# SOBA Documentation

***Release 1***

**Eduardo Merino**

**Jul 01, 2019**

# Contents

# CHAPTER 1

# SOBA Overview

SOBA (Simulation of Occupancy Based on Agents) is a tool of simulation of occupancy in buildings implemented in Python.

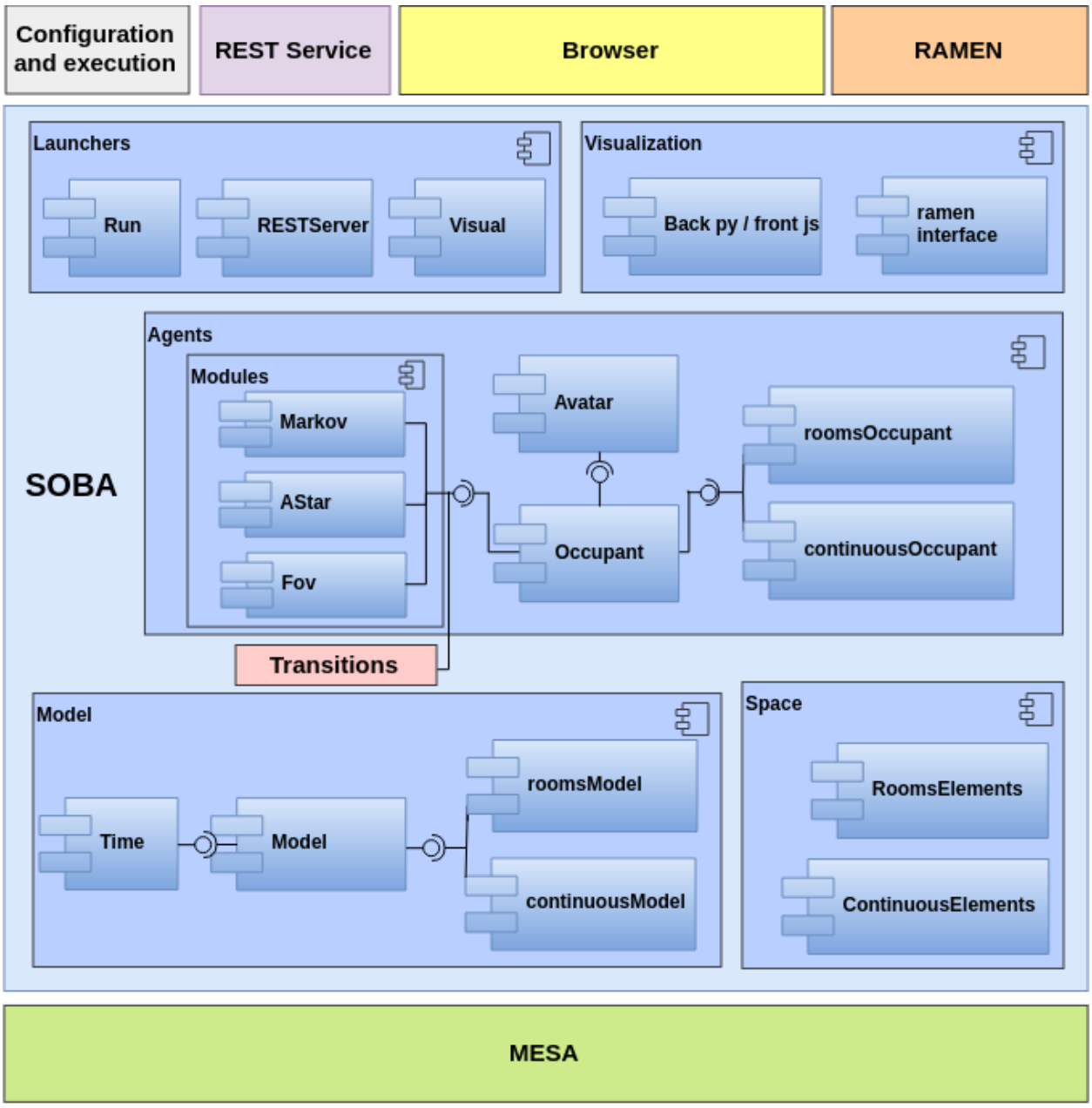
This software is useful for studies that require working with crowds, mainly in buildings. For example, SOBA has been used to carry out studies on the improvement of evacuations and for the improvement of energy efficiency in buildings.

The simulations are configured by declaring one or more types of occupants, with specific and definable behavior and a physical space. Regarding space, two different models are provided: a simplified model with a room defined by rooms, and a model with a continuous space. The simulation and results can be evaluated both in real time and post-simulation.

It is provided as open source software:

`Github repository`: https://github.com/gsi-upm/soba

## 1.1 Arquitecture Description




### 1.1.1 SEBA Components

- MESA. Mesa is an Apache2 licensed agent-based modeling (or ABM) framework in Python. It allows users to create agent-based models using built-in core components (such as spatial grids and agent schedulers) or customized implementations.
- Transitions. This external package is a lightweight, object-oriented state machine implementation in Python.
- RAMEN. It is an agent-based social simulation visualization tool for indoor crowd analytics based on the library Three.js. It allows to visualize a social simulation in a 3D environment and also to create the floor plan of a

building.

- Browser. Using a browser a simple visualization can be made to know the performance of the simulation. This is also useful for debugging.
- REST Service. The software provide an API defined as a REST service (Get, Post, Pull and Push methods are defined) to interact with the simulation.

### 1.1.2 SEBA Modules

SOBA is implemented through 5 modules which group independent components with a related function.

- ***Model.***
  - **Model**. Base Class to create simulation models. It creates and manages space and agents. The model creates and manages space and agents, provides a scheduler that controls the agents activation regime, stores model-level parameters and serves as a container for the rest of components.
  - **Continuous Model**. Base Class to create simulation models on a continuous space.
  - **Rooms Model**. Base Class to create simulation models on a simplified space based on rooms.
  - **Time**. Component of time management during the simulation in sexagesimal units and controller of the scheduler during the simulation.
- ***Agents.***
  - **Occupant**. An object of the Occupant class is a type of agent developed and characterized to simulate the behavior of crowds in buildings. The occupants are agents with their activity defined by markov states.
  - **Continuous Occupant**.This class enables to create occupants that are modelled with a continuous space models. based on considering a scaled grid (x, y). Cell size of 0.5m ^ 2 by default.
  - **Rooms Occupant**. This class enables to create occupants that are modelled with a simplified models based on a discrete space associated with rooms.
  - **Avatar**. It enables to create avatars that represent virtual occupants, that is, they are not controlled by the simulation but by an API Rest, providing a means of interaction between simulation and real human participation.
  - ***Agents modules.***
    - **Markov**. Base class to models the activity of the agents by means of Markovian behavior.
    - **AStar**. Auxiliar class used by the occupants to move in the building.
    - FOV.This component is a permissive field of view, which is useful to define the occupant visibility.
- ***Space.***
  - **Grid**. The space where the agents are situated and where they perform their actions is defined by means of a grid with coordinates (x, y).
  - **ContinuousItems**. Various classes that define the representation of physical space objects in the continuous space model.
  - **RoomsItems**. Various classes that define the representation of physical space objects in the simplificated space model based on rooms.
- ***Visualization.***

  - **Back.py** and **front js**. Two components provide a simple mechanism to represent the model in a web interface, based on HTML rendering though a server interface, implemented with web sockets. Connection between a JS file and a class py by means of parameters rendering.
  - **ramenInterface**. Component to make the conexión with the RAMEN API.

- ***Launchers*.**
  - **RESTServer**. Specification of the REST service server deployment.
  - **Visual**. Manages the execution of the simulation in Browser by launch JS (front)/.py (back) files
  - **Run**. Provides the execution of the simulation from terminal.

CHAPTER 2

# How install

## 2.1 SOBA package instalation

To install SOBA the best option is to use the package management system PIP. For this, we execute the following command.

```
$ pip install soba
```

In case of error, this other command should be used, ensuring to have installed python 3 version and pip 3 version.

```
$ pip3 install soba
```

## 2.2 First run

Now let's execute an example. In this first execution we opted for the continuous model. First, we must first download the repository of the github SOBA project and access to the example directory.

```
$ git clone https://github.com/gsi-upm/soba

$ cd soba/projects/examples
```

Then, execute the run file.

```
$ python continuousExample.py
```

or

```
$ python3 continuousExample.py
```

Different options are provided for execution:

1. Visual mode

```
$ python3 continuousExample.py -v
```

1.1 Launching REST Server

```
$ python3 continuousExample.py -v -s
```

1.2 Using RAMEN tool

```
$ python3 continuousExample.py -v -r
```

2. Batch mode

```
$ python3 continuousExample.py -b
```

2.1 Launching REST Server

```
$ python3 continuousExample.py -b -s
```

2.2 Using RAMEN tool

```
$ python3 continuousExample.py -b -r
```

CHAPTER 3

# Introductory Tutorial

## 3.1 Instalation

If the SOBA package is not yet installed, we must first do so. To install SOBA the best option is to use the package management system PIP. For this, we execute the following command.

```
$ pip install soba
```

In case of error, this other command should be used, ensuring to have installed python 3 and pip 3.

```
$ pip3 install soba
```

## 3.2 Tutorial

The SOBA tool can be provided to be used directly on two scenarios:

1. Generic case with a space defined as a grid of a given square size (by default, half a meter on each side).
2. Simplified case with a room defined by rooms, to perform simulations in simplified buildings that require less consumption of resources and specifications.

An introductory tutorial will be presented for each case, although most parameters are common or similar.

SOBA enables the performance of the simulations in two modes:

1. With visual representation.
2. In batch mode.

In the tutorials, the small modifications required to use each posibility are reflected.

In addition, two added mechanisms are provided to interact with the simulation:

1. Use an API on a REST server to obtain information and create and manage avatars.
2. use the external tool RAMEN for advanced 3D-visualization on Three.js.

IMPORTANT NOTE: The .py files described in this tutorial are available in the github repository https://github.com/gsi-upm/soba/tree/master/projects/basicExamples

## 3.3 Implementing a sample model with continuous space

Once soba is installed, the implementation can be started. First we define the generic parameters to both types of scenario.

1.- We define the characteristics of the occupants

```
from collections import OrderedDict
#JSON to store all the informacion.
jsonsOccupants = []

#Number of occupants
N = 3

#Definition of the states
states = OrderedDict([('Leaving','out'), ('Resting', 'sofa'), ('Working in my␣
↪laboratory', 'wp')])

#Definition of the schedule
schedule = {'t1': "08:01:00", 't2': "08:10:00", 't3': "08:20:00"}

#Possible Variation on the schedule
variation = {'t1': "00:01:00", 't2': "00:01:00", 't3': "00:01:00"}

#Probability of state change associated with the Markovian chain as a function of the␣
↪temporal period
markovActivity = {
    '-t1': [[100, 0, 0], [0, 0, 0], [0, 0, 0]],
    't1-t2': [[0, 0, 100], [0, 50, 50], [0, 50, 50]],
    't2-t3': [[100, 0, 0], [0, 50, 50], [0, 50, 50]],
    't3-': [[0, 0, 100], [100, 0, 0], [0, 0, 100]]
}

#Time associated to each state (minutes)
timeActivity = {
    '-t1': [3, 0, 0], 't1-t2': [3, 3, 3], 't2-t3': [3, 3, 3], 't3-': [3, 3, 3]
}


#Time variation associated to each state (minutes)
timeActivityVariation = {
    '-t1': [1, 0, 0], 't1-t2': [1, 1, 1], 't2-t3': [1, 1, 1], 't3-': [1, 1, 1]
}

#Store the information
jsonOccupant = {'type': 'example' , 'N': N, 'states': states , 'schedule': schedule,
↪'variation': variation,
'markovActivity': markovActivity, 'timeActivity': timeActivity, "timeActivityVariation
↪": timeActivityVariation}

jsonsOccupants.append(jsonOccupant)
```

2.- We define the building plan or the distribution of the space.

```
import soba.visualization.ramen.mapGenerator as ramen

with open('labgsi.blueprint3d') as data_file:
    jsonMap = ramen.returnMap(data_file)
```

3.- We implement a Model inheriting a base class of SOBA.

```
from soba.models.continuousModel import ContinuousModel
from time import time

class ModelExample(ContinuousModel):

    def __init__(self, width, height, jsonMap, jsonsOccupants, seed = int(time())):
        super().__init__(width, height, jsonMap, jsonsOccupants, seed = seed,␣
↪timeByStep = 60)
        self.createOccupants(jsonsOccupants)

    def step(self):
        if self.clock.clock.hour > 17:
            self.finishSimulation = True
        super().step()
```

4.- We call the execution methods.

```
4.1-With visual representation.
```

```
import soba.run
import sys
from optparse import OptionParser

parameters = {'width': 40, 'height': 40, 'jsonMap': jsonMap, 'jsonsOccupants':␣
↪jsonsOccupants}

sys.argv = []
sys.argv.append("-1")
sys.argv.append("-v")

soba.run.run(ModelExample, parameters, visualJS="example.js")
```

```
SOBA is running
Interface starting at http://127.0.0.1:7777
Socket opened!
{"type":"get_params"}
{"type":"reset"}
{"type":"get_step","step":1}
01:08:01:00
{"type":"get_step","step":2}
01:08:02:00
{"type":"get_step","step":3}
01:08:03:00
{"type":"get_step","step":4}
01:08:04:00
{"type":"get_step","step":5}
01:08:05:00
{"type":"get_step","step":6}
01:08:06:00
{"type":"get_step","step":7}
```

(continues on next page)

(continued from previous page)

```
01:08:07:00
{"type":"get_step","step":8}
01:08:08:00
{"type":"get_step","step":9}
01:08:09:00
{"type":"get_step","step":10}
01:08:10:00
{"type":"get_step","step":11}
01:08:11:00
{"type":"get_step","step":12}
01:08:12:00
{"type":"get_step","step":13}
01:08:13:00
{"type":"get_step","step":14}
01:08:14:00
{"type":"get_step","step":15}
01:08:15:00
{"type":"get_step","step":16}
01:08:16:00
{"type":"get_step","step":17}
01:08:17:00
{"type":"get_step","step":18}
01:08:18:00
{"type":"get_step","step":19}
01:08:19:00
{"type":"get_step","step":20}
01:08:20:00
{"type":"get_step","step":21}
01:08:21:00
{"type":"get_step","step":22}
01:08:22:00
{"type":"get_step","step":23}
01:08:23:00
{"type":"get_step","step":24}
01:08:24:00
{"type":"get_step","step":25}
01:08:25:00
{"type":"get_step","step":26}
01:08:26:00
{"type":"get_step","step":27}
01:08:27:00
{"type":"get_step","step":28}
01:08:28:00
{"type":"get_step","step":29}
01:08:29:00
{"type":"get_step","step":30}
01:08:30:00
{"type":"get_step","step":31}
01:08:31:00
{"type":"get_step","step":32}
01:08:32:00
{"type":"get_step","step":33}
01:08:33:00
{"type":"get_step","step":34}
01:08:34:00
{"type":"get_step","step":35}
01:08:35:00
```

(continues on next page)

(continued from previous page)

```
{"type":"get_step","step":36}
01:08:36:00
{"type":"get_step","step":37}
01:08:37:00
{"type":"get_step","step":38}
01:08:38:00
{"type":"get_step","step":39}
01:08:39:00
{"type":"get_step","step":40}
01:08:40:00
{"type":"get_step","step":41}
01:08:41:00
{"type":"get_step","step":42}
01:08:42:00
{"type":"get_step","step":43}
01:08:43:00
{"type":"get_step","step":44}
01:08:44:00
{"type":"get_step","step":45}
01:08:45:00
{"type":"get_step","step":46}
01:08:46:00
{"type":"get_step","step":47}
01:08:47:00
{"type":"get_step","step":48}
01:08:48:00
{"type":"get_step","step":49}
01:08:49:00
{"type":"get_step","step":50}
01:08:50:00
{"type":"get_step","step":51}
01:08:51:00
{"type":"get_step","step":52}
01:08:52:00
{"type":"get_step","step":53}
01:08:53:00
{"type":"get_step","step":54}
01:08:54:00
{"type":"get_step","step":55}
01:08:55:00
{"type":"get_step","step":56}
01:08:56:00
{"type":"get_step","step":57}
01:08:57:00
{"type":"get_step","step":58}
01:08:58:00
```

```
4.1- Bacth mode.
```

```
import soba.run
import sys
#Fixed parameters during iterations
fixed_params = {"width": 40, "height": 40, "jsonMap": jsonMap, "jsonsOccupants":␣
↪jsonsOccupants}
#Variable parameters to each iteration
variable_params = {"seed": range(10, 500, 10)}
```

(continues on next page)

(continued from previous page)

```
sys.argv = []
sys.argv.append("-1")
sys.argv.append("-b")

soba.run.run(ModelExample, fixed_params, variable_params)
```

## 3.4 Implementing a sample model with simplified space

Once soba is installed, the implementation can be started. First we define the generic parameters to both types of scenario.

