 : Gama HMI.....	16
Figure 11 : Model file .....	17
Figure 12 : Dashboard project.....	18

# 1 Introduction

## 1.1 Purpose

The main objective of this document is to develop a three-dimensional drone simulator. This document gives the detailed description of the functional requirement proposed by the client. The purpose of this project is to simulate two different scenarios: First scenario is about delivering packages and the other is about healing people who got into an accident. Both scenario shows how drones can make decisions by themselves, interact with the environment and between each other to apply it in real life, that is the autonomous “Agent” concept. This project will describe the software interface using UML diagrams and we will show how this simulation works.

## 1.2 Definitions – Abbreviations

**Drone:** is an unmanned aerial vehicle (UAV) that is lifted and propelled by rotors

**Autonomous Agent:** is an intelligent agent operating on an owner’s behalf but without any interference of that ownership entity

**SMA:** System Multi-Agent

**UML:** Unified Modeling Language Java: Platform independence

**GAMA:** is a modeling and simulation development environment for building spatially explicit agent-based simulations

**GAML (GAma Modeling Language):** An agent-oriented language based on Java.

## 2 General description

In this part, we will give a general description of the system with its mains functionalities.

### 2.1 Context diagram

The diagram bellow is a context diagram which describe the interaction between the user and the system.

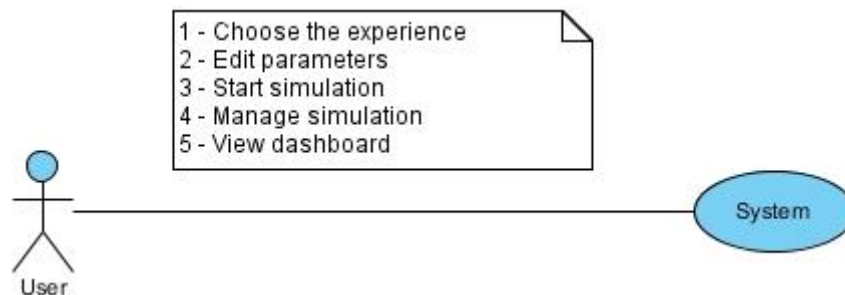


Figure 1 : Context diagram

Actions	Descriptions
<b>Choose the experience</b>	The user can choose the scenario that he wants to simulate. (Delivery scenario or ambulance scenario or both)
<b>Edit parameters</b>	The user can choose the parameters of this simulation (number of drones, speed etc.)
<b>Start simulation</b>	The user starts the simulation
<b>Manage simulation</b>	The user can define the speed of simulation and choose the angle of view etc.
<b>View the dashboard</b>	The user can watch the dashboard which contains the statistics in real time of simulation (level of battery, number of delivery etc.)

### 2.2 User requirement

The user does not need any IT skills. He just needs GAMA and **Java 1.8** be installed on his machine, approximately 200MB of disk space and a minimum of 4GB of RAM on his computer and launch the project for enjoying it.

### 2.3 Development constraints

The company needs a drone simulation software. We have chosen to work with GAMA platform because is the most fitted for this kind of project. The only requirement for us was to learn the concept of system agent-oriented to better understand how GAMA works.

## 2.4 Work hypotheses

**SMA concept:** It's the best concept for autonomous simulation, an alternative is to work with object-oriented paradigm combined with threads.

### 3 Functional needs

#### 3.1 Use case

In this part we will describe the functional needs of our system by using the UML use case diagram.

This diagram represents the chronological steps of interaction between the user and the drone simulator's system.