1.- We define the characteristics of the occupants

```
from collections import OrderedDict
#JSON to store all the informacion.
jsonsOccupants = []

#Number of occupants
N = 3

#Definition of the states
states = OrderedDict([('out','Pos1'), ('Working in my laboratory', {'Pos2': 1, 'Pos3
↪': 2})])

#Definition of the schedule
schedule = {'t1': "08:01:00", 't2': "08:10:00", 't3': "08:20:00"}

#Possible Variation on the schedule
variation = {'t1': "00:01:00", 't2': "00:01:00", 't3': "00:01:00"}

#Probability of state change associated with the Markovian chain as a function of the␣
↪temporal period
markovActivity = {
    '-t1': [[100, 0, 0], [0, 0, 0], [0, 0, 0]],
    't1-t2': [[0, 0, 100], [0, 50, 50], [0, 50, 50]],
    't2-t3': [[100, 0, 0], [0, 50, 50], [0, 50, 50]],
    't3-': [[0, 0, 100], [0, 100, 0], [0, 100, 0]]
}

#Time associated to each state (minutes)
timeActivity = {
    '-t1': [3, 0, 0], 't1-t2': [3, 3, 3], 't2-t3': [3, 3, 3], 't3-': [3, 3, 3]
}


#Time variation associated to each state (minutes)
timeActivityVariation = {
    '-t1': [1, 0, 0], 't1-t2': [1, 1, 1], 't2-t3': [1, 1, 1], 't3-': [1, 1, 1]
}

#Store the information
jsonOccupant = {'type': 'example' , 'N': N, 'states': states , 'schedule': schedule,
↪'variation': variation,
```

(continues on next page)

(continued from previous page)

```
                'markovActivity': markovActivity, 'timeActivity': timeActivity}
jsonsOccupants.append(jsonOccupant)
```

2.- We define the building plan or the distribution of the space.

```
jsonMap = {
  'Pos1': {'entrance':'', 'conectedTo': {'U':'Pos2'}, 'measures': {'dx':2, 'dy':2}},
  'Pos2': {'measures': {'dx':3, 'dy':3.5}, 'conectedTo': {'R':'Pos3'}},
  'Pos3': {'measures': {'dx':3, 'dy':3.5}}
}
```

3.- We implement a Model inheriting a base class of SOBA.

```
from soba.models.roomsModel import RoomsModel
import datetime as dt

class ModelExample(RoomsModel):

    def __init__(self, width, height, jsonMap, jsonsOccupants, seed = int(time())):
        super().__init__(width, height, jsonMap, jsonsOccupants, seed = seed)

    def step(self):
        if self.clock.clock.day > 3:
            self.finishSimulation = True
        super().step()
```

4.- We call the execution methods. 4.1- With visual representation.

```
import soba.run
import sys

cellW = 4
cellH = 4

sys.argv = []
sys.argv.append("-1")
sys.argv.append("-v")

parameters = {'width': cellW, 'height': cellH, 'jsonMap': jsonMap, 'jsonsOccupants':␣
↪jsonsOccupants}
soba.run.run(ModelExample, parameters, visualJS="example.js")
```

```
4.1- Bacth mode.
```

```
#Fixed parameters during iterations
fixed_params = {"width": cellW, "height": cellH, "jsonMap": jsonMap, "jsonsOccupants
↪": jsonsOccupants}
#Variable parameters to each iteration
variable_params = {"seed": range(10, 500, 10)}

sys.argv = []
sys.argv.append("-1")
sys.argv.append("-b")

soba.run.run(ModelExample, fixed_params, variable_params)
```

## 3.5 Running the simulation using the terminal

```
$ git clone https://github.com/gsi-upm/soba

$ cd soba/projects/examples
```

Then, execute the run file.

```
$ python continuousExample.py
```

or

```
$ python3 continuousExample.py
```

Different options are provided for execution:

1. Visual mode

```
$ python3 continuousExample.py -v
```

1.1 Launching REST Server

```
$ python3 continuousExample.py -v -s
```

1.2 Using RAMEN tool

```
$ python3 continuousExample.py -v -r
```

2. Batch mode

```
$ python3 continuousExample.py -b
```

2.1 Launching REST Server

```
$ python3 continuousExample.py -b -s
```

2.2 Using RAMEN tool

```
$ python3 continuousExample.py -b -r
```

CHAPTER 4

# APIs

## 4.1 Model Module Documentation

### 4.1.1 Model

**class** soba.models.generalModel.**GeneralModel**(*width*, *height*, *seed=1561972004*, *timeByStep=60*)

Base Class to create simulation models. It creates and manages space and agents.

**Attributes:** height: Height in number of grid cells. width: Width in number of grid cells. schedule: BaseScheduler object for agent activation. grid: Grid object to implement space. running: Parameter to control the models execution. NStep: Measure of the number of steps. occupants: List of Occupant objects created. agents: List of the all Agent objects created. asciMap: Representation of the map as ASCI used to get FOV information. seed: Seed employ in random generations. finishSimulation: Parameter to stop the software simulation.

**Methods:** finishTheSimulation: Finish with the execution of the simulation software. run_model: Model execution. step: Execution of the scheduler steps.

**finishTheSimulation**()

Finish with the execution of the simulation software.

**step**()

Main step of the simulation, execution of the scheduler steps.

### 4.1.2 Continuous Model

**class** soba.models.continuousModel.**ContinuousModel**(*width*, *height*, *jsonMap*, *jsonsOccupants*, *seed=datetime.datetime(2019, 7, 1, 9, 6, 44, 185560)*, *scale=0.5*, *timeByStep=60*)

**Base Class to create simulation models on a simplified space based on rooms.**

**Attributes:** Those inherited from the Occupant class. rooms: List of Room objects. walls: List of Wall objects. pois: List of Pois objects. generalItems: List of GeneralItem objects. doors: List of Door objects.

**Methods:** createOccupants: Create occupants of the ContinuousOccupant type. getScaledCoordinate: Gets the value of a coordinate or a cell size scaled. setMap: Define the map plane distribution. getAsciMap: Get the plane of the grid in an ASCII format, such as a matrix, to apply the fov algorithm. thereIsClosedDoor: Check if one door is closed or open. getDoorInPos: Get a Door object in a position given. getOccupantsPos: Get a Occupant objects in a position given. thereIsOccupant: Check if there is any Occupant object in a position given. getOccupantId: Get the occupant with the unique_id given. getPOIsId: Get the pois object with the id given. getPOIsPos: Get the position of a poi object. checkFreePOI: Check if one Point of interest is free (there is no occupant on this). xyInGrid: Check if one position is inside the grid. nearPos: Check if two positions are consecutive.

**checkFreePOI**(*p*)

**Check if one Point of interest is free (there is no occupant on this).**

**Args:** p: Poi object.

Return: True (yes), False (no).

**createAvatar**(*idAvatar*, *pos*, *color='red'*, *initial_state='walking'*)
Create one avatar

**createOccupants**(*jsonsOccupants*)

**Create occupants of the ContinuousOccupant type.**

**Args:** jsonOccupants: Json of description of the occupants.

**getAsciMap**()
Get the plane of the grid in an ASCII format, such as a matrix, to apply the fov algorithm.

**getDoorInPos**(*pos*)

**Get a Door object in a position given.**

**Args:** pos: Position of the door as (x, y).

Return: Door object or false.

**getOccupantId**(*Id*)

**Get the occupant with the unique_id given.**

**Args:** Id: Unique_id given as number

Return: Occupant object or False.

**getOccupantsPos**(*pos*)

**Get a Occupant objects in a position given.**

**Args:** pos: Position as (x, y).

Return: List of Cccupant objects.

**getPOIsId**(*poiId*)

**Get the pois object with the id given.**

**Args:** poiId: poi id given as number

Return: list of pois object or False.

**getPOIsPos**(*poiPos*)

**Get the position of a poi object.**

**Args:** poiPos: position as (x, y)

Return: poi object or False.

**getScaledCoordinate**(*coordenate*, *scale*)

**Gets the value of a coordinate or a cell size scaled to the size of the grid in relation to a given scale.**

**Args:** coordenate: Coordenate value to be scale scale: Value of the scale to be applied.

Return: value of the coordinate.

**nearPos**(*pos1*, *pos2*)

**Check if two positions are consecutive.**

**Args:** pos1: Position as (x, y). pos2: Position as (x, y).

Return: True (yes), False (no).

**setMap**(*jsonMap*, *scale*)

**Define the map plane distribution.**

**Args:** jsonMap: Json of description of the map plane. scale: Value of the scale.

**step**()

Main step of the simulation, execution of the scheduler steps.

**thereIsClosedDoor**(*pos*)

**Check if one door is closed or open.**

**Args:** pos: Position of the door as (x, y).

Return: State of the door as boolean.

**thereIsOccupant**(*pos*)

**Check if there is any Occupant object in a position given.**

**Args:** pos: Position as (x, y).

Return: True (yes), False (no).

**xyInGrid**(*pos*)

**Check if one position is inside the grid.**

**Args:** pos: Position as (x, y).

Return: True (yes), False (no).

### 4.1.3 Rooms Model

**class** soba.models.roomsModel.**RoomsModel**(*width*, *height*, *jsonRooms*, *jsonsOccupants*, *seed=1561972004*, *timeByStep=60*)

Base Class to create simulation models on a simplified space based on rooms.

**Attributes:** Those inherited from the Occupant class. rooms: List of Room objects. Those inherited from the Occupant class. rooms: List of Room objects. walls: List of Wall objects. pois: List of Pois objects. generalItems: List of GeneralItem objects. doors: List of Door objects.

**Methods:** createOccupants: Create occupants of the RoomsOccupant type. createRooms: Create the rooms in the grid space. setMap: Define the map plane distribution. createDoors: Create the doors in the grid space. createWalls: Create the walls in the grid space. thereIsClosedDoor: Check if one door is closed or open. thereIsOtherOccupantInRoom: Evaluate if there is another occupant apart of the past as a parameter in a room. thereIsSomeOccupantInRoom: Evaluate if there is any occupant in a room. thereIsOccupantInRoom: Evaluate if there is one specific occupant in a room. getRoom: Get one Room Object by position. pushAgentRoom: Add one agent to a room. popAgentRoom: Remove one agent from a room. openDoor: Change the status of the door to open. closeDoor: Change the status of the door to close.

**closeDoor**(*agent*, *room1*, *room2*)

**Change the status of the door to close (False).**

**Args:** agent: Agent object which want to close the door. room1, room2: The two common rooms of the door as Room object.

**createDoors**()
Create the doors in the grid space.

**createOccupants**(*jsonsOccupants*)

**Create occupants of the RoomsOccupant type.**

**Args:** jsonOccupants: Json of description of the occupants.

**createRooms**(*jsonRooms*)

**Create the rooms in the grid space.**

**Args:** jsonRooms: Json of description of the map plane.

**createWalls**()
Create the walls in the grid space.

**getRoom**(*pos*)

**Get one Room Object by position.**

**Args:** pos: Position of the grid as (x,y).

Return: Room Object or False

**openDoor**(*agent*, *room1*, *room2*)

**Change the status of the door to open (True).**

**Args:** agent: Agent object which want to open the door. room1, room2: The two common rooms of the door as Room object.

**popAgentRoom**(*agent*, *pos*)

**Remove one agent from a room.**

**Args:** agent: Agent Object pos: Position of the grid as (x,y)

**pushAgentRoom**(*agent*, *pos*)

**Add one agent to a room.**

**Args:** agent: Agent Object pos: Position of the grid as (x,y)

**setMap**()
Define the map plane distribution.

**step**()
Main step of the simulation, execution of the scheduler steps.

**thereIsClosedDoor**(*beforePos*, *nextPos*)

**Check if one door is closed or open.**

**Args:** beforePos, nextPos: The two common positions of the door as (x, y).

Return: True (closed) or False (opened).

**thereIsOccupantInRoom**(*room*, *agent*)

**Evaluate if there is one specific occupant in a room.**

**Args:** room: Room object to be checked.

Return: True (yes), False (no)

**thereIsOtherOccupantInRoom**(*room*, *agent*)

**Evaluate if there is another occupant apart of the past as a parameter in a room.**

**Args:** room: Room object to be checked. agent: Occupant object to be ignored.

Return: True (yes), False (no)

**thereIsSomeOccupantInRoom**(*room*)

**Evaluate if there is any occupant in a room.**

**Args:** room: Room object to be checked.

Return: True (yes), False (no)

### 4.1.4 Time Control

**class** soba.models.timeControl.**Time**(*model*, *timeByStep=60*, *day=1*, *hour=7*, *minute=50*, *seg=0*, *microsecond=0*)

Class that inherits of the Agent class to manage the time in sexagesimal units.

**Attributes:** clock: Clock for time monitoring during simulation. timeByStep: Time in seconds associated with each step. startDay: Time of the beginning of a day in the simulation. endDay: Time of the end of a day in the simulation.

**Methods:** step: Advance of the clock in a step. increaseTime: Increase the value of the clock a given time. decreaseTime: Decrease the value of the clock a given time.

**decreaseTime**(*days=0*, *hours=0*, *minutes=0*, *seconds=0*, *microseconds=0*)

**Decrease the value of the clock a given time.**

**Args:** seconds, days, hours, minutes, microseconds: Time value to be decrease.

Return: The new Clock object.

**increaseTime**(*seconds=0*, *days=0*, *hours=0*, *minutes=0*, *microseconds=0*)

**Increase the value of the clock a given time.**

**Args:** seconds, days, hours, minutes, microseconds: Time value to be increase.

Return: The new Clock object.

**step**()

Advance of the clock in a step.

## 4.2 Agents Module Documentation

### 4.2.1 Occupant

**class** soba.agents.occupant.**Occupant**(*unique_id*, *model*, *json*, *speed=0.71428*)

Base class to models occupants as Occupant objects. The occupants are agents with their activity defined by markov states.

**Attributes:** color: Color with which the occupant will be represented in the visualization. positionByState: Position associated to each state for an occupant. timeActivity: Time that is required to complete an activity (state) in minutes. schedule: Activity periods (hours:minutes). states: States of the occupant. machine: State machine defined by the attribute 'states'. movements: List of movements that will be followed by the occupant. pos_to_go: Position to which the occupant wishes to move. markov_machine: Object of the Markov class that regulates markovian behavior.

**Methods:** setTodaySchedule: Calculate and define the schedules of the occupants. start_activity: Defines the actions that are made when a state is started. finish_activity: Defines the actions that are made when a state is finished. changeSchedule: Force a possible change of state to reach a certain end of period. getPeriod: Get the temporary period in which the occupant is. step: Method invoked by the Model scheduler in each step. Step common to all occupants.

**changeSchedule**()

Force a possible change of state to reach a certain end of period.

**Return:** True if the period has been changed, False otherwise.

**finish_activity**()

Defines the actions that are made when a state is finished.

**getPeriod**()

Get the temporary period in which the occupant is.

**Return:** Current period as String

**setTodaySchedule**()

Calculate and define the schedules of the occupants applying the information provided and normal Gaussian variations.

**start_activity**()

Defines the actions that are made when a state is started. Default, this method calculates the value of the attributes 'time_activity' and 'movements' corresponding to the new state.

**step**()

Method invoked by the Model scheduler in each step. Step common to all occupants.

### 4.2.2 Continuous Occupant

**class** soba.agents.continuousOccupant.**ContinuousOccupant**(*unique_id*, *model*, *json*, *speed=0.71428*)

This class enables to create occupants that are modelled with a continuous space models. based on considering a scaled grid (x, y). Cell size of 0.5m ^ 2 by default. The occupants are agents with their activity defined by markov states.

**Attributes:** Those Inherited from the Occupant class. fov: List of positions (x, y) that the occupant can see.

**Methods:** getPosState: Auxiliary method to distribute the occupants between the points of interests with same id for more than one occupant. getWay: Invocation of the AStar resource to calculate the optimal path.

getPlaceToGo: Obtaining the position associated with the current state. posInMyFOV: Check if a position is in my field of vision. evalAvoid: Check the future movement to be made by another agent to assess a possible collision. checkFreeSharedPOI: Get a free position of a shared point of interest if possible. checkCanMove: Get a new path in case of possible collision. evalCollision: Evaluate a possible collision with an agent and solve it if necessary by calculating another path. makeMovement: Carry out a movement: displacement between cells or reduction of the movement cost parameter. reportMovement: Auxiliary method to notify a movement giving its orientation and speed. checkLeaveArrive: Evaluates the entrance and exit of the building by an occupying agent. getFOV: Calculation of the occupant's field of vision, registered in the attribute fov. step: Method invoked by the Model scheduler in each step.