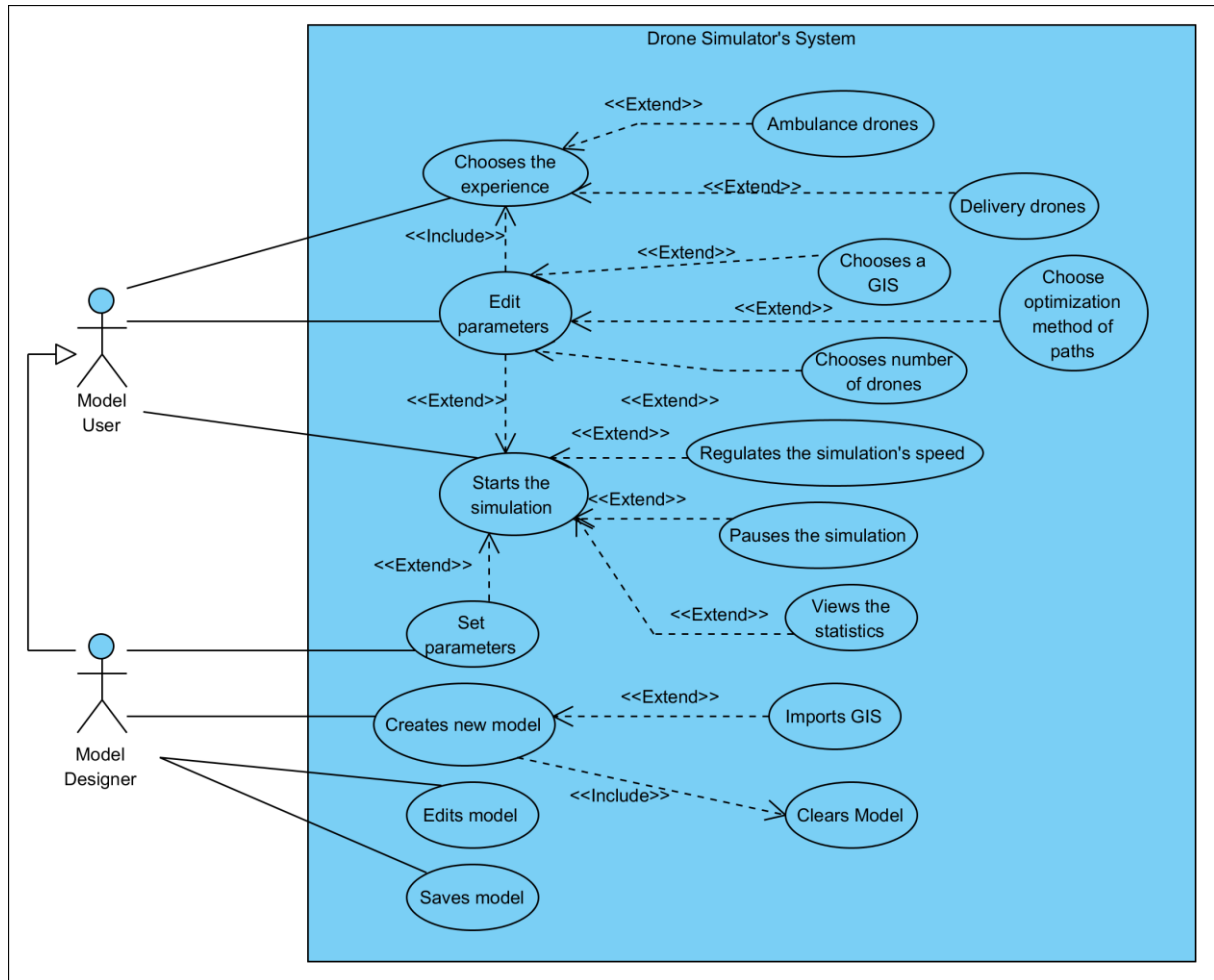


Figure 2 : User use case

Choose the experience	
<b>Description</b>	The user can choose the scenario that he wants to simulate. (Delivery scenario or ambulance scenario or both)
<b>Actors</b>	User
<b>Inputs</b>	Scenarios
<b>Process</b>	Load scenario with initial parameters
<b>Outputs</b>	Interface of simulation

Edit parameters	
<b>Description</b>	The user can choose the parameters of this simulation
<b>Actors</b>	User
<b>Inputs</b>	GIS, numbers of drones, speed of drones, speed of simulation, algorithm of shortest path
<b>Process</b>	Replace default parameters
<b>Outputs</b>	Interface of simulation

Start simulation	
<b>Description</b>	The user starts the simulation and can manage it, view dashboard which contain statistics
<b>Actors</b>	User
<b>Inputs</b>	Start, Pause, Stop, Restart, Regulate simulation's speed
<b>Process</b>	Simulation is working in real time
<b>Outputs</b>	Interface of simulation

## 3.2 Sequence diagram

In this part we will explain the different operations performed by the drone. We used the sequence diagram because it seems to be the best way to explain agent behaviors.

### 3.2.1 Communication between nearest drones



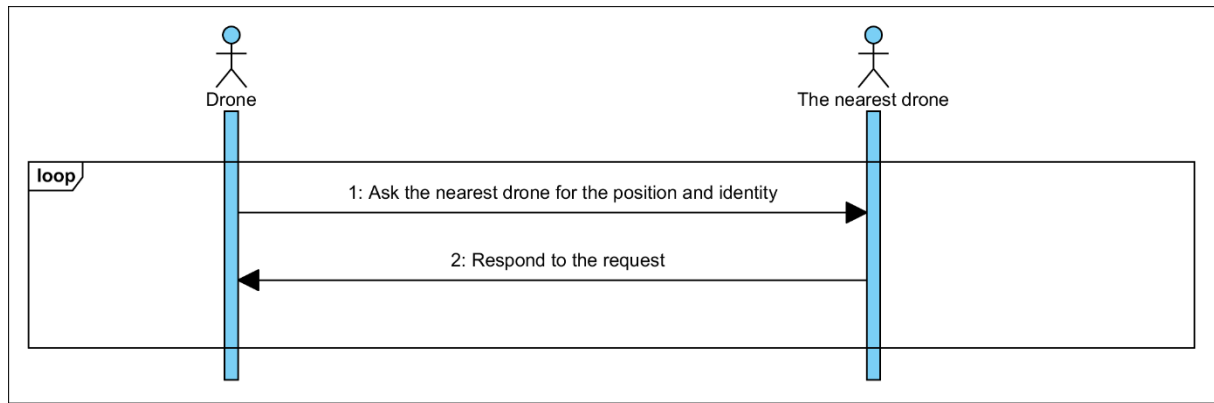


Figure 3 : Communication between nearest drones

Each drone can communicate with the nearest drone, the process of communication follows the steps below:

- The drone asks the closest drone about its identity and position
- The asked drone responds to the request