**checkCanMove**()

**Get a new path in case of possible collision.** Return: List of positions

**checkFreeSharedPOI**()

**Get a free position of a shared point of interest if possible.** Return: POI object

**checkLeaveArrive**()
Evaluates the entrance and exit of the building by an occupying agent.

**evalAvoid**(*otherAgent*)

**Check the future movement to be made by another agent to assess a possible collision.**

**Args:** otherAgent: The other agent to be avoid.

Return: Boolean

**evalCollision**()

**Evaluate a possible collision with an agent, invoking the evalAvoid method, and solve it if necessary by calculating an**
Return: True if the collision exists and is avoided, False otherwise.

**getFOV**()
Calculation of the occupant's field of vision, registered in the attribute fov

**getPlaceToGo**()

**Obtaining the position associated with the current state. It is invoked when you enter a new state.**
Return: Position as coordinate (x, y).

**getPosState**(*name*)

**Auxiliary method to distribute the occupants between the points of interests with same id for more than one occupan**

**Args:** name: Poi id/name.

Return: Position associated with this occupant.

**getWay**(*pos=None*, *pos_to_go=None*, *other=[]*)

**Invocation of the AStar resource to calculate the optimal path.**

**Args:** pos: Initial position, by default the current position of the occupant. pos_to_go: Final position, by default the value of the 'pos_to_go' attribute of the occupant. other: List of auxiliary positions given to be considered impenetrable by the occupants, that is, they will not be used by the AStar.

Return: List of positions (x, y).

**makeMovement**()
Carry out a movement: displacement between cells or reduction of the movement cost parameter.

**posInMyFOV**(*pos*)

**Check if the position is in my field of vision**

**Args:** pos: Position to be checked

Return: Boolean

**reportMovement**()
Auxiliary method to notify a movement giving its orientation and speed.

**step**()
Method invoked by the Model scheduler in each step. Evaluate if appropriate and, if so, perform: A change of state, a movement or advance in the cost of a movement, or an advance in the performance of an activity.

### 4.2.3 Rooms Occupant

**class** soba.agents.roomsOccupant.**RoomsOccupant**(*unique_id*, *model*, *json*, *speed=0.7*)
This class enables to create occupants that are modelled with a simplified models based on a discrete space associated with rooms. The occupants are agents with their activity defined by markov states.

**Attributes:** Those inherited from the Occupant class.

**Methods:** getPosState: Auxiliary method to distribute the occupants between the rooms shared by more than one occupant object. getWay: Invocation of the AStar resource to calculate the optimal path. occupantMovePos: Calculation of the control attributes that regulate the cost (steps) of the movement between rooms according to their size. getPlaceToGo: Obtaining the position associated with the current state. step: Method invoked by the Model scheduler in each step.

**getPlaceToGo**()

**Obtaining the position associated with the current state. It is invoked when you enter a new state.**
Return: Position as coordinate (x, y).

**getPosState**(*name*, *posAux*)

**Auxiliary method to distribute the occupants between the rooms shared by more than one occupant object.**

**Args:** name: State name. posAux: Name of the room associated with this state, string, or dictionary of room names with number of occupants. {'RoomName1': numberofOccupantsAssigned1, 'RoomName2': numberofOccupantsAssigned2... }

Return: Position associated with this occupant

**getWay**(*pos=None*, *pos_to_go=None*)

**Invocation of the AStar resource to calculate the optimal path.**

**Args:** pos: Initial position, by default the current position of the occupant. pos_to_go: Final position, by default the value of the 'pos_to_go' attribute of the occupant.

Return: List of positions (x, y).

**occupantMovePos**(*new_position*)

**Calculation of the control attributes that regulate the cost (steps) of the movement between rooms according to their**

**Args:** new_position: Room object to which it moves.

**step**()
Method invoked by the Model scheduler in each step. Evaluate if appropriate and, if so, perform: A change of state, a movement or advance in the cost of a movement, or an advance in the performance of an activity.

### 4.2.4 Avatar

**class** soba.agents.avatar.**Avatar**(*unique_id*, *model*, *initial_pos*, *color='red'*, *initial_state='walking'*)

This class enables to create avatars that represent virtual occupants, that is, they are not controlled by the simulation but by an API Rest. However, certain important aspects such as position in space inherit from the occupant class.

**Attributes:** model: Simulation model. unique_id: Unique avatar identifier as an occupant. fov: List of positions (x, y) that the avatar can see. state: Current avatar state. pos: Current avatar position. color: Color of the avatar in the visualization. shape: Shape of the avatar in the visualization.

**Methods:** getWay: Invocation of the AStar resource to calculate the optimal path. posInMyFOV: Check if a position is in my field of vision. makeMovementAvatar: Carry out a movement: displacement between cells. checkLeaveArrive: Notify the entrance and exit of the building by an occupying agent. getFOV: Calculation of the occupant's field of vision, registered in the attribute fov.

**checkLeaveArrive**()

Notify the entrance and exit of the building by an occupying agent.

**getFOV**()

Calculation of the occupant's field of vision, registered in the attribute fov

**getWay**(*pos=None*, *pos_to_go=None*, *other=[]*)

**Invocation of the AStar resource to calculate the optimal path.**

**Args:** pos: Initial position, by default the current position of the occupant. pos_to_go: Final position, by default the value of the 'pos_to_go' attribute of the occupant. other: List of auxiliary positions given to be considered impenetrable by the occupants, that is, they will not be used by the AStar.

Return: List of positions (x, y).

**makeMovementAvatar**(*pos*)

**Carry out a movement: displacement between cells.**

**Args:** pos: Position to be moved.

**posInMyFOV**(*pos*)

**Check if the position is in my field of vision**

**Args:** pos: Position to be checked

Return: Boolean

**step**()

Method invoked by the Model scheduler in each step. Step common to all occupants.

### 4.2.5 Agent Modules

#### AStar

soba.agents.resources.aStar.**canMovePos**(*model*, *cellPos*, *posAux*, *others=[]*)

**Evaluate if a position is reachable in a continuous space.**

**Args:** model: Model which invokes the algorithm. cellPos: a first one position given as (x, y). posAux: a second one position given as (x, y). others: List of auxiliary positions given to be considered impenetrable, that is, they will not be used by the AStar.

Return: List of positions (x, y).

soba.agents.resources.aStar.**getConectedCellsContinuous**(*model*, *cell*, *others*)

**Gets a list of connected cells in a continuous space.**

**Args:** model: Model which invokes the algorithm. cell: cell object corresponding to the position. other: List of auxiliary positions given to be considered impenetrable, that is, they will not be used by the AStar.

Return: List of positions (x, y).

soba.agents.resources.aStar.**getConectedCellsRooms**(*model*, *cell*)

**Gets a list of connected cells in a space defined by rooms.**

**Args:** model: Model which invokes the algorithm. cell: cell object corresponding to the room.

Return: List of positions (x, y).

soba.agents.resources.aStar.**getPathContinuous**(*model*, *start*, *finish*, *other=[]*)

**Calculate the optimal path in the models with the space continuous.**

**Args:** model: Model which invokes the algorithm. start: Initial position. finish: Final position. other: List of auxiliary positions given to be considered impenetrable, that is, they will not be used by the AStar.

Return: List of positions (x, y).

soba.agents.resources.aStar.**getPathRooms**(*model*, *start*, *finish*)

**Calculate the optimal path in the models with the space defined by rooms.**

**Args:** model: Model which invokes the algorithm. start: Initial position. finish: Final position.

Return: List of positions (x, y).

soba.agents.resources.aStar.**print_progress**(*iteration*, *total*, *prefix=”*, *suffix=”*, *decimals=1*, *bar_length=100*)

Call in a loop to create terminal progress bar @params:

iteration - Required : current iteration (Int) total - Required : total iterations (Int) prefix - Optional : prefix string (Str) suffix - Optional : suffix string (Str) decimals - Optional : positive number of decimals in percent complete (Int) bar_length - Optional : character length of bar (Int)

### Markov

**class** soba.agents.resources.behaviourMarkov.**Markov**(*agent_aux*)

Base class to models the activity of the agents by means of Markovian behavior.

**Attributes:** agent: Agent that is controlled by this models.

**Methods:** runStep: Execute a Markovian state change by evaluating the initial state and the probabilities associated with each possible state. getNextState: Evaluate a random change based on the probabilities corresponding to each state.

**getNextState**(*markov_matrix*, *NumberCurrentState*)

**Evaluate a random change based on the probabilities corresponding to each state.**

**Args:** markov_matrix: Markov matrix corresponding to a certain moment. NumberCurrentState: Unique id as number of the current state.

**runStep**(*markov_matrix*)

**Execute a Markovian state change by evaluating the initial state and the probabilities associated with each possible s**

**Args:** markov_matrix: Markov matrix corresponding to a certain moment.

#### FOV

soba.agents.resources.fov.**FOV_RADIUS = 30000**
In the file aStar.py the filed of vision algorithm is implemented.

**class** soba.agents.resources.fov.**Map**(*map*)
Class to calculate the field of vision (fov).

**Attributes:** data: Map to which to apply the algorithm.

**Methods:** do_fov: Calculate the field of view from a position (x, y).

**More information:** http://www.roguebasin.com/index.php?title=Python_shadowcasting_implementation

**do_fov**(*x*, *y*)

**Calculate the field of view from a position (x, y).**

**Args:** x, y: Observer's position

Return: Array of sight positions.

soba.agents.resources.fov.**makeFOV**(*dungeon*, *pos*)

**Create the invocation object of the fov algorithm and invoke it.**

**Args:** dungeon: Array pos: Observer's position

Return: Array of sight positions.

## 4.3 Space Module Documentation

### 4.3.1 Items in continuous modeling

In the file continuousItems.py four classes are defined to implement the elements of the physical space in a continuous model:

-GeneralItem: Class that implements generic elements positioned on the map with the effect of being impenetrable. -Door: Class that implements bulding plane doors. -Wall: Class that implements building walls. -Poi: Class that implements points of interest where Occupancy objects perform certain actions.

**class** soba.space.continuousElements.**Door**(*model*, *pos1*, *pos2*, *rot*, *state=True*)

**Class that implements bulding plane doors.**

**Attributes:** state: Door status, open (True) or closed (False). pos1: First position to access to the door. pos2: Second position to access to the door. rot: Door orientation in the grid ('x' or 'y').

**Methods:** open: Change the status of the door to open. close: Change the status of the door to close.

**close**()
Change the status of the door to close (False)

**open**()
Change the status of the door to open (True)

**class** soba.space.continuousElements.**GeneralItem**(*model*, *pos*, *color=None*)

**Class that implements generic elements positioned on the map with the effect of being impenetrable.**

**Attributes:** pos: Position where the object is located. color: Color with which the object will be represented in the visualization.

**class** soba.space.continuousElements.**Poi**(*model*, *pos*, *ide*, *share=True*, *color=None*)

**Class that implements relevant elements in the simulations: points of interest where Occupancy objects perform certain a**

**Attributes:** pos: Position where the object is located. ide: Unique identifier associated with the point of interest. share: Define if the poi can be shared by more than one occupant. color: Color with which the object will be represented in the visualization.

**class** soba.space.continuousElements.**Wall**(*block1*, *block2*, *block3*, *color=None*)

**Class that implements building walls.**

**Attributes:**

**block1, block2, block3: lists of positions that contain positions between which an** occupant can move obeying with the impenetrability of the wall.

color: Color with which the object will be represented in the visualization.

### 4.3.2 Items in simplified modeling

**In the file continuousItems.py three classes are defined to implement the elements** of the physical space in a simplified model based on a room distribution:

-Room: Class that implements the rooms through which the Agent/Ocupant objects are located, move and where activities are carried out. -Door: Class that implements bulding plane doors. -Wall: Class that implements building walls.

**class** soba.space.roomsElements.**Door**(*room1=False*, *room2=False*, *state=False*)

**Class that implements bulding plane doors.**

**Attributes:** state: Door status, open (True) or closed (False). room1: First room to croos the door. room2: Second room to croos the door.

**Methods:** open: Change the status of the door to open. close: Change the status of the door to close.

**close**()
Change the status of the door to closed (False)

**open**()
Change the status of the door to open (True)

**class** soba.space.roomsElements.**Room**(*name*, *conectedTo*, *dx*, *dy*, *pos=(0, 0)*)

**Class that implements the rooms through which the Agent/Ocupant objects are located, move and where activities are car**

**Attributes:** name: Unique name of the room. roomsConected: List of accessible rooms from this room. dx: Size in the ordinate x (meters). dy: Size in the ordinate y (meters). pos: Position of the room (x, y). agentsInRoom: List of agent objects in the room walls: List of Wall objects of the room. doors: List of Doors objects of the room.

**class** soba.space.roomsElements.**Wall**(*room1=False*, *room2=False*)

**Class that implements building walls.**

**Attributes:** room1: First room to croos the door. room2: Second room to croos the door.

## 4.4 Launchers Module Documentation

### 4.4.1 Visual

soba.launchers.visual.**run**(*model*, *parameters*, *visual*, *back=False*)

**Execute the simulation with visual representation.**

**Args:** parameters: Parameters associated with the simulation model and others such as grid size. model: Model that is simulated. visual: JS files with the visualization elements that are included in the JavaScript browser visualization template. back: Python file working as backend visualization.

CHAPTER 5

# REST API

## 5.1 REST API

SOBA provides an API defined as a REST service (Get, Post, Pull and Push) to interact with the simulation. Specifically, the following methods are defined.

This API is supported by default at http://127.0.0.1:10000

| HTTP Method | URI | Action |
|---|---|---|
| GET | /api/v1/occupants | List with all occupants |
| GET | /api/v1/occupants/movements | Movement of all occupants |
| GET | /api/v1/occupants/positions | Position of all occupants |
| GET | /api/v1/occupants/states | States of all occupants |
| GET | /api/v1/occupants/{id} | Information about one occupant |
| GET | /api/v1/occupants/{id}/movement | Movement of one occupant |
| GET | /api/v1/occupants/{id}/position | Position of one occupant |
| GET | /api/v1/occupants/{id}/state | State of one occupant |
| GET | /api/v1/occupants/{id}/fov | FOV of one occupant |
| PUT | /api/v1/occupants/{id} | Create an occupant |
| POST | /api/v1/occupants/{id}/position | Move an occupant |

**GET /api/v1/occupants**

Return a list with all the occupants in the simulations.

Result:

```
{
  "occupants":
    [
      unique_id1, unique_id2, ..., unique_idN
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

Example:

```
{
  "occupants":
    [
      100001, 1, 0, 3, 2
    ]
}
```

---

**GET /api/v1/occupants/movements**

Return information about the movement all occupants are performing.

Result:

```
{
  "unique_id1":
    {
      "orientation": "orientation1",
      "speed": speed1
    },
  "unique_id2":
    {
      "orientation": "orientation2",
      "speed": speed2
    }
}
```

*Double speed: Speed of the occupants in meters per second.*

*String orientation: Orientation of movement as a cardinal point.*

Example:

```
{
  "1":
    {
      "orientation": "E",
      "speed": 0.71428
    },
  "0":
    {
      "orientation": "W",
      "speed": 0.71428
    },
  "3":
    {
      "orientation": "N",
      "speed": 0.71428
    },
  "2":
    {
      "orientation": "E",
```

(continues on next page)

(continued from previous page)

```
      "speed": 0.71428
    }
}
```

---

**GET /api/v1/occupants/positions**

Returns the position of all occupants on the grid x, y.

Result:

```
{
  "unique_id1":
    {
      "x": x1,
      "y": y1
    },
  "unique_id2":
    {
      "x": x2,
      "y": y2
    },
  ...
    ,
  "unique_idN":
    {
      "x": xN,
      "y": yN
    }
}
```

Example:

```
{
  "100001":
    {
      "x": 3,
      "y": 5
    },
  "1":
    {
      "x": 0,
      "y": 6
    },
  "0":
    {
      "x": 11,
      "y": 10
    },
  "3":
    {
      "x": 12,
      "y": 4
    },
  "2":
    {
```

(continues on next page)

(continued from previous page)

```
      "x": 7,
      "y": 11
    }
}
```

---

**GET /api/v1/occupants/states**

Returns the state or activity of all occupants.

Result:

```
{
  "unique_id1": "state1",
  unique_id2: "state2"
}
```

Example:

```
{
  "100001": "walking",
  "1": "Resting",
  "0": "Working in my laboratory",
  "3": "Working in my laboratory",
  "2": "Outside of building"
}
```

---

**GET /api/v1/occupants/{id}**

Returns general information (unique_id, state, FOV (field of vision), position and movement) of one occupant. The unique_id of the occupant must be provided.

Result:

```
{
  "occupant":
    {
      "movement":
        {
          "orientation": "orientation",
          "speed": speed
        },
      "unique_id": "unique_id",
      "position":
        {
          "x": x,
          "y": y
        },
              {
      "fov":
        [
          {
            "x": x1,
            "y": y1
```

(continues on next page)

(continued from previous page)

```
            },
            {
              "x": x2,
              "y": y2
            },
            {
              "x": x3,
              "y": y3
            },
            ...
            {
              "x": xN,
              "y": yN
            }
          ],
        "state": "state"
      }
}
```

*double unique_id: Unique identifier of an occupant.*

*string state: State or activity of an occupant.*

*double fov: Fielf of vision of an occupant.*

*double position: Position on the grid as (x, y) of an occupant.*

*double movement: Movement of an occupant.*

*double speed: Speed of the occupants in meters per second.*

*string orientation: Orientation of movement as a cardinal point.*

Example:

(continues on next page)

```
{
  "occupant":
    {
      "movement":
        {
          "orientation": "E",
          "speed": 0.71428
        },
      "unique_id": "1",
      "position":
        {
          "x": 0,
          "y": 6
        },
      "fov":
        [
          {"x": 5, "y": 0}, {"x": 6, "y": 0}, {"x": 7, "y": 0}, {"x
↪": 8, "y": 0}, {"x": 9, "y": 0}, {"x": 4, "y": 1}, {"x": 5, "y": 1}
↪, {"x": 6, "y": 1}, {"x": 7, "y": 1}, {"x": 8, "y": 1}, {"x": 9, "y
↪": 1}, {"x": 3, "y": 2}, {"x": 4, "y": 2}, {"x": 5, "y": 2}, {"x":␣
↪6, "y": 2}, {"x": 7, "y": 2}, {"x": 8, "y": 2}, {"x": 9, "y": 2}, {
↪"x": 2, "y": 3}, {"x": 3, "y": 3}, {"x": 4, "y": 3}, {"x": 5, "y":␣
↪3}, {"x": 6, "y": 3}, {"x": 7, "y": 3}, {"x": 8, "y": 3}, {"x": 9,
↪"y": 3}, {"x": 1, "y": 4}, {"x": 2, "y": 4}, {"x": 3, "y": 4}, {"x
↪": 4, "y": 4}, {"x": 5, "y": 4}, {"x": 6, "y": 4}, {"x": 7, "y": 4}
↪, {"x": 8, "y": 4}, {"x": 9, "y": 4}, {"x": 0, "y": 5}, {"x": 1, "y
↪": 5}, {"x": 2, "y": 5}, {"x": 3, "y": 5}, {"x": 4, "y": 5}, {"x":␣
↪5, "y": 5}, {"x": 6, "y": 5}, {"x": 7, "y": 5}, {"x": 8, "y": 5}, {
↪"x": 9, "y": 5}, {"x": 1, "y": 6}, {"x": 2, "y": 6}, {"x": 3, "y":␣
↪6}, {"x": 4, "y": 6}, {"x": 5, "y": 6}, {"x": 6, "y": 6}, {"x": 7,
↪"y": 6}, {"x": 8, "y": 6}, {"x": 9, "y": 6}, {"x": 0, "y": 7}, {"x
↪": 1, "y": 7}, {"x": 2, "y": 7}, {"x": 3, "y": 7}, {"x": 4, "y": 7}
```

(continued from previous page)

```
        ],
      "state": "Working in my laboratory"
    }
}
```

---

**GET /api/v1/occupants/{id}/movement**

Return information about the movement one occupant is performing. The unique_id of the occupant must be provided.

Results:

```
{
  "movement":
    {
      "orientation": "orientation",
      "speed": speed
    }
}
```

*Double speed: Speed of the occupants in meters per second.*

*String orientation: Orientation of movement as a cardinal point.*

Example:

```
{
  "movement":
    {
      "orientation": "E",
      "speed": 0.71428
    }
}
```

---

**GET /api/v1/occupants/{id}/position**

Returns the position of one occupant on the grid x, y. The unique_id of the occupant must be provided.

Result:

```
{
  "position" :
    {
      "x": x,
      "y": y
    }
}
```

Example:

```
{
  "position" :
    {
      "x": 4,
      "y": 7
```

(continues on next page)

(continued from previous page)

```
    }
}
```

### GET /api/v1/occupants/{id}/state

Returns the state or activity of one occupant. The unique_id of the occupant must be provided.

Result:

```
{"state": "state"}
```

Example:

```
{"state": "Working in my laboratory"}
```

### GET /api/v1/occupants/{id}/fov

Returns the position of the FOV (field of vision) of one occupant. The unique_id of the occupant must be provided.

Result:

```
{
  "fov":
    [
      {
        "x": x1,
        "y": y1
      },
      {
        "x": x2,
        "y": y2
      },
      {
        "x": x3,
        "y": y3
      },
      ...
      {
        "x": xN,
        "y": yN
      }
    ]
}
```

Example:

```
{
  "fov":
    [
      {"x": 5, "y": 0}, {"x": 6, "y": 0}, {"x": 7, "y": 0}, {"x": 8,
↪"y": 0}, {"x": 9, "y": 0}, {"x": 4, "y": 1}, {"x": 5, "y": 1}, {"x
↪": 6, "y": 1}, {"x": 7, "y": 1}, {"x": 8, "y": 1}, {"x": 9, "y": 1}
↪, {"x": 3, "y": 2}, {"x": 4, "y": 2}, {"x": 5, "y": 2}, {"x": 6, "y
↪": 2}, {"x": 7, "y": 2}, {"x": 8, "y": 2}, {"x": 9, "y": 2}, {"x":␣
↪2, "y": 3}, {"x": 3, "y": 3}, {"x": 4, "y": 3}, {"x": 5, "y": 3},␣
↪"x": 6, "y": 3}, {"x": 7, "y": 3}, {"x": 8, "y": 3}, {"x": 9, "y":␣
↪3}, {"x": 1, "y": 4}, {"x": 2, "y": 4}, {"x": 3, "y": 4}, {"x": 4,
↪"y": 4}, {"x": 5, "y": 4}, {"x": 6, "y": 4}, {"x": 7, "y": 4}, {"x
↪": 8, "y": 4}, {"x": 9, "y": 4}, {"x": 0, "y": 5}, {"x": 1, "y": 5}
↪, {"x": 2, "y": 5}, {"x": 3, "y": 5}, {"x": 4, "y": 5}, {"x": 5, "y
↪": 5}, {"x": 6, "y": 5}, {"x": 7, "y": 5}, {"x": 8, "y": 5}, {"x":␣
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

---

**PUT /api/v1/occupants/{id}**

Create an avatar object in a given position to be part of the simulation. The unique_id and the position (x, y) of the avatar must be provided.

Args:

```
{
  "x": x,
  "y": y
}
```

Results:

```
{
  "avatar":
    {
      "position":
        {
          "x": x,
          "y": y
        },
      "id": unique_id
    }
}
```

Example:

```
{
  "x": 4,
  "y": 5
}
```

```
{
  "avatar":
    {
      "position":
        {
          "x": 3,
          "y": 5
        },
      "id": 100010
    }
}
```

---

**POST /api/v1/occupants/{id}/position**

Move an avatar object to a given position. The unique_id and the new position (x, y) of the avatar must be provided.

Args:

```
{
  "x": x,
  "y": y
}
```

Result:

```
{
  "avatar":
    {
      "position":
        {
          "x": x,
          "y": y
        },
      "id": unique_id
    }
}
```

Example:

```
{
  "x": 4,
  "y": 5
}
```

```
{
  "avatar":
    {
      "position":
        {
          "x": 4,
          "y": 5
        },
      "id": 100010
    }
}
```

## 5.2 REST API Test

### 5.2.1 Test execution guide

To make a test of the REST service, the first thing to do is to make sure to run the SOBA example with the server option specified (-s), providing or not (optional) the port where the service will be defined.

```
$ git clone https://github.com/gsi-upm/soba

$ cd soba/projects/examples

$ python3 continuousExample.py -v -s
```

Then, once the software has run, following the previous case, selecting the start option of the interface in the browser. Once the simulation has started, run the 'apiTest.py' test script. If a port was defined in the execution, change the

variable port of the file to the chosen value.

```
$ python3 apiTest.py
```

### 5.2.2 Tests in a notebook

To execute the tests with greater control, it is recommended to download and use the jupyter notebook that is provided in the following address: Notebook

This notebook is also presented below.

#### Initialization of params

```
from unittest import TestCase
import json, requests
from jsonschema import validate
import socket
import unittest


ipServer = socket.gethostbyname(socket.gethostname())

#Port defined when executing, 10000 default
port = '10000'

#Template of the URLs
URLBASE = "http://127.0.0.1:"+ port
URISOBA = "/api/v1/occupants"

stringTemplate = {"type": "string"}
numberTemplate = {"type": "number"}

#Number of test for each pair (URI, Method)
N = 1
```

#### Tests

```
print(str('Testing {}').format('GET /api/v1/occupants'))
template = {
    "type": "object",
    "properties": {
        "occupants": {
            "type": "array"
            }
    },
    "required": ["occupants"]
}

for i in range(N):
    url = URLBASE + URISOBA
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
```

(continues on next page)

(continued from previous page)

```
    validate(datajson, template)
    for o in datajson["occupants"]:
        validate(o, numberTemplate)
```

```
Testing GET /api/v1/occupants
Response:  {'occupants': [1, 0, 3, 100000, 2]}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/movements'))
template = {
    "type": "object",
    "properties": {
        "orientation": {
            "type": "string"
            },
        "speed": {
            "type": "number"
            }
    },
    "required": ["orientation", "speed"]
}

template2 = {
    "type": "object"
}

for i in range(N):
    url = URLBASE + URISOBA + "/movements"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template2)
    for k, v  in datajson.items():
        validate(k, stringTemplate)
        validate(int(k), numberTemplate)
        validate(v, template)
```

```
Testing GET /api/v1/occupants/movements
Response:  {'0': {'speed': 1.38, 'orientation': 'SE'}, '1': {'speed': 1.38,
↪'orientation': 'W'}, '2': {'speed': 1.38, 'orientation': 'SE'}, '3': {'speed': 1.38,
↪ 'orientation': 'E'}}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/positions'))
template = {
    "type": "object",
    "properties": {
        "x": {
            "type": "number"
            },
        "y": {
            "type": "number"
            }
    },
    "required": ["x", "y"]
}
```

(continues on next page)

(continued from previous page)

```
for i in range(N):
    url = URLBASE + URISOBA + "/positions"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    for k, v in datajson.items():
        validate(k, stringTemplate)
        validate(int(k), numberTemplate)
        validate(v, template)
```

```
Testing GET /api/v1/occupants/positions
Response:  {'0': {'y': 8, 'x': 12}, '1': {'y': 6, 'x': 0}, '2': {'y': 8, 'x': 13},
↪'100000': {'y': 7, 'x': 5}, '3': {'y': 6, 'x': 14}}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/states'))
for i in range(N):
    url = URLBASE + URISOBA + "/states"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    for k,v in datajson.items():
        validate(v, stringTemplate)
        validate(k, stringTemplate)
        validate(int(k), numberTemplate)
```

```
Testing GET /api/v1/occupants/states
Response:  {'0': 'Working in my laboratory', '1': 'Working in my laboratory', '2':
↪'Working in my laboratory', '100000': 'walking', '3': 'Working in my laboratory'}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/{id}'))
template = {
    "type": "object",
    "properties": {
        "occupant":{
            "type": "object",
            "properties": {
                    "state":{
                        "type": "string"
                    },
                    "fov": {
                        "type": "array"
                    },
                    "unique_id":{
                        "type": "string"
                    },
                    "movement": {
                        "type": "object",
                        "properties": {
                            "orientation":{
                                "type": "string"
                            },
                            "speed":{
                                "type": "number"
                            },
                        },
```

(continues on next page)

(continued from previous page)

```
                        "required": ["orientation", "speed"]
                    },
                    "position": {
                        "type": "object",
                        "properties": {
                            "x":{
                                "type": "number"
                            },
                            "y":{
                                "type": "number"
                            }
                        },
                        "required": ["x", "y"]
                    }
            },
    "required": ["state", "fov", "unique_id", "movement", "position"]
        }
    },
    "required": ["occupant"]
}

template2 = {
    "type": "object",
    "properties": {
        "x": {
            "type": "number"
            },
        "y": {
            "type": "number"
        }
    },
    "required": ["x", "y"]
}

for i in range(N):
    url = URLBASE + URISOBA + "/" + str(0)
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
    validate(int(datajson['occupant']['unique_id']), numberTemplate)
    print(template)
    for p in datajson['occupant']['fov']:
        validate(p, template2)
```

```
Testing GET /api/v1/occupants/{id}
Response:  {'occupant': {'unique_id': '0', 'fov': [{'y': 0, 'x': 9}, {'y': 0, 'x': 10}
↪, {'y': 0, 'x': 11}, {'y': 0, 'x': 12}, {'y': 0, 'x': 13}, {'y': 0, 'x': 14}, {'y':␣
↪0, 'x': 15}, {'y': 0, 'x': 16}, {'y': 0, 'x': 17}, {'y': 0, 'x': 18}, {'y': 1, 'x':␣
↪9}, {'y': 1, 'x': 10}, {'y': 1, 'x': 11}, {'y': 1, 'x': 12}, {'y': 1, 'x': 13}, {'y
↪': 1, 'x': 14}, {'y': 1, 'x': 15}, {'y': 1, 'x': 16}, {'y': 1, 'x': 17}, {'y': 1, 'x
↪': 18}, {'y': 2, 'x': 9}, {'y': 2, 'x': 10}, {'y': 2, 'x': 11}, {'y': 2, 'x': 12}, {
↪'y': 2, 'x': 13}, {'y': 2, 'x': 14}, {'y': 2, 'x': 15}, {'y': 2, 'x': 16}, {'y': 2,
↪'x': 17}, {'y': 2, 'x': 18}, {'y': 3, 'x': 9}, {'y': 3, 'x': 10}, {'y': 3, 'x': 11},
↪ {'y': 3, 'x': 12}, {'y': 3, 'x': 13}, {'y': 3, 'x': 14}, {'y': 3, 'x': 15}, {'y':␣
↪3, 'x': 16}, {'y': 3, 'x': 17}, {'y': 3, 'x': 18}, {'y': 4, 'x': 9}, {'y': 4, 'x':␣
↪10}, {'y': 4, 'x': 11}, {'y': 4, 'x': 12}, {'y': 4, 'x': 13}, {'y': 4, 'x': 14}, {'y
↪': 4, 'x': 15}, {'y': 4, 'x': 16}, {'y': 4, 'x': 17}, {'y': 4, 'x': 18}, {'y': 4, 'x
↪': 19}, {'y': 5, 'x': 9}, {'y': 5, 'x': 10}, {'y': 5, 'x': 11}, {'y': 5, 'x': 12}, {
↪'y': 5, 'x': 13}, {'y': 5, 'x': 14}, {'y': 5, 'x': 15}, {'y': 5, 'x': 16}, {'y': 5,
↪'x': 17}, {'y': 5, 'x': 18}, {'y': 5, 'x': 19}, {'y': 6, 'x': 9}, {'y': 6, 'x': 10},
↪ {'y': 6, 'x': 11}, {'y': 6, 'x': 12}, {'y': 6, 'x': 13}, {'y': 6, 'x': 14}, {'y':␣
↪6, 'x': 15}, {'y': 6, 'x': 16}, {'y': 6, 'x': 17}, {'y': 6, 'x': 18}, {'y': 6, 'x':␣
↪19}, {'y': 7, 'x': 9}, {'y': 7, 'x': 10}, {'y': 7, 'x': 11}, {'y': 7, 'x': 12}, {'y
```

(continues on next page)