### 3.2.2 Charging sequence

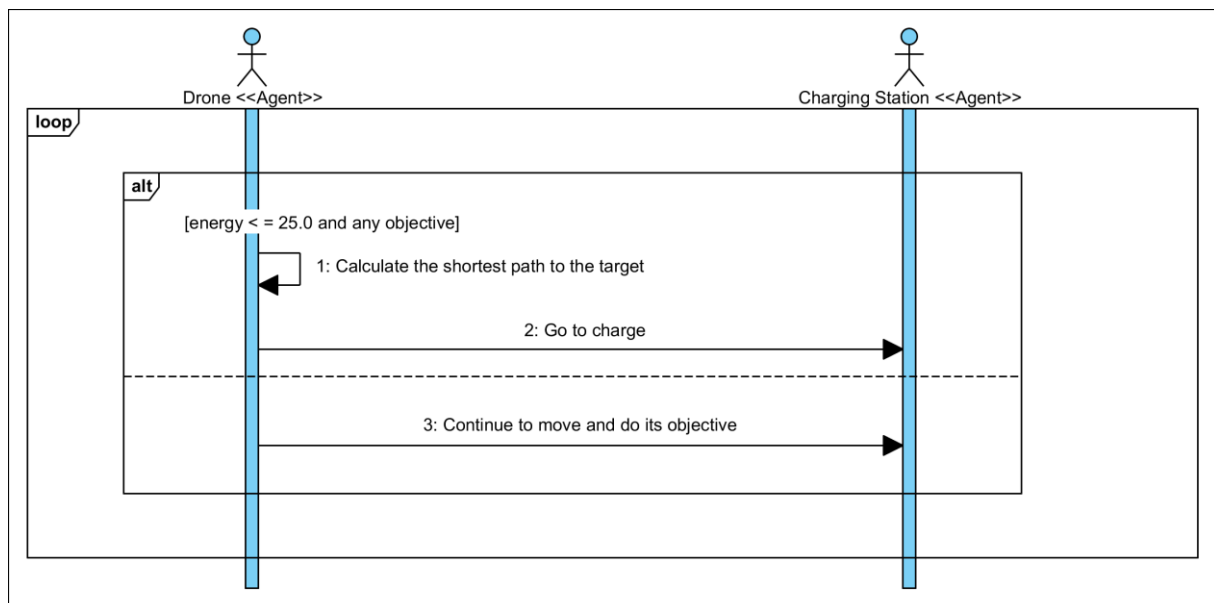


Figure 4 : Charging sequence

In our simulation, the process of charging is between the drone agent and the charging station agent follows the steps below:

- The drone loose battery when it is moving
- Once the level of battery is under 25%, the drone determines the shortest path to the nearest charging station
- The drone goes to the charging station
- Once the drone arrived, the charging station recharge the drone's battery
- The drone is going back to its objective (healing for ambulance drones and delivering for delivery drones)

### 3.2.3 Healing Sequence

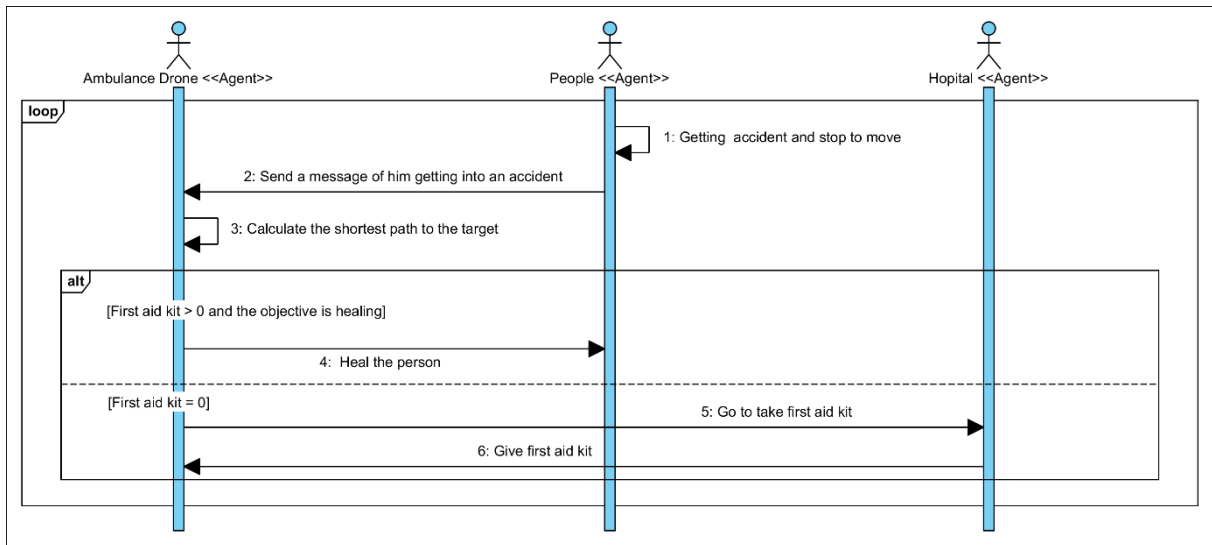


Figure 5 : Healing Sequence

The healing operation involves three agents (Ambulance drone, people, hospital) and follows the steps below:

- The people agent getting to an accident according to a probability defined by the model designer
- After that the people agent reports to ambulance drones that he has had an accident and needs healing
- Once the drone receives the report, the ambulance check if he has enough first aid kit, if it's true, the ambulance determines the shortest path to the victim of accident
- The ambulance drones go to heal accidented people
- Once the drone arrived it gives the first medication to the accidented people, then people is going back to move
- Once the drone receives the report, the ambulance check if he has enough first aid kit, if it's false, the ambulance determines the shortest path to the hospital
- The ambulance drones go to the hospital
- Once the drone arrived, the hospital gives first aid kit to the ambulance

### 3.2.4 Delivery sequence

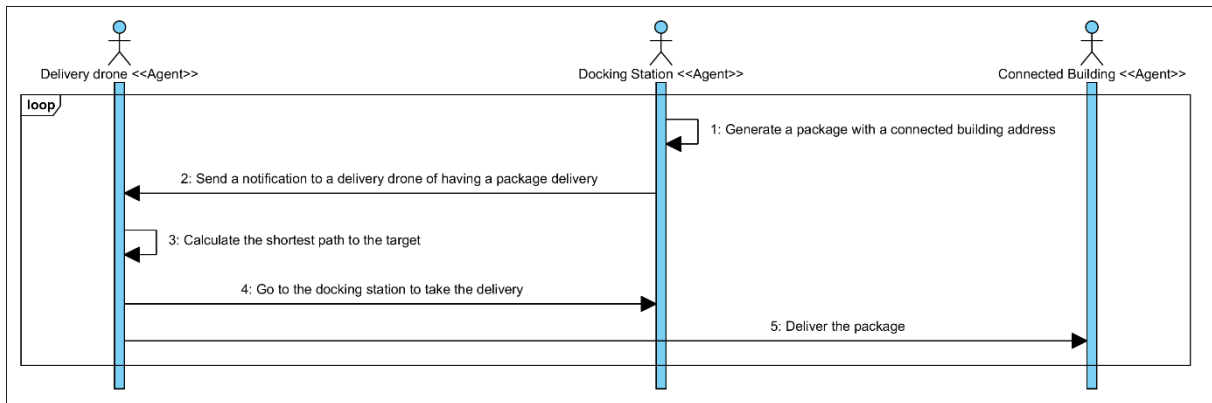


Figure 6 : Delivery sequence

The delivery operation involves three agents (Delivery drone, docking station, connected building) and follows the steps below:

- The docking station generate a package with delivery address
- The docking station send a notification to one of delivery drone that is has a package to deliver
- Once the delivery drone receives the message, he determines the shortest path to the docking station
- When the drone arrived at the docking station he takes the package
- After that the drone deliver the package at the delivery address