(continued from previous page)

```
{'type': 'object', 'required': ['occupant'], 'properties': {'occupant': {'type':
↪'object', 'required': ['state', 'fov', 'unique_id', 'movement', 'position'],
↪'properties': {'unique_id': {'type': 'string'}, 'fov': {'type': 'array'}, 'position
↪': {'type': 'object', 'required': ['x', 'y'], 'properties': {'y': {'type': 'number'}
↪, 'x': {'type': 'number'}}}, 'state': {'type': 'string'}, 'movement': {'type':
↪'object', 'required': ['orientation', 'speed'], 'properties': {'speed': {'type':
↪'number'}, 'orientation': {'type': 'string'}}}}}}}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/{id}/movement'))
template = {
    "type": "object",
    "properties": {
        "movement":{
            "type": "object",
            "properties": {
                    "orientation": {
                        "type": "string"
                    },
                    "speed": {
                        "type": "number"
                    }
            },
        "required": ["orientation", "speed"]
        }
    },
    "required": ["movement"]
}

for i in range(N):
    url = URLBASE + URISOBA + "/" + str(0) + "/movement"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
```

```
Testing GET /api/v1/occupants/{id}/movement
Response:  {'movement': {'speed': 1.38, 'orientation': 'SE'}}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/{id}/position'))
template = {
    "type": "object",
    "properties": {
        "position":{
            "type": "object",
            "properties": {
                "x": {
                    "type": "number"
                    },
                "y": {
                    "type": "number"
                }
            },
            "required": ["x", "y"]
        }
    },
```

(continues on next page)

(continued from previous page)

```
    "required": ["position"]
}

for i in range(N):
    url = URLBASE + URISOBA + "/" + str(0) + "/position"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
```

```
Testing GET /api/v1/occupants/{id}/position
Response:  {'position': {'y': 8, 'x': 12}}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/{id}/state'))
template = {
    "type": "object",
    "properties":{
        "state": {
            "type": "string"
        }
    },
    "required": ["state"]
}

for i in range(N):
    url = URLBASE + URISOBA + "/" + str(0) + "/state"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
```

```
Testing GET /api/v1/occupants/{id}/state
Response:  {'state': 'Working in my laboratory'}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/{id}/fov'))
template = {
    "type": "object",
    "properties": {
        "fov": {
            "type": "array"
            }
    },
    "required": ["fov"]
}


template2 = {
    "type": "object",
    "properties": {
        "x": {
            "type": "number"
            },
        "y": {
            "type": "number"
        }
```

(continues on next page)

(continued from previous page)

```
    },
    "required": ["x", "y"]
}

for i in range(N):
    url = URLBASE + URISOBA + "/" + str(0) + "/fov"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
    for p in datajson['fov']:
        validate(p, template2)
```

```
Testing GET /api/v1/occupants/{id}/fov
Response:  {'fov': [{'y': 0, 'x': 9}, {'y': 0, 'x': 10}, {'y': 0, 'x': 11}, {'y': 0,
↪'x': 12}, {'y': 0, 'x': 13}, {'y': 0, 'x': 14}, {'y': 0, 'x': 15}, {'y': 0, 'x': 16}
↪, {'y': 0, 'x': 17}, {'y': 0, 'x': 18}, {'y': 1, 'x': 9}, {'y': 1, 'x': 10}, {'y':␣
↪1, 'x': 11}, {'y': 1, 'x': 12}, {'y': 1, 'x': 13}, {'y': 1, 'x': 14}, {'y': 1, 'x':␣
↪15}, {'y': 1, 'x': 16}, {'y': 1, 'x': 17}, {'y': 1, 'x': 18}, {'y': 2, 'x': 9}, {'y
↪': 2, 'x': 10}, {'y': 2, 'x': 11}, {'y': 2, 'x': 12}, {'y': 2, 'x': 13}, {'y': 2, 'x
↪': 14}, {'y': 2, 'x': 15}, {'y': 2, 'x': 16}, {'y': 2, 'x': 17}, {'y': 2, 'x': 18},
↪{'y': 3, 'x': 9}, {'y': 3, 'x': 10}, {'y': 3, 'x': 11}, {'y': 3, 'x': 12}, {'y': 3,
↪'x': 13}, {'y': 3, 'x': 14}, {'y': 3, 'x': 15}, {'y': 3, 'x': 16}, {'y': 3, 'x': 17}
↪, {'y': 3, 'x': 18}, {'y': 4, 'x': 9}, {'y': 4, 'x': 10}, {'y': 4, 'x': 11}, {'y':␣
↪4, 'x': 12}, {'y': 4, 'x': 13}, {'y': 4, 'x': 14}, {'y': 4, 'x': 15}, {'y': 4, 'x':␣
↪16}, {'y': 4, 'x': 17}, {'y': 4, 'x': 18}, {'y': 4, 'x': 19}, {'y': 5, 'x': 9}, {'y
↪': 5, 'x': 10}, {'y': 5, 'x': 11}, {'y': 5, 'x': 12}, {'y': 5, 'x': 13}, {'y': 5, 'x
↪': 14}, {'y': 5, 'x': 15}, {'y': 5, 'x': 16}, {'y': 5, 'x': 17}, {'y': 5, 'x': 18},
↪{'y': 5, 'x': 19}, {'y': 6, 'x': 9}, {'y': 6, 'x': 10}, {'y': 6, 'x': 11}, {'y': 6,
↪'x': 12}, {'y': 6, 'x': 13}, {'y': 6, 'x': 14}, {'y': 6, 'x': 15}, {'y': 6, 'x': 16}
↪, {'y': 6, 'x': 17}, {'y': 6, 'x': 18}, {'y': 6, 'x': 19}, {'y': 7, 'x': 9}, {'y':␣
↪7, 'x': 10}, {'y': 7, 'x': 11}, {'y': 7, 'x': 12}, {'y': 7, 'x': 13}, {'y': 7, 'x':␣
↪14}, {'y': 7, 'x': 15}, {'y': 7, 'x': 16}, {'y': 7, 'x': 17}, {'y': 7, 'x': 18}, {'y
↪': 8, 'x': 9}, {'y': 8, 'x': 10}, {'y': 8, 'x': 11}, {'y': 8, 'x': 13}, {'y': 8, 'x
↪': 14}, {'y': 8, 'x': 15}, {'y': 8, 'x': 16}, {'y': 8, 'x': 17}, {'y': 8, 'x': 18},
↪{'y': 9, 'x': 9}, {'y': 9, 'x': 10}, {'y': 9, 'x': 11}, {'y': 9, 'x': 12}, {'y': 9,
↪'x': 13}, {'y': 9, 'x': 14}, {'y': 9, 'x': 15}, {'y': 9, 'x': 16}, {'y': 9, 'x': 17}
↪, {'y': 9, 'x': 18}, {'y': 10, 'x': 8}, {'y': 10, 'x': 9}, {'y': 10, 'x': 10}, {'y
↪': 10, 'x': 11}, {'y': 10, 'x': 12}, {'y': 10, 'x': 13}, {'y': 10, 'x': 14}, {'y':␣
↪10, 'x': 15}, {'y': 10, 'x': 16}, {'y': 10, 'x': 17}, {'y': 10, 'x': 18}, {'y': 11,
↪'x': 6}, {'y': 11, 'x': 7}, {'y': 11, 'x': 8}, {'y': 11, 'x': 9}, {'y': 11, 'x': 10}
↪, {'y': 11, 'x': 11}, {'y': 12, 'x': 4}, {'y': 12, 'x': 5}, {'y': 12, 'x': 6}, {'y
↪': 12, 'x': 7}, {'y': 12, 'x': 8}, {'y': 12, 'x': 9}, {'y': 12, 'x': 10}, {'y': 12,
↪'x': 11}, {'y': 13, 'x': 3}, {'y': 13, 'x': 4}, {'y': 13, 'x': 5}, {'y': 13, 'x': 6}
↪, {'y': 13, 'x': 7}, {'y': 13, 'x': 8}, {'y': 13, 'x': 9}, {'y': 13, 'x': 10}, {'y
↪': 13, 'x': 11}, {'y': 14, 'x': 1}, {'y': 14, 'x': 2}, {'y': 14, 'x': 3}, {'y': 14,
↪'x': 4}, {'y': 14, 'x': 5}, {'y': 14, 'x': 6}, {'y': 14, 'x': 7}, {'y': 14, 'x': 8},
↪ {'y': 14, 'x': 9}, {'y': 14, 'x': 10}, {'y': 15, 'x': 0}, {'y': 15, 'x': 1}, {'y':␣
↪15, 'x': 2}, {'y': 15, 'x': 3}, {'y': 15, 'x': 4}, {'y': 15, 'x': 5}, {'y': 15, 'x
↪': 6}, {'y': 15, 'x': 7}, {'y': 15, 'x': 8}, {'y': 15, 'x': 9}, {'y': 15, 'x': 10},
↪{'y': 16, 'x': 0}, {'y': 16, 'x': 1}, {'y': 16, 'x': 2}, {'y': 16, 'x': 3}, {'y':␣
↪16, 'x': 4}, {'y': 16, 'x': 5}, {'y': 16, 'x': 6}, {'y': 16, 'x': 7}, {'y': 16, 'x
↪': 8}, {'y': 16, 'x': 9}, {'y': 16, 'x': 10}, {'y': 17, 'x': 0}, {'y': 17, 'x': 1},
↪{'y': 17, 'x': 2}, {'y': 17, 'x': 3}, {'y': 17, 'x': 4}, {'y': 17, 'x': 5}, {'y':␣
↪17, 'x': 6}, {'y': 17, 'x': 7}, {'y': 17, 'x': 8}, {'y': 17, 'x': 9}, {'y': 18, 'x
↪': 0}, {'y': 18, 'x': 1}, {'y': 18, 'x': 2}, {'y': 18, 'x': 3}, {'y': 18, 'x': 4}, {
↪'y': 18, 'x': 5}, {'y': 18, 'x': 6}, {'y': 18, 'x': 7}, {'y': 18, 'x': 8}, {'y': 18,
↪ 'x': 9}]}
```

(continues on next page)

(continued from previous page)

```
print(str('Testing {}').format('PUT /api/v1/occupants/{id}'))
template = {
    "type": "object",
        "properties": {
            "avatar":{
                "type": "object",
                "properties": {
                    "position":{
                        "type": "object",
                        "properties": {
                            "x": {
                                "type": "number",
                            },
                            "y": {
                                "type": "number"
                            }
                        },
                        "required": ["x", "y"]
                    },
                    "id":{
                        "type": "number"
                    }
            },
            "required": ["position", "id"]
        }
    },
    "required": ["avatar"]
}

dataBody = {"x": 10, "y": 10}

for i in range(N):
    url = URLBASE + URISOBA + "/" + str(0)
    data = requests.put(url, json=dataBody, headers={'Content-Type': "application/json
↪", 'Accept': "application/json"})
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
```

```
Testing PUT /api/v1/occupants/{id}
Response:  {'avatar': {'position': {'y': 10, 'x': 10}, 'id': 100000}}
```

```
print(str('Testing {}').format('POST /api/v1/occupants/{id}/position'))
template = {
    "type": "object",
        "properties": {
            "avatar":{
                "type": "object",
                "properties": {
                    "position":{
                        "type": "object",
                        "properties": {
                            "x": {
                                "type": "number",
```

(continues on next page)

(continued from previous page)

```
                                },
                                "y": {
                                    "type": "number"
                                }
                            },
                            "required": ["x", "y"]
                        },
                        "id":{
                            "type": "number"
                        }
                },
                "required": ["position", "id"]
            }
        },
        "required": ["avatar"]
}

dataBody = {"x": 5, "y": 7}

for i in range(N):
    url = URLBASE + URISOBA + "/" + str(100000) + "/position"
    data = requests.post(url, json=dataBody, headers={'Content-Type': "application/
↪json", 'Accept': "application/json"})
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
```

```
Testing POST /api/v1/occupants/{id}/position
Response:  {'avatar': {'position': {'y': 7, 'x': 5}, 'id': 100000}}
```

CHAPTER 6

# SEBA Documentation

## 6.1 SEBA Overview

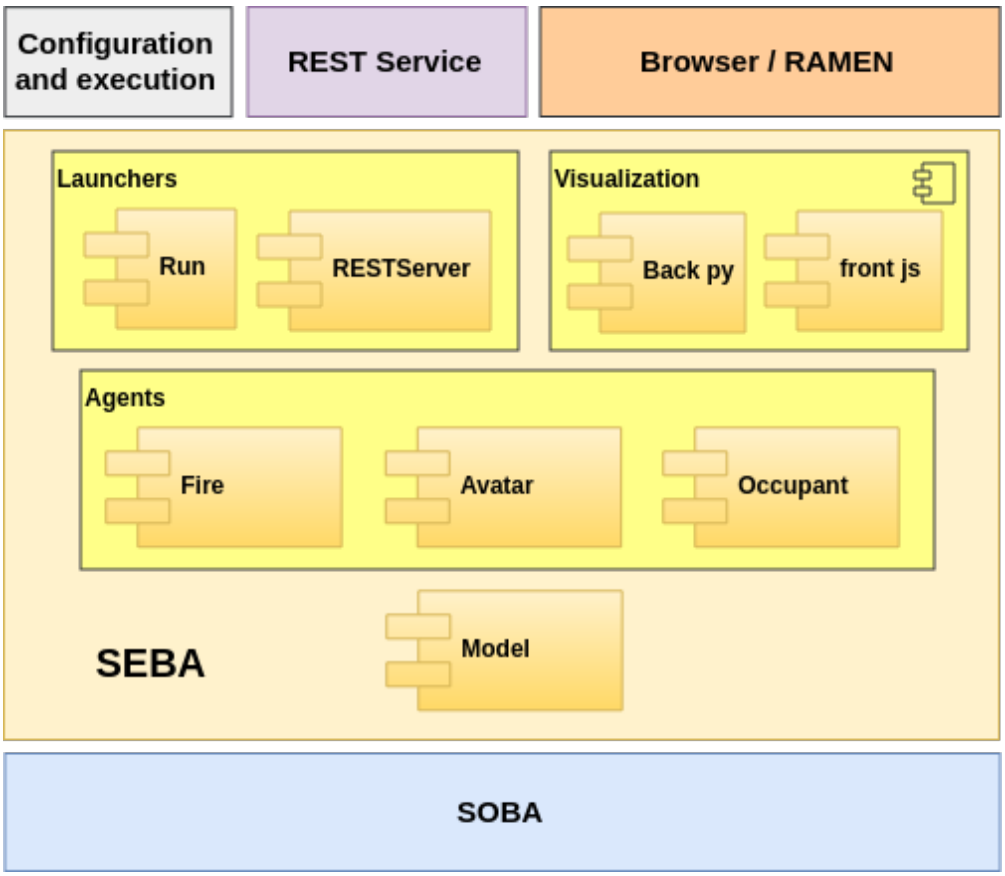
SEBA (Simulation of Evacuations Based on sobA) is a useful simulation tool for studies related to emergencies and evacuations in buildings. It is implemented in Python on SOBA software.

Simulations can be defined in three different ways. First, using behavioral modeling and strategies already implemented, by defining the space and specific characteristics of the occupants. Second, through inheritance and modification of the models defined in the classes already implemented. Third, by modifying the current software implementation.

It is provided as open source software:

`Github repository`: https://github.com/gsi-upm/soba/tree/master/projects/seba

### 6.1.1 Arquitecture Description




#### SEBA Components

- SOBA. Software tool base for the implementation of SEBA.
- RAMEN. It is an agent-based social simulation visualization tool for indoor crowd analytics based on the library Three.js. It allows to visualize a social simulation in a 3D environment and also to create the floor plan of a building.
- Browser. Using a browser a simple visualization can be made to know the performance of the simulation. This is also useful for debugging.
- REST Service. The software provide an API defined as a REST service (Get, Post, Pull and Push methods are defined) to interact with the simulation.

#### SEBA Modules

SEBA is implemented through 4 modules with classes with related functionalities.

- **Model**. This component is the center of the simulations. In the model class, the simulation starts, creating the variables and defining the conditions. During execution, it manages and functions as an intermediary between instances. In this class the strategies of the models are specified.
- ***Agents*.**

  - **Occupant**. An object of the Occupant class is a type of agent developed and characterized to simulate the behavior of crowds in buildings. The occupants are agents with their activity defined by markov states.
  - **Avatar**. It enables to create avatars that represent virtual occupants, that is, they are not controlled by the simulation but by an API Rest, providing a means of interaction between simulation and real human participation.
  - **Fire**. Through this component the threat of the emergency is modeled, specifically a fire that spreads through the building.

- ***Visualization*.**
  - **Back.py** and **front js**. Two components provide a simple mechanism to represent the model in a web interface, based on HTML rendering though a server interface, implemented with web sockets. Connection between a JS file and a class py by means of parameters rendering.

- ***Launchers*.**
  - **RESTServer**. Specification of the REST service server deployment.
  - **Run**. Provides the execution of the simulation from terminal.

## 6.2 How install and run

### 6.2.1 SOBA package instalation

First of all, it is necessary to have the SOBA package installed. For this, we execute the following command.

```
$ pip install soba
```

In case of error, this other command should be used, ensuring to have installed python 3 version and pip 3 version.

```
$ pip3 install soba
```

### 6.2.2 SEBA Github repository

To use the SEBA software, we must first download the repository of the github SOBA project and access its directory.

```
$ git clone https://github.com/gsi-upm/soba

$ cd soba/projects/seba
```

Then, execute the run file.

```
$ python run.py
```

or

```
$ python3 run.py
```

Different options are provided for execution:

1. Visual mode

```
$ python3 run.py -v
```

1.1 Launching REST Server

```
$ python3 run.py -v -s
```

1.2 Using RAMEN tool

```
$ python3 run.py -v -r
```

2. Batch mode

```
$ python3 run.py -b
```

2.1 Launching REST Server

```
$ python3 run.py -b -s
```

2.2 Using RAMEN tool

```
$ python3 run.py -b -r
```

## 6.3 Use Case

### 6.3.1 Instalation

If the SOBA package is not yet installed, we must first do so. To install SOBA the best option is to use the package management system PIP. For this, we execute the following command.

```
$ pip install soba
```

In case of error, this other command should be used, ensuring to have installed python 3 and pip 3.

```
$ pip3 install soba
```

### 6.3.2 Tutorial

SEBA enables the performance of the simulations in two modes:

1. With visual representation.
2. In batch mode.

In the tutorials, the small modifications required to use each posibility are reflected.

In addition, two added mechanisms are provided to interact with the simulation:

1. Use an API on a REST server to obtain information and create and manage avatars.
2. use the external tool RAMEN for advanced 3D-visualization on Three.js.

### Configuring the simulation

First we define the generic parameters.

1.- We define the characteristics of the occupants

```
from soba.models.continuousModel import ContinuousModel
import soba.visualization.ramen.mapGenerator as ramen
import soba.run
from collections import OrderedDict
import json
import sys
from model import SEBAModel
from visualization.back import Visualization
import datetime as dt


strategies = ['nearest', 'safest', 'uncrowded']

today = dt.date.today()
timeHazard = dt.datetime(today.year, today.month, 1, 8, 30, 0, 0)

families = []

family1 = {'N': 3, 'child': 1, 'adult': 2} #Only two are really neccesary
family2 = {'N': 3, 'child': 2, 'adult': 1}

# Uncomment to consider families
#families.append(family1)
#families.append(family2)

sebaConfiguration = {'families': families, 'hazard': timeHazard}

#JSON to store all the informacion.
jsonsOccupants = []

#Number of occupants
N = 4

#Definition of the states
states = OrderedDict([('Leaving','out'), ('Resting', 'sofa'), ('Working in my␣
↪laboratory', 'wp')])

#Definition of the schedule
schedule = {'t1': "08:01:00", 't2': "08:10:00", 't3': "08:20:00"}

#Possible Variation on the schedule
variation = {'t1': "00:01:00", 't2': "00:01:00", 't3': "00:01:00"}

#Probability of state change associated with the Markovian chain as a function of the␣
↪temporal period
markovActivity = {
    '-t1': [[100, 0, 0], [0, 0, 0], [0, 0, 0]],
    't1-t2': [[0, 0, 100], [0, 50, 50], [0, 50, 50]],
    't2-t3': [[100, 0, 0], [0, 50, 50], [0, 50, 50]],
    't3-': [[0, 0, 100], [0, 100, 0], [0, 100, 0]]
}
```

(continues on next page)

(continued from previous page)

```
#Time associated to each state (minutes)
timeActivity = {
    '-t1': [3, 0, 0], 't1-t2': [3, 3, 3], 't2-t3': [3, 3, 3], 't3-': [3, 3, 3]
}

#Time variation associated to each state (minutes)
timeActivityVariation = {
    '-t1': [1, 0, 0], 't1-t2': [1, 1, 1], 't2-t3': [1, 1, 1], 't3-': [1, 1, 1]
}

#Store the information
jsonOccupant = {'type': 'example' , 'N': N, 'states': states , 'schedule': schedule,
↪'variation': variation,
'markovActivity': markovActivity, 'timeActivity': timeActivity, 'timeActivityVariation
↪': timeActivityVariation,
'strategy': 'nearest'}

cellW = 20
cellH = 20

jsonsOccupants.append(jsonOccupant)
```

2.- We define the building plan or the distribution of the space.

```
import soba.visualization.ramen.mapGenerator as ramen

with open('auxiliarFiles/labgsi.blueprint3d') as data_file:
    jsonMap = ramen.returnMap(data_file)
```

3.- We call the execution methods.

```
3.1-With visual representation.
```

```
sys.argv = []
sys.argv.append("-l")
sys.argv.append("-v")

back = Visualization(cellW, cellH)
parameters = {'width': cellW, 'height': cellH, 'jsonMap': jsonMap, 'jsonsOccupants':␣
↪jsonsOccupants, 'sebaConfiguration': sebaConfiguration}
soba.run.run(SEBAModel, parameters, visualJS="visualization/front.js", back=back)
```

```
SOBA is running
{'t1': datetime.datetime(2017, 10, 1, 8, 1, 0, 291422), 't2': datetime.datetime(2017,␣
↪10, 1, 8, 9, 45, 947140), 't3': datetime.datetime(2017, 10, 1, 8, 20, 5, 813700)}
{'t1': datetime.datetime(2017, 10, 1, 8, 1, 5, 873230), 't2': datetime.datetime(2017,␣
↪10, 1, 8, 9, 40, 597876), 't3': datetime.datetime(2017, 10, 1, 8, 20, 18, 968479)}
{'t1': datetime.datetime(2017, 10, 1, 8, 0, 35, 656475), 't2': datetime.datetime(2017,
↪ 10, 1, 8, 9, 46, 200526), 't3': datetime.datetime(2017, 10, 1, 8, 20, 45, 775966)}
{'t1': datetime.datetime(2017, 10, 1, 8, 1, 12, 804713), 't2': datetime.datetime(2017,
↪ 10, 1, 8, 10, 12, 223587), 't3': datetime.datetime(2017, 10, 1, 8, 20, 8, 565048)}
Interface starting at http://127.0.0.1:7777
Socket opened!
{"type":"get_params"}
{"type":"reset"}
{'t1': datetime.datetime(2017, 10, 1, 8, 1, 0, 291422), 't2': datetime.datetime(2017,␣
↪10, 1, 8, 9, 45, 947140), 't3': datetime.datetime(2017, 10, 1, 8, 20, 5, 813700)}
```

(continues on next page)

(continued from previous page)

```
{'t1': datetime.datetime(2017, 10, 1, 8, 1, 5, 873230), 't2': datetime.datetime(2017,␣
↪10, 1, 8, 9, 40, 597876), 't3': datetime.datetime(2017, 10, 1, 8, 20, 18, 968479)}
{'t1': datetime.datetime(2017, 10, 1, 8, 0, 35, 656475), 't2': datetime.datetime(2017,
↪ 10, 1, 8, 9, 46, 200526), 't3': datetime.datetime(2017, 10, 1, 8, 20, 45, 775966)}
{'t1': datetime.datetime(2017, 10, 1, 8, 1, 12, 804713), 't2': datetime.datetime(2017,
↪ 10, 1, 8, 10, 12, 223587), 't3': datetime.datetime(2017, 10, 1, 8, 20, 8, 565048)}
{"type":"get_step","step":1}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:51:00
{"type":"get_step","step":2}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:52:00
{"type":"get_step","step":3}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:53:00
{"type":"get_step","step":4}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:54:00
{"type":"get_step","step":5}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:55:00
{"type":"get_step","step":6}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:56:00
{"type":"get_step","step":7}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:57:00
{"type":"get_step","step":8}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:58:00
{"type":"get_step","step":9}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:59:00
{"type":"get_step","step":10}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:00:00
{"type":"get_step","step":11}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:01:00
{"type":"get_step","step":12}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:02:00
{"type":"get_step","step":13}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:03:00
{"type":"get_step","step":14}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:04:00
{"type":"get_step","step":15}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:05:00
{"type":"get_step","step":16}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:06:00
{"type":"get_step","step":17}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:07:00
```

(continues on next page)

(continued from previous page)

```
{"type":"get_step","step":18}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:08:00
{"type":"get_step","step":19}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:09:00
{"type":"get_step","step":20}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:10:00
{"type":"get_step","step":21}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:11:00
{"type":"get_step","step":22}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:12:00
{"type":"get_step","step":23}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:13:00
{"type":"get_step","step":24}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:14:00
{"type":"get_step","step":25}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:15:00
{"type":"get_step","step":26}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:16:00
{"type":"get_step","step":27}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:17:00
{"type":"get_step","step":28}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:18:00
{"type":"get_step","step":29}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:19:00
{"type":"get_step","step":30}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:20:00
{"type":"get_step","step":31}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:21:00
{"type":"get_step","step":32}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:22:00
{"type":"get_step","step":33}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:23:00
{"type":"get_step","step":34}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:24:00
{"type":"get_step","step":35}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:25:00
{"type":"get_step","step":36}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:26:00
```

(continues on next page)

(continued from previous page)

```
{"type":"get_step","step":37}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:27:00
{"type":"get_step","step":38}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:28:00
{"type":"get_step","step":39}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:29:00
{"type":"get_step","step":40}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:30:00
{"type":"get_step","step":41}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:31:00
{"type":"get_step","step":42}
Situation:  Emergency , Occupants dead:  0 , Occupants alive:  4
01:08:32:00
{"type":"get_step","step":43}
Situation:  Emergency , Occupants dead:  0 , Occupants alive:  4
01:08:33:00
{"type":"get_step","step":44}
Situation:  Emergency , Occupants dead:  0 , Occupants alive:  4
01:08:34:00
{"type":"get_step","step":45}
Situation:  Emergency , Occupants dead:  0 , Occupants alive:  4
01:08:35:00
{"type":"get_step","step":46}
Situation:  Emergency , Occupants dead:  0 , Occupants alive:  4
01:08:36:00
{"type":"get_step","step":47}
Situation:  Emergency , Occupants dead:  0 , Occupants alive:  4
01:08:37:00
{"type":"get_step","step":48}
Situation:  Emergency , Occupants dead:  0 , Occupants alive:  4
01:08:38:00
{"type":"get_step","step":49}
Situation:  Emergency , Occupants dead:  1 , Occupants alive:  3
01:08:39:00
{"type":"get_step","step":50}
Situation:  Emergency , Occupants dead:  1 , Occupants alive:  3
01:08:40:00
{"type":"get_step","step":51}
Situation:  Emergency , Occupants dead:  2 , Occupants alive:  2
01:08:41:00
{"type":"get_step","step":52}
Situation:  Emergency , Occupants dead:  2 , Occupants alive:  2
01:08:42:00
{"type":"get_step","step":53}
Situation:  Emergency , Occupants dead:  2 , Occupants alive:  2
01:08:43:00
{"type":"get_step","step":54}
Situation:  Emergency , Occupants dead:  2 , Occupants alive:  2
Simulation terminated.
```

```
3.1- Bacth mode.
```

```
import soba.run
import sys

#Fixed parameters during iterations
fixed_params = {"width": cellW, "height": cellH, "jsonMap": jsonMap, "jsonsOccupants
↪": jsonsOccupants, 'sebaConfiguration': sebaConfiguration}

#Variable parameters to each iteration
variable_params = {"seed": range(10, 500, 10)}


sys.argv = []
sys.argv.append("-1")
sys.argv.append("-b")


soba.run.run(SEBAModel, fixed_params, variable_params)
```

```
0it [00:00, ?it/s]
```

```
SOBA is running
{'t1': datetime.datetime(2017, 10, 1, 8, 1, 26, 631730), 't2': datetime.datetime(2017,
↪ 10, 1, 8, 10, 14, 305579), 't3': datetime.datetime(2017, 10, 1, 8, 19, 29, 91994)}
{'t1': datetime.datetime(2017, 10, 1, 8, 0, 59, 832323), 't2': datetime.datetime(2017,
↪ 10, 1, 8, 10, 12, 426719), 't3': datetime.datetime(2017, 10, 1, 8, 19, 45, 598289)}
{'t1': datetime.datetime(2017, 10, 1, 8, 1, 5, 310232), 't2': datetime.datetime(2017,␣
↪10, 1, 8, 10, 2, 170971), 't3': datetime.datetime(2017, 10, 1, 8, 20, 0, 85829)}
{'t1': datetime.datetime(2017, 10, 1, 8, 0, 56, 507996), 't2': datetime.datetime(2017,
↪ 10, 1, 8, 10, 8, 660524), 't3': datetime.datetime(2017, 10, 1, 8, 20, 24, 60747)}
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:51:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:52:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:53:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:54:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:55:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:56:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:57:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:58:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:07:59:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:00:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:01:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:02:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:03:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:04:00
```

(continues on next page)

(continued from previous page)

```
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:05:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:06:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:07:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:08:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:09:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:10:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:11:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:12:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:13:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:14:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:15:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:16:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:17:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:18:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:19:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:20:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:21:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:22:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:23:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:24:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:25:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:26:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:27:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:28:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:29:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:30:00
Situation:  Normal , Occupants dead:  0 , Occupants alive:  4
01:08:31:00
Situation:  Emergency , Occupants dead:  0 , Occupants alive:  4
01:08:32:00
Situation:  Emergency , Occupants dead:  0 , Occupants alive:  4
```

(continues on next page)

(continued from previous page)

```
01:08:33:00
Situation:  Emergency , Occupants dead:  0 , Occupants alive:  4
01:08:34:00
Situation:  Emergency , Occupants dead:  0 , Occupants alive:  4
01:08:35:00
Situation:  Emergency , Occupants dead:  0 , Occupants alive:  4
01:08:36:00
Situation:  Emergency , Occupants dead:  0 , Occupants alive:  4
01:08:37:00
Situation:  Emergency , Occupants dead:  0 , Occupants alive:  4
01:08:38:00
Situation:  Emergency , Occupants dead:  0 , Occupants alive:  4
01:08:39:00
Situation:  Emergency , Occupants dead:  1 , Occupants alive:  3
01:08:40:00
Situation:  Emergency , Occupants dead:  1 , Occupants alive:  3
Simulation terminated.
```

## 6.4 APIs

### 6.4.1 Model Module Documentation

#### Model

### 6.4.2 Agents Module Documentation

#### Occupant

#### Fire

**class** projects.seba.fire.**Fire**(*model*, *pos*)

This class enables to create fire object on a position. The objects of this class are controlled by one FireControl object.

**Attributes:** grade: Intensity level of the fire. pos: Fire position.

**class** projects.seba.fire.**FireControl**(*unique_id*, *model*, *posInit*, *expansionRate=0.7*, *growthRate=0.7*)

This class enables to create agents that control the fire expansión, representing the emergency threat.

**Attributes:** fireExpansion: Set of Fire objects belonging to this FireControl. limitFire: Fire objects that are in the limit to make the expansion. expansionRate: Rate of expansion of the threat. growthRate: Value of growth in intensity of the fire.

**Methods:** createFirePos: Create a Fire object in a given position. getFirePos: Get a Fire object in a position given. expansionFire: Make the expansion of fire limits. growthFire: Make the growth in intensity of the fire. step: Method invoked by the Model scheduler in each step.

**createFirePos**(*pos*)

**Create a Fire object in a given position.**

**Args:** pos: Position to put the Fire object as (x, y)

**expansionFire**()
Make the expansion of fire limits.

**getFirePos**(*pos*)

**Get a Fire object in a position given.**

**Args:** pos: Position to be checked.

Return: Fire object or False

**growthFire**()
Make the growth in intensity of the fire.

**step**()
Method invoked by the Model scheduler in each step.

### Avatar

**class** projects.seba.avatar.**EmergencyAvatar**(*unique_id*, *model*, *initial_pos*, *color='red'*, *initial_state='walking'*)
This class enables to create avatars that represent virtual occupants, that is, they are not controlled by the simulation but by an API Rest. This class inherits from the avatar class of SOBA.

**Attributes:** Those Inherited from the Avatar class of SOBA. alive: Current state of an avatar, live or not. life: Number of remaining life points of the avatar.

**Methods:** getExitGate: Obtain the optimal way to evacuate the building according to an evacuation strategy. getPosFireFOV: Obtain the positions in the avatar's field of vision where there is fire. makeEmergencyAction: Method that is invoked when initiating an emergency to make the decision of response.