## 3.3 State diagram

### 3.3.1 Ambulance drone

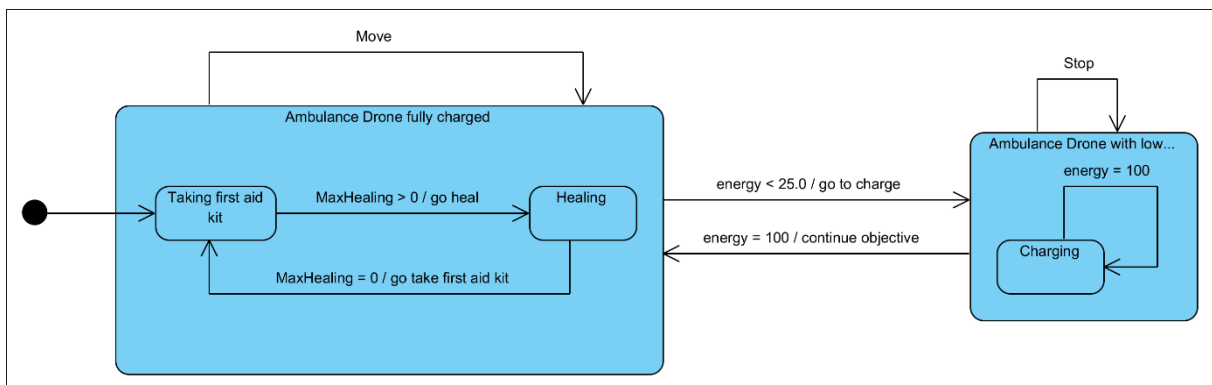


Figure 7 : Ambulance state diagram

### 3.3.2 Delivery drone

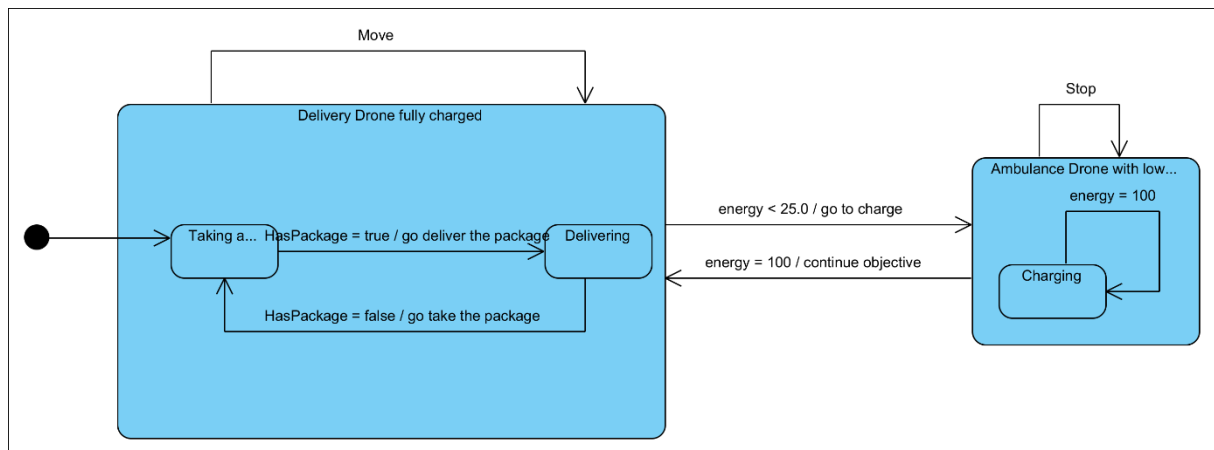


Figure 8 : Delivery state diagram

## 4 Data structure specifications

### 4.1 Class diagram

A class diagram is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among objects.

In the diagram, classes are represented with boxes that contain three compartments:

- The top compartment contains the name of the class. It is printed in bold and centered, and the first letter is capitalized.
- The middle compartment contains the attributes of the class. They are left-aligned, and the first letter is lowercase.
- The bottom compartment contains the operations the class can execute. They are also left-aligned, and the first letter is lowercase.

It was led to think about how the simulation works to define a functional model, which will lead to the implementation of the code. This diagram also allows having a view of the program before writing and visualizing more complex methods to think about it directly. Here is the diagram we designed:

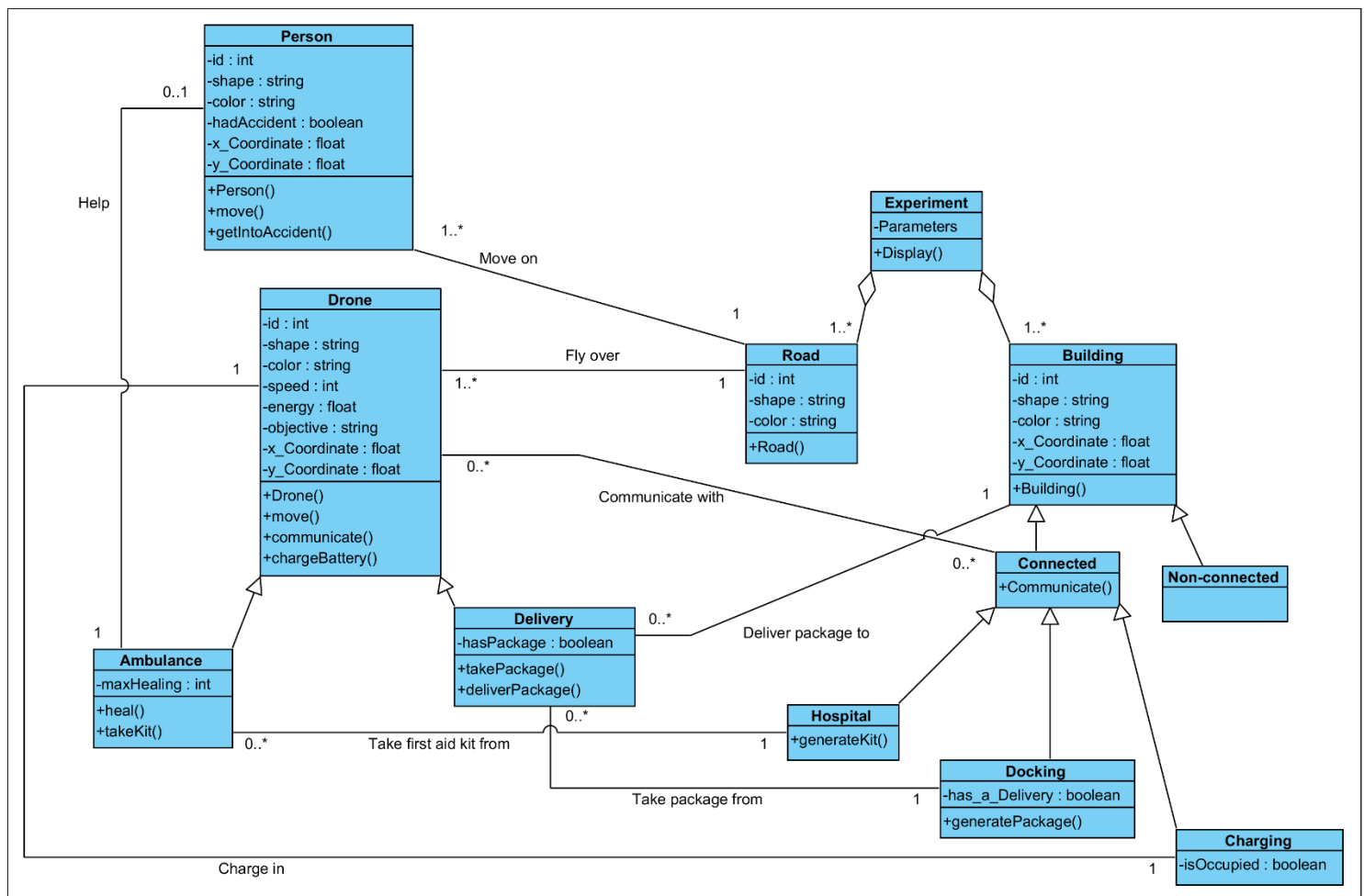


Figure 9: Class diagram

We have determined a set of 12 classes, which can be divided into several groups.

- Drone

The drone super class has an *id* (int), which will serve as communication between the drones themselves or with the connected building. Its *shape* (string) and *color* (string) that distinguishes it from other types of drones. The drone has a *speed* (int) that will be assigned to it during the simulation. Its *energy* (float) will decrease with each step of the simulation. It also has an *objective* (string) which depends about drone itself (has enough of energy or not) and the situation of the other agent (infected person, package to delivery). Finally, the *x\_Coordinate* and *y\_Coordinate* (float) of the drone that will serve to define its new objectives and target.

The drone also has a constructor ***Drone()*** and ***communicate()***, ***move()***, ***chargeBattery()*** as public methods.

One drone can charge from one of any charging station.

One or many drones can fly (move) over one road and communicate or not with non or more connected building.

In our diagram, we have two types (child class) of drone named **Ambulance** and **Delivery** which inherit from drone class.

- Ambulance

In addition to the properties of the drone class, the **Ambulance** class has a *maxHealing* (int) which means the number of kits that it can support. It also has a ***heal()*** and ***takeKit()*** as public methods. The ambulance drone can take the first aid kit from one hospital station and help no one or more person (It depends on the number of kits, the battery and if there are a person infected or not).

- Delivery

The **Delivery** class has a Boolean attribute called *hasPackage* (Boolean), in order to know if it should take a package or not, using ***takePackage()*** and ***deliverPackage()*** methods.

This drone can take the package from any Docking station and deliver it to any connected building.

- Person

Attributes: *id*, *shape*, *color*, *hadAccident* (Boolean) to know him status, *the x\_Coordinate and y\_Coordinate* that will help the ambulance to detect his position and go to therapy him.

This class also has a constructor called ***Person()***, and two methods called ***move()*** and ***getIntoAccident()***.

***move()*** : to move on road randomly.

***getIntoAccident()*** : to get infected randomly at any time.

One or many persons can move on one road and get into accident randomly at any time.

- Environment

When the user starts the simulation, the environment will display all the agent declared. It's composed of one or many roads and of one or many buildings.

- Road

The child class Road has as attributes: *id* (int), *shape* (string), *color* (string), and the constructor ***Road()***.

- Building

The **Building** is a super class which has an *id*, *shape*, *color*, and the *x\_Coordinate* and *y\_Coordinate* as attributes. And **Building()** as constructor.

- Connected

This child class, inherit from the Building class and has a method: ***Communicate()*** that allow the communication between its self and the other agent.

- Non-Connected

The non-connected building is a type of Building that it cannot communicate with any other agent.

- Docking

The Docking class is a connected building that has: *has\_a\_Delivery* (Boolean) as attribute to know if the delivery drone can take a package or not. The ***generatePackage()*** method is called when the station doesn't have a package in it.

- Charging

The Charging class is a connected building that means the charging station and it has as attribute: *isOccupied* (Boolean) which doesn't allow the charging of more than one drone at the same time.

- Hospital

The Hospital class is a connected building that contains kits. It can generate the Kit if there's not enough using the ***generateKit()*** method.

## 5 External interfaces specifications

For this project, we have used '**GAMA**', a modeling and simulation development environment for building spatially explicit agent-based simulations. GAMA is used for multiple application domains. It is using **GAML (GAMA MODELING LANGUAGE)**, a high-level and intuitive agent-based language, based on Java.



GAMA 1.7<sup>1</sup> comes in 5 different versions (32 & 64 bits for Windows & Linux, and 64 bits for MacOS X).

GAMA 1.7 requires that **Java 1.8** be installed on your machine, approximately 200MB of disk space and a minimum of 4GB of RAM (to increase the portion of memory usable by GAMA). **Please note that GAMA is not considered as compatible with Java 1.9 and Java 1.10** as it has not been tested under these environments.

After launching GAMA 1.7, the user will see this HMI:



Figure 10 : Gama HMI

As we see, all projects and models made or imported, in the workspace, by the user/developer himself are grouped in the folder "User models".