**getExitGate**()

**Obtain the optimal way to evacuate the building according to an evacuation strategy.** Return: List of positions (x, y)

**getPosFireFOV**()

**Check if the position is in my field of vision** Return: List of positions (x, y)

**makeEmergencyAction**()
Method that is invoked when initiating an emergency to make the decision of response.

**step**()
Method invoked by the Model scheduler in each step. Step common to all occupants.

## 6.5 REST API

### 6.5.1 REST API Definition

SEBA provides an API defined as a REST service (Get, Post, Pull and Push) to interact with the simulation. Specifically, the following methods are defined.

This API is supported by default at http://127.0.0.1:10000

| HTTP Method | URI | Action |
|---|---|---|
| GET | /api/v1/occupants | List with all occupants |
| GET | /api/v1/occupants/movements | Movement of all occupants |
| GET | /api/v1/occupants/positions | Position of all occupants |
| GET | /api/v1/occupants/states | States of all occupants |
| GET | /api/v1/occupants/{id} | Information about one occupant |
| GET | /api/v1/occupants/{id}/movement | Movement of one occupant |
| GET | /api/v1/occupants/{id}/position | Position of one occupant |
| GET | /api/v1/occupants/{id}/state | State of one occupant |
| GET | /api/v1/occupants/{id}/fov | FOV of one occupant |
| GET | /api/v1/occupants/{id}/route/{route_id} | Evacuation route of one occupant |
| GET | /api/v1/occupants/{id}/fire | Fire in the one occupant's FOV |
| PUT | /api/v1/occupants/{id} | Create an emergency occupant |
| POST | /api/v1/occupants/{id}/position | Move an occupant |
| GET | /api/v1/fire | Positions with fire |

**GET /api/v1/occupants**

Return a list with all the occupants in the simulations.

Result:

```
{
  "occupants":
    [
      unique_id1, unique_id2, ..., unique_idN
    ]
}
```

Example:

```
{
  "occupants":
    [
      100001, 1, 0, 3, 2
    ]
}
```

**GET /api/v1/occupants/movements**

Return information about the movement all occupants are performing.

Result:

```
{
  "unique_id1":
    {
      "orientation": "orientation1",
      "speed": speed1
    },
  "unique_id2":
    {
      "orientation": "orientation2",
```

(continues on next page)

(continued from previous page)

```
      "speed": speed2
    }
}
```

*Double speed: Speed of the occupants in meters per second.*

*String orientation: Orientation of movement as a cardinal point.*

Example:

```
{
  "1":
    {
      "orientation": "E",
      "speed": 0.71428
    },
  "0":
    {
      "orientation": "W",
      "speed": 0.71428
    },
  "3":
    {
      "orientation": "N",
      "speed": 0.71428
    },
  "2":
    {
      "orientation": "E",
      "speed": 0.71428
    }
}
```

---

**GET /api/v1/occupants/positions**

Returns the position of all occupants on the grid x, y.

Result:

```
{
  "unique_id1":
    {
      "x": x1,
      "y": y1
    },
  "unique_id2":
    {
      "x": x2,
      "y": y2
    },
  ...
    ,
  "unique_idN":
    {
      "x": xN,
      "y": yN
```

(continues on next page)

(continued from previous page)

```
    }
}
```

Example:

```
{
  "100001":
    {
      "x": 3,
      "y": 5
    },
  "1":
    {
      "x": 0,
      "y": 6
    },
  "0":
    {
      "x": 11,
      "y": 10
    },
  "3":
    {
      "x": 12,
      "y": 4
    },
  "2":
    {
      "x": 7,
      "y": 11
    }
}
```

---

**GET /api/v1/occupants/states**

Returns the state or activity of all occupants.

Result:

```
{
  "unique_id1": "state1",
  unique_id2: "state2"
}
```

Example:

```
{
  "100001": "walking",
  "1": "Resting",
  "0": "Working in my laboratory",
  "3": "Working in my laboratory",
  "2": "Outside of building"
}
```

---

**GET /api/v1/occupants/{id}**

Returns general information (unique_id, state, FOV (field of vision), position and movement) of one occupant. The unique_id of the occupant must be provided.

Result:

```
{
  "occupant":
    {
      "movement":
        {
          "orientation": "orientation",
          "speed": speed
        },
      "unique_id": "unique_id",
      "position":
        {
          "x": x,
          "y": y
        },
                {
      "fov":
        [
          {
            "x": x1,
            "y": y1
          },
          {
            "x": x2,
            "y": y2
          },
          {
            "x": x3,
            "y": y3
          },
          ...
          {
            "x": xN,
            "y": yN
          }
        ],
      "state": "state"
    }
}
```

*double unique_id: Unique identifier of an occupant.*

*string state: State or activity of an occupant.*

*double fov: Fielf of vision of an occupant.*

*double position: Position on the grid as (x, y) of an occupant.*

*double movement: Movement of an occupant.*

*double speed: Speed of the occupants in meters per second.*

*string orientation: Orientation of movement as a cardinal point.*

Example:

```
{
  "occupant":
    {
      "movement":
        {
          "orientation": "E",
          "speed": 0.71428
        },
      "unique_id": "1",
      "position":
        {
          "x": 0,
          "y": 6
        },
      "fov":
        [
          {"x": 5, "y": 0}, {"x": 6, "y": 0}, {"x": 7, "y": 0}, {"x
↪": 8, "y": 0}, {"x": 9, "y": 0}, {"x": 4, "y": 1}, {"x": 5, "y": 1}
↪, {"x": 6, "y": 1}, {"x": 7, "y": 1}, {"x": 8, "y": 1}, {"x": 9, "y
↪": 1}, {"x": 3, "y": 2}, {"x": 4, "y": 2}, {"x": 5, "y": 2}, {"x":␣
↪6, "y": 2}, {"x": 7, "y": 2}, {"x": 8, "y": 2}, {"x": 9, "y": 2}, {
↪"x": 2, "y": 3}, {"x": 3, "y": 3}, {"x": 4, "y": 3}, {"x": 5, "y":␣
↪3}, {"x": 6, "y": 3}, {"x": 7, "y": 3}, {"x": 8, "y": 3}, {"x": 9,
↪"y": 3}, {"x": 1, "y": 4}, {"x": 2, "y": 4}, {"x": 3, "y": 4}, {"x
↪": 4, "y": 4}, {"x": 5, "y": 4}, {"x": 6, "y": 4}, {"x": 7, "y": 4}
↪, {"x": 8, "y": 4}, {"x": 9, "y": 4}, {"x": 0, "y": 5}, {"x": 1, "y
↪": 5}, {"x": 2, "y": 5}, {"x": 3, "y": 5}, {"x": 4, "y": 5}, {"x":␣
↪5, "y": 5}, {"x": 6, "y": 5}, {"x": 7, "y": 5}, {"x": 8, "y": 5}, {
↪"x": 9, "y": 5}, {"x": 1, "y": 6}, {"x": 2, "y": 6}, {"x": 3, "y":␣
↪6}, {"x": 4, "y": 6}, {"x": 5, "y": 6}, {"x": 6, "y": 6}, {"x": 7,
↪"y": 6}, {"x": 8, "y": 6}, {"x": 9, "y": 6}, {"x": 0, "y": 7}, {"x
↪": 1, "y": 7}, {"x": 2, "y": 7}, {"x": 3, "y": 7}, {"x": 4, "y": 7}
↪, {"x": 5, "y": 7}, {"x": 6, "y": 7}, {"x": 7, "y": 7}, {"x": 8, "y
↪": 7}, {"x": 9, "y": 7}, {"x": 0, "y": 8}, {"x": 1, "y": 8}, {"x":␣
↪2, "y": 8}, {"x": 3, "y": 8}, {"x": 4, "y": 8}, {"x": 5, "y": 8}, {
↪"x": 6, "y": 8}, {"x": 7, "y": 8}, {"x": 8, "y": 8}, {"x": 9, "y":␣
↪8}, {"x": 1, "y": 9}, {"x": 2, "y": 9}, {"x": 3, "y": 9}, {"x": 4,
↪"y": 9}, {"x": 5, "y": 9}, {"x": 6, "y": 9}, {"x": 7, "y": 9}, {"x
↪": 8, "y": 9}, {"x": 9, "y": 9}, {"x": 1, "y": 10}, {"x": 2, "y":␣
↪10}, {"x": 3, "y": 10}, {"x": 4, "y": 10}, {"x": 5, "y": 10}, {"x
↪": 6, "y": 10}, {"x": 7, "y": 10}, {"x": 8, "y": 10}, {"x": 9, "y
↪": 10}, {"x": 10, "y": 10}, {"x": 11, "y": 10}, {"x": 10, "y": 11},
↪ {"x": 11, "y": 11}, {"x": 12, "y": 11}, {"x": 13, "y": 11}, {"x":␣
↪12, "y": 12}, {"x": 13, "y": 12}, {"x": 14, "y": 12}, {"x": 15, "y
↪": 12}, {"x": 16, "y": 12}, {"x": 14, "y": 13}, {"x": 15, "y": 13},
↪ {"x": 16, "y": 13}, {"x": 17, "y": 13}, {"x": 18, "y": 13}, {"x":␣
↪16, "y": 14}, {"x": 17, "y": 14}, {"x": 18, "y": 14}, {"x": 18, "y
↪": 15}
        ],
      "state": "Working in my laboratory"
    }
}
```


**GET /api/v1/occupants/{id}/movement**

Return information about the movement one occupant is performing. The unique_id of the occupant must be provided.

Results:

```
{
  "movement":
    {
      "orientation": "orientation",
      "speed": speed
    }
}
```

*Double speed: Speed of the occupants in meters per second.*

*String orientation: Orientation of movement as a cardinal point.*

Example:

```
{
  "movement":
    {
      "orientation": "E",
      "speed": 0.71428
    }
}
```

---

**GET /api/v1/occupants/{id}/position**

Returns the position of one occupant on the grid x, y. The unique_id of the occupant must be provided.

Result:

```
{
  "position" :
    {
      "x": x,
      "y": y
    }
}
```

Example:

```
{
  "position" :
    {
      "x": 4,
      "y": 7
    }
}
```

---

**GET /api/v1/occupants/{id}/state**

Returns the state or activity of one occupant. The unique_id of the occupant must be provided.

Result:

```
{"state": "state"}
```

Example:

```
{"state": "Working in my laboratory"}
```

### GET /api/v1/occupants/{id}/fov

Returns the position of the FOV (field of vision) of one occupant. The unique_id of the occupant must be provided.

Result:

```
{
  "fov":
    [
      {
        "x": x1,
        "y": y1
      },
      {
        "x": x2,
        "y": y2
      },
      {
        "x": x3,
        "y": y3
      },
      ...
      {
        "x": xN,
        "y": yN
      }
    ]
}
```

Example:

```
{
  "fov":
    [
      {"x": 5, "y": 0}, {"x": 6, "y": 0}, {"x": 7, "y": 0}, {"x": 8,
↪"y": 0}, {"x": 9, "y": 0}, {"x": 4, "y": 1}, {"x": 5, "y": 1}, {"x
↪": 6, "y": 1}, {"x": 7, "y": 1}, {"x": 8, "y": 1}, {"x": 9, "y": 1}
↪, {"x": 3, "y": 2}, {"x": 4, "y": 2}, {"x": 5, "y": 2}, {"x": 6, "y
↪": 2}, {"x": 7, "y": 2}, {"x": 8, "y": 2}, {"x": 9, "y": 2}, {"x":␣
↪2, "y": 3}, {"x": 3, "y": 3}, {"x": 4, "y": 3}, {"x": 5, "y": 3}, {
↪"x": 6, "y": 3}, {"x": 7, "y": 3}, {"x": 8, "y": 3}, {"x": 9, "y":␣
↪3}, {"x": 1, "y": 4}, {"x": 2, "y": 4}, {"x": 3, "y": 4}, {"x": 4,
↪"y": 4}, {"x": 5, "y": 4}, {"x": 6, "y": 4}, {"x": 7, "y": 4}, {"x
↪": 8, "y": 4}, {"x": 9, "y": 4}, {"x": 0, "y": 5}, {"x": 1, "y": 5}
↪, {"x": 2, "y": 5}, {"x": 3, "y": 5}, {"x": 4, "y": 5}, {"x": 5, "y
↪": 5}, {"x": 6, "y": 5}, {"x": 7, "y": 5}, {"x": 8, "y": 5}, {"x":␣
↪9, "y": 5}, {"x": 1, "y": 6}, {"x": 2, "y": 6}, {"x": 3, "y": 6}, {
↪"x": 4, "y": 6}, {"x": 5, "y": 6}, {"x": 6, "y": 6}, {"x": 7, "y":␣
↪6}, {"x": 8, "y": 6}, {"x": 9, "y": 6}, {"x": 0, "y": 7}, {"x": 1,
↪"y": 7}, {"x": 2, "y": 7}, {"x": 3, "y": 7}, {"x": 4, "y": 7}, {"x
↪": 5, "y": 7}, {"x": 6, "y": 7}, {"x": 7, "y": 7}, {"x": 8, "y": 7}
↪, {"x": 9, "y": 7}, {"x": 0, "y": 8}, {"x": 1, "y": 8}, {"x": 2, "y
↪": 8}, {"x": 3, "y": 8}, {"x": 4, "y": 8}, {"x": 5, "y": 8}, {"x":␣
↪6, "y": 8}, {"x": 7, "y": 8}, {"x": 8, "y": 8}, {"x": 9, "y": 8}, {
↪"x": 1, "y": 9}, {"x": 2, "y": 9}, {"x": 3, "y": 9}, {"x": 4, "y":␣
↪9}, {"x": 5, "y": 9}, {"x": 6, "y": 9}, {"x": 7, "y": 9}, {"x": 8,
↪"y": 9}, {"x": 9, "y": 9}, {"x": 1, "y": 10}, {"x": 2, "y": 10}, {
↪"x": 3, "y": 10}, {"x": 4, "y": 10}, {"x": 5, "y": 10}, {"x": 6, "y
↪": 10}, {"x": 7, "y": 10}, {"x": 8, "y": 10}, {"x": 9, "y": 10}, {
↪"x": 10, "y": 10}, {"x": 11, "y": 10}, {"x": 10, "y": 11}, {"x":␣
```

(continues on next page)

(continued from previous page)

```
    ]
}
```

---

**PUT /api/v1/occupants/{id}**

Create an avatar object in a given position to be part of the simulation. The unique_id and the position (x, y) of the avatar must be provided.

Args:

```
{
  "x": x,
  "y": y
}
```

Results:

```
{
  "avatar":
    {
      "position":
        {
          "x": x,
          "y": y
        },
      "id": unique_id
    }
}
```

Example:

```
{
  "x": 4,
  "y": 5
}
```

```
{
  "avatar":
    {
      "position":
        {
          "x": 3,
          "y": 5
        },
      "id": 100010
    }
}
```

---

**POST /api/v1/occupants/{id}/position**

Move an avatar object to a given position. The unique_id and the new position (x, y) of the avatar must be provided.

Args:

```
{
  "x": x,
  "y": y
}
```

Result:

```
{
  "avatar":
    {
      "position":
        {
          "x": x,
          "y": y
        },
      "id": unique_id
    }
}
```

Example:

```
{
  "x": 4,
  "y": 5
}
```

```
{
  "avatar":
    {
      "position":
        {
          "x": 4,
          "y": 5
        },
      "id": 100010
    }
}
```

---

**GET /api/v1/occupants/{id}/route/{route_id}**

Returns the path that an avatar must follow to evacuate the building based on a strategy. The unique_id of the avatar and the strategy used must be provided.

Result:

```
{
  "positions":
    [
      {
        "x": x1,
        "y": y1
      },
      {
        "x": x2,
        "y": y2
```

(continues on next page)

(continued from previous page)

```
      },
      ...
      {
        "x": xN,
        "y": yN
      }
    ]
}
```

Example:

```
{
  "positions":
    [
      {
        "y": 14,
        "x": 2
      },
      {
        "y": 14,
        "x": 1
      },
      {
        "y": 14,
        "x": 0
      }
    ]
}
```

---

**GET /api/v1/occupants/{id}/fire**

Returns the positions in the field of vision of the agent where there is fire.

Result:

```
{
  "positions":
    [
      {
        "x": x1,
        "y": y1
      },
      {
        "x": x2,
        "y": y2
      },
      ...
      {
        "x": xN,
        "y": yN
      }
    ]
}
```

Example:

```
{
  "positions":
    [
      {
        "y": 10,
        "x": 9
      },
      {
        "y": 11,
        "x": 8
      },
      {
        "y": 10,
        "x": 10
      }
    ]
}
```

---

**PUT /api/v1/occupants/{id}**

Create an EmergencyAvatar object in a given position to be part of the simulation. The unique_id and the position (x, y) of the avatar must be provided.