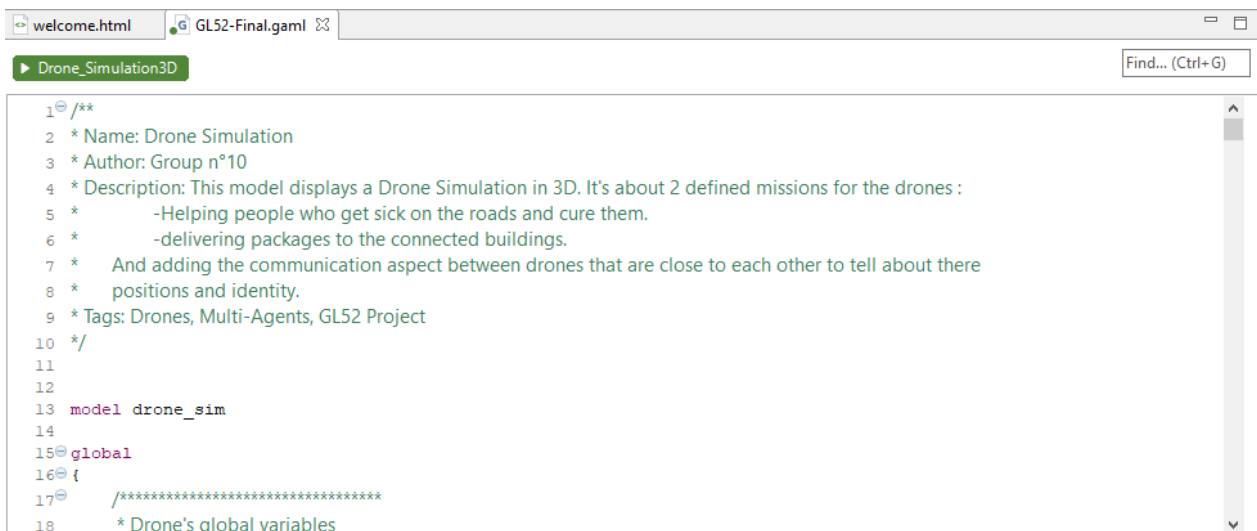
<sup>1</sup> <https://github.com/gama-platform/gama/releases/tag/v1.7-rc2>



It contains a project called “GL52”, that have 3 important subfolders:

- Images:
  - Grass.png: a grass image used as the ground of the city.
- Includes:
  - 3 shapefiles (.shp) that represent buildings, roads and bounds to form a city using GIS (Geographic Information System).
  - shortest\_paths.csv: contains the shortest paths data using several types of optimizer for the shortest path computation:
    - Dijkstra
    - Bellmann
    - AStar
    - Floyd Warshall
- Models:
  - GL52-Final.gaml: the main file of this project (3D Drone simulation).

After opening the model file, a window that has a running button and the code<sup>2</sup> will appear:



```
1 /**
2  * Name: Drone Simulation
3  * Author: Group n°10
4  * Description: This model displays a Drone Simulation in 3D. It's about 2 defined missions for the drones :
5  *             -Helping people who get sick on the roads and cure them.
6  *             -delivering packages to the connected buildings.
7  *             And adding the communication aspect between drones that are close to each other to tell about there
8  *             positions and identity.
9  * Tags: Drones, Multi-Agents, GL52 Project
10 */
11
12
13 model drone_sim
14
15 global
16 {
17     /*****
18      * Drone's global variables
```

Figure 11 : Model file

---

<sup>2</sup> The code is commented to help the reader understanding how the simulation works

When running the program, the user will have in front of him 5 windows:

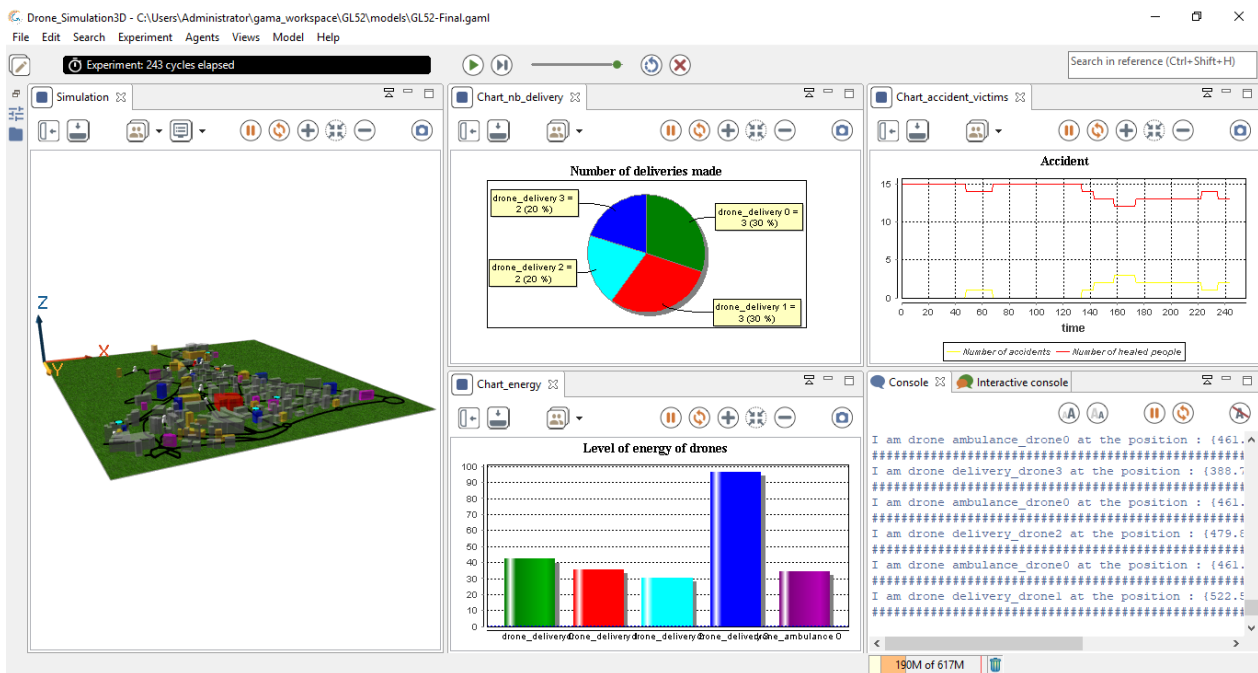
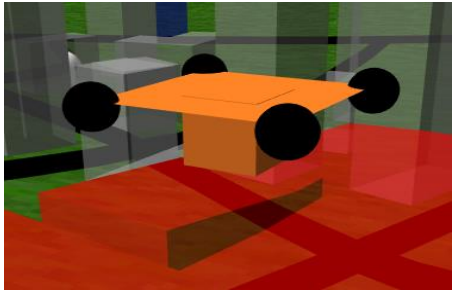
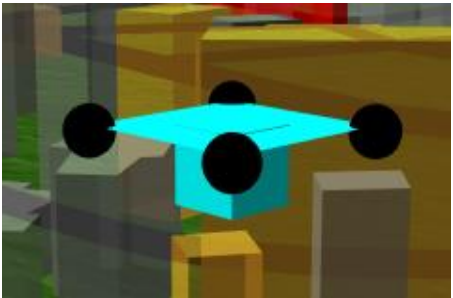
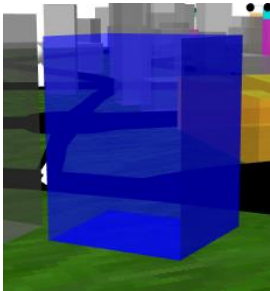
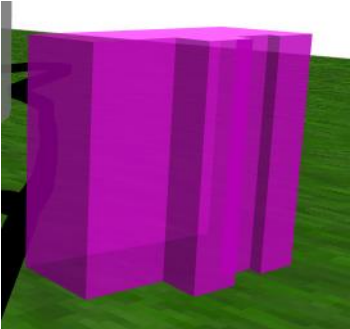

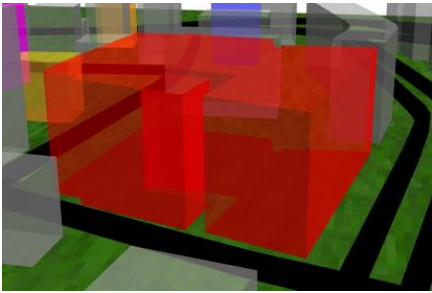
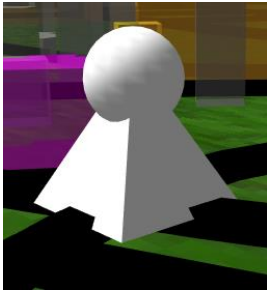
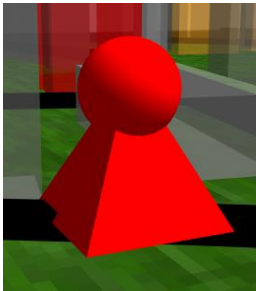


Figure 12 : Dashboard project

- **Simulation:** a 3D simulation of the different missions of drones.
- **Chart\_nb\_delivery:** A pie chart of number of delivery made by every delivery drones.
- **Chart\_energy:** A histogram that represents the level of energy of initialized drones.
- **Chart\_accident\_victims:** A series chart that represents the number of victims and healed people.
- **Console:** where the user can see the interaction and the communication between:
  - drone <--> drone
  - drone <--> building

The simulation will initialize 7 agents:

Ambulance drone	 A 3D model of an orange ambulance drone with four black propellers, positioned on a red and brown textured ground.
Delivery drone	 A 3D model of a cyan delivery drone with four black propellers, positioned in a simulated urban environment with grey and yellow buildings.
Charging station	 A 3D model of a blue, rectangular charging station with a transparent front panel, standing on a green grassy field.
Docking station	 A 3D model of a purple, rectangular docking station with a transparent front panel, standing on a green grassy field.
Connected buildings	 A 3D model of a yellow, rectangular building with a U-shaped cutout, standing on a green grassy field.

Hospital	
People	<p>Normal state:</p> 
	<p>Victim :</p> 

To see the simulation, we have created a YouTube playlist of 3 important project development phases (link down below<sup>3</sup>)

---

<sup>3</sup> [https://www.youtube.com/playlist?list=PLHOgBlrc10YkoE0\\_Y1QVv\\_oCFU9P7tNM0](https://www.youtube.com/playlist?list=PLHOgBlrc10YkoE0_Y1QVv_oCFU9P7tNM0)

## 6 Performance needs

This simulation can be launch in any OS (Windows, Linux, Mac) in few seconds. It reacts in real time if we adjust the cycle's simulation. The folder of this project contains 12 files, divided into 3 subfolders, that has a size of 821 Ko.

## 7 Development constraint

We developed our project using custom GIS files. If the user wants to upload new GIS files, he must modify the data of the “buildings” shapefile, otherwise GAMA will raise an exception.

## 8 References

- Multi-Agent System comparison,  
<https://www.lri.fr/~caillou/c1bDeveloppement.pdf>
- GAMA platform forum, <https://groups.google.com/forum/#!forum/gama-platform>
- GAMA documentation,  
<https://github.com/gamaplatform/gama/wiki/resources/pdf/docGAMAv17.pdf>
- GAMA installation, [http://gama-platform.org/getting\\_started](http://gama-platform.org/getting_started)
- GIS wiki, [https://en.wikipedia.org/wiki/GIS\\_file\\_formats](https://en.wikipedia.org/wiki/GIS_file_formats)