Args:

```
{
  "x": x,
  "y": y
}
```

Result:

```
{
  "avatar":
    {
      "position":
        {
          "x": x,
          "y": y
        },
      "id": unique_id
    }
}
```

Example:

```
{
  "x": 3,
  "y": 2
}
```

```
{
  "avatar":
    {
      "position":
```

(continues on next page)

(continued from previous page)

```
        {
          "x": 3,
          "y": 2
        },
      "id": 100001
    }
}
```

**GET /api/v1/fire**

Returns the positions where there is fire.

Result:

```
{
  "positions":
    [
      {
        "x": x1,
        "y": y1
      },
      {
        "x": x2,
        "y": y2
      },
      ...
      {
        "x": xN,
        "y": yN
      }
    ]
}
```

Example:

```
{
  "positions":
    [
      {"x": 7, "y": 9}, {"x": 8, "y": 10}, {"x": 8, "y": 9}, {"x": 6,
↪ "y": 9}, {"x": 6, "y": 8}, {"x": 7, "y": 10}, {"x": 7, "y": 8}, {
↪"x": 6, "y": 10}, {"x": 8, "y": 8}
    ]
}
```

### 6.5.2 # REST API Test

#### Test execution guide

To make a test of the REST service, the first thing to do is to make sure to run the SEBA software with the server option specified (-s), providing or not (optional) the port where the service will be defined.

```
$ git clone https://github.com/gsi-upm/soba

$ cd soba/projects/seba

$ python3 run.py -v -s
```

Then, once the software has run, following the previous case, selecting the start option of the interface in the browser. Once the simulation has started, run the 'apiTest.py' test script. If a port was defined in the execution, change the variable port of the file to the chosen value.

```
$ cd restClient

$ python3 apiTest.py
```

### Tests in a notebook

To execute the tests with greater control, it is recommended to download and use the jupyter notebook that is provided in the following address: Notebook

This notebook is also presented below.

### Initialization of params

```
from unittest import TestCase
import json, requests
from jsonschema import validate
import socket
import unittest


ipServer = socket.gethostbyname(socket.gethostname())

#Port defined when executing, 10000 default
port = '10000'

#Template of the URLs
URLBASE = "http://127.0.0.1:"+ port
URISOBA = "/api/v1/occupants"
URISEBA = "/api/v1/occupants"
URIFIRE = "/api/v1/fire"

stringTemplate = {"type": "string"}
numberTemplate = {"type": "number"}

#Number of test for each pair (URI, Method)
N = 1
```

### Tests

```
print(str('Testing {}').format('GET /api/v1/occupants'))
template = {
    "type": "object",
```

(continues on next page)

(continued from previous page)

```
    "properties": {
        "occupants": {
            "type": "array"
            }
    },
    "required": ["occupants"]
}

for i in range(N):
    url = URLBASE + URISOBA
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
    for o in datajson["occupants"]:
        validate(o, numberTemplate)
```

```
Testing GET /api/v1/occupants
Response:  {'occupants': [1, 0, 3, 100000, 2]}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/movements'))
template = {
    "type": "object",
    "properties": {
        "orientation": {
            "type": "string"
            },
        "speed": {
            "type": "number"
            }
    },
    "required": ["orientation", "speed"]
}

template2 = {
    "type": "object"
}

for i in range(N):
    url = URLBASE + URISOBA + "/movements"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template2)
    for k, v  in datajson.items():
        validate(k, stringTemplate)
        validate(int(k), numberTemplate)
        validate(v, template)
```

```
Testing GET /api/v1/occupants/movements
Response:  {'0': {'speed': 1.38, 'orientation': 'SE'}, '1': {'speed': 1.38,
↪'orientation': 'W'}, '2': {'speed': 1.38, 'orientation': 'SE'}, '3': {'speed': 1.38,
↪ 'orientation': 'E'}}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/positions'))
```

(continues on next page)

(continued from previous page)

```
template = {
    "type": "object",
    "properties": {
        "x": {
            "type": "number"
            },
        "y": {
            "type": "number"
            }
    },
    "required": ["x", "y"]
}

for i in range(N):
    url = URLBASE + URISOBA + "/positions"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    for k, v in datajson.items():
        validate(k, stringTemplate)
        validate(int(k), numberTemplate)
        validate(v, template)
```

```
Testing GET /api/v1/occupants/positions
Response:  {'0': {'y': 8, 'x': 12}, '1': {'y': 6, 'x': 0}, '2': {'y': 8, 'x': 13},
↪'100000': {'y': 7, 'x': 5}, '3': {'y': 6, 'x': 14}}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/states'))
for i in range(N):
    url = URLBASE + URISOBA + "/states"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    for k,v in datajson.items():
        validate(v, stringTemplate)
        validate(k, stringTemplate)
        validate(int(k), numberTemplate)
```

```
Testing GET /api/v1/occupants/states
Response:  {'0': 'Working in my laboratory', '1': 'Working in my laboratory', '2':
↪'Working in my laboratory', '100000': 'walking', '3': 'Working in my laboratory'}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/{id}'))
template = {
    "type": "object",
    "properties": {
        "occupant":{
            "type": "object",
            "properties": {
                    "state":{
                        "type": "string"
                    },
                    "fov": {
                        "type": "array"
                    },
```

(continues on next page)

(continued from previous page)

```
                    "unique_id":{
                        "type": "string"
                    },
                    "movement": {
                        "type": "object",
                        "properties": {
                            "orientation":{
                                "type": "string"
                            },
                            "speed":{
                                "type": "number"
                            },
                        },
                        "required": ["orientation", "speed"]
                    },
                    "position": {
                        "type": "object",
                        "properties": {
                            "x":{
                                "type": "number"
                            },
                            "y":{
                                "type": "number"
                            }
                        },
                        "required": ["x", "y"]
                    }
            },
    "required": ["state", "fov", "unique_id", "movement", "position"]
        }
    },
    "required": ["occupant"]
}

template2 = {
    "type": "object",
    "properties": {
        "x": {
            "type": "number"
            },
        "y": {
            "type": "number"
        }
    },
    "required": ["x", "y"]
}

for i in range(N):
    url = URLBASE + URISOBA + "/" + str(0)
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
    validate(int(datajson['occupant']['unique_id']), numberTemplate)
    print(template)
    for p in datajson['occupant']['fov']:
        validate(p, template2)
```

```
Testing GET /api/v1/occupants/{id}
Response:  {'occupant': {'unique_id': '0', 'fov': [{'y': 0, 'x': 9}, {'y': 0, 'x': 10}
↪, {'y': 0, 'x': 11}, {'y': 0, 'x': 12}, {'y': 0, 'x': 13}, {'y': 0, 'x': 14}, {'y':␣
↪0, 'x': 15}, {'y': 0, 'x': 16}, {'y': 0, 'x': 17}, {'y': 0, 'x': 18}, {'y': 1, 'x':␣
↪9}, {'y': 1, 'x': 10}, {'y': 1, 'x': 11}, {'y': 1, 'x': 12}, {'y': 1, 'x': 13}, {'y
↪': 1, 'x': 14}, {'y': 1, 'x': 15}, {'y': 1, 'x': 16}, {'y': 1, 'x': 17}, {'y': 1, 'x
↪': 18}, {'y': 2, 'x': 9}, {'y': 2, 'x': 10}, {'y': 2, 'x': 11}, {'y': 2, 'x': 12}, {
↪'y': 2, 'x': 13}, {'y': 2, 'x': 14}, {'y': 2, 'x': 15}, {'y': 2, 'x': 16}, {'y': 2,
↪'x': 17}, {'y': 2, 'x': 18}, {'y': 3, 'x': 9}, {'y': 3, 'x': 10}, {'y': 3, 'x': 11},
↪ {'y': 3, 'x': 12}, {'y': 3, 'x': 13}, {'y': 3, 'x': 14}, {'y': 3, 'x': 15}, {'y':␣
↪3, 'x': 16}, {'y': 3, 'x': 17}, {'y': 3, 'x': 18}, {'y': 4, 'x': 9}, {'y': 4, 'x':␣
↪10}, {'y': 4, 'x': 11}, {'y': 4, 'x': 12}, {'y': 4, 'x': 13}, {'y': 4, 'x': 14}, {'y
↪': 4, 'x': 15}, {'y': 4, 'x': 16}, {'y': 4, 'x': 17}, {'y': 4, 'x': 18}, {'y': 4, 'x
↪': 19}, {'y': 5, 'x': 9}, {'y': 5, 'x': 10}, {'y': 5, 'x': 11}, {'y': 5, 'x': 12}, {
↪'y': 5, 'x': 13}, {'y': 5, 'x': 14}, {'y': 5, 'x': 15}, {'y': 5, 'x': 16}, {'y': 5,
↪'x': 17}, {'y': 5, 'x': 18}, {'y': 5, 'x': 19}, {'y': 6, 'x': 9}, {'y': 6, 'x': 10},
↪ {'y': 6, 'x': 11}, {'y': 6, 'x': 12}, {'y': 6, 'x': 13}, {'y': 6, 'x': 14}, {'y':␣
↪6, 'x': 15}, {'y': 6, 'x': 16}, {'y': 6, 'x': 17}, {'y': 6, 'x': 18}, {'y': 6, 'x':␣
↪19}, {'y': 7, 'x': 9}, {'y': 7, 'x': 10}, {'y': 7, 'x': 11}, {'y': 7, 'x': 12}, {'y
↪': 7, 'x': 13}, {'y': 7, 'x': 14}, {'y': 7, 'x': 15}, {'y': 7, 'x': 16}, {'y': 7, 'x
↪': 17}, {'y': 7, 'x': 18}, {'y': 8, 'x': 9}, {'y': 8, 'x': 10}, {'y': 8, 'x': 11}, {
↪'y': 8, 'x': 13}, {'y': 8, 'x': 14}, {'y': 8, 'x': 15}, {'y': 8, 'x': 16}, {'y': 8,
↪'x': 17}, {'y': 8, 'x': 18}, {'y': 9, 'x': 9}, {'y': 9, 'x': 10}, {'y': 9, 'x': 11},
↪ {'y': 9, 'x': 12}, {'y': 9, 'x': 13}, {'y': 9, 'x': 14}, {'y': 9, 'x': 15}, {'y':␣
↪9, 'x': 16}, {'y': 9, 'x': 17}, {'y': 9, 'x': 18}, {'y': 10, 'x': 8}, {'y': 10, 'x
↪': 9}, {'y': 10, 'x': 10}, {'y': 10, 'x': 11}, {'y': 10, 'x': 12}, {'y': 10, 'x':␣
↪13}, {'y': 10, 'x': 14}, {'y': 10, 'x': 15}, {'y': 10, 'x': 16}, {'y': 10, 'x': 17},
↪ {'y': 10, 'x': 18}, {'y': 11, 'x': 6}, {'y': 11, 'x': 7}, {'y': 11, 'x': 8}, {'y':␣
↪11, 'x': 9}, {'y': 11, 'x': 10}, {'y': 11, 'x': 11}, {'y': 12, 'x': 4}, {'y': 12, 'x
↪': 5}, {'y': 12, 'x': 6}, {'y': 12, 'x': 7}, {'y': 12, 'x': 8}, {'y': 12, 'x': 9}, {
↪'y': 12, 'x': 10}, {'y': 12, 'x': 11}, {'y': 13, 'x': 3}, {'y': 13, 'x': 4}, {'y':␣
↪13, 'x': 5}, {'y': 13, 'x': 6}, {'y': 13, 'x': 7}, {'y': 13, 'x': 8}, {'y': 13, 'x
↪': 9}, {'y': 13, 'x': 10}, {'y': 13, 'x': 11}, {'y': 14, 'x': 1}, {'y': 14, 'x': 2},
↪ {'y': 14, 'x': 3}, {'y': 14, 'x': 4}, {'y': 14, 'x': 5}, {'y': 14, 'x': 6}, {'y':␣
↪14, 'x': 7}, {'y': 14, 'x': 8}, {'y': 14, 'x': 9}, {'y': 14, 'x': 10}, {'y': 15, 'x
↪': 0}, {'y': 15, 'x': 1}, {'y': 15, 'x': 2}, {'y': 15, 'x': 3}, {'y': 15, 'x': 4}, {
↪'y': 15, 'x': 5}, {'y': 15, 'x': 6}, {'y': 15, 'x': 7}, {'y': 15, 'x': 8}, {'y': 15,
↪ 'x': 9}, {'y': 15, 'x': 10}, {'y': 16, 'x': 0}, {'y': 16, 'x': 1}, {'y': 16, 'x':␣
↪2}, {'y': 16, 'x': 3}, {'y': 16, 'x': 4}, {'y': 16, 'x': 5}, {'y': 16, 'x': 6}, {'y
↪': 16, 'x': 7}, {'y': 16, 'x': 8}, {'y': 16, 'x': 9}, {'y': 16, 'x': 10}, {'y': 17,
↪'x': 0}, {'y': 17, 'x': 1}, {'y': 17, 'x': 2}, {'y': 17, 'x': 3}, {'y': 17, 'x': 4},
↪ {'y': 17, 'x': 5}, {'y': 17, 'x': 6}, {'y': 17, 'x': 7}, {'y': 17, 'x': 8}, {'y':␣
↪17, 'x': 9}, {'y': 18, 'x': 0}, {'y': 18, 'x': 1}, {'y': 18, 'x': 2}, {'y': 18, 'x
↪': 3}, {'y': 18, 'x': 4}, {'y': 18, 'x': 5}, {'y': 18, 'x': 6}, {'y': 18, 'x': 7}, {
↪'y': 18, 'x': 8}, {'y': 18, 'x': 9}], 'state': 'Working in my laboratory', 'position
↪': {'y': 8, 'x': 12}, 'movement': {'speed': 1.38, 'orientation': 'SE'}}}
{'type': 'object', 'required': ['occupant'], 'properties': {'occupant': {'type':
↪'object', 'required': ['state', 'fov', 'unique_id', 'movement', 'position'],
↪'properties': {'unique_id': {'type': 'string'}, 'fov': {'type': 'array'}, 'position
↪': {'type': 'object', 'required': ['x', 'y'], 'properties': {'y': {'type': 'number'}
↪, 'x': {'type': 'number'}}}, 'state': {'type': 'string'}, 'movement': {'type':
↪'object', 'required': ['orientation', 'speed'], 'properties': {'speed': {'type':
↪'number'}, 'orientation': {'type': 'string'}}}}}}}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/{id}/movement'))
template = {
    "type": "object",
```

(continues on next page)

(continued from previous page)

```
    "properties": {
        "movement":{
            "type": "object",
            "properties": {
                    "orientation": {
                        "type": "string"
                    },
                    "speed": {
                        "type": "number"
                    }
            },
        "required": ["orientation", "speed"]
        }
    },
    "required": ["movement"]
}

for i in range(N):
    url = URLBASE + URISOBA + "/" + str(0) + "/movement"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
```

```
Testing GET /api/v1/occupants/{id}/movement
Response:  {'movement': {'speed': 1.38, 'orientation': 'SE'}}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/{id}/position'))
template = {
    "type": "object",
    "properties": {
        "position":{
            "type": "object",
            "properties": {
                "x": {
                    "type": "number"
                    },
                "y": {
                    "type": "number"
                }
            },
            "required": ["x", "y"]
        }
    },
    "required": ["position"]
}

for i in range(N):
    url = URLBASE + URISOBA + "/" + str(0) + "/position"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
```

```
Testing GET /api/v1/occupants/{id}/position
Response:  {'position': {'y': 8, 'x': 12}}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/{id}/state'))
template = {
    "type": "object",
    "properties":{
        "state": {
            "type": "string"
        }
    },
    "required": ["state"]
}

for i in range(N):
    url = URLBASE + URISOBA + "/" + str(0) + "/state"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
```

```
Testing GET /api/v1/occupants/{id}/state
Response:  {'state': 'Working in my laboratory'}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/{id}/fov'))
template = {
    "type": "object",
    "properties": {
        "fov": {
            "type": "array"
            }
    },
    "required": ["fov"]
}


template2 = {
    "type": "object",
    "properties": {
        "x": {
            "type": "number"
            },
        "y": {
            "type": "number"
        }
    },
    "required": ["x", "y"]
}

for i in range(N):
    url = URLBASE + URISOBA + "/" + str(0) + "/fov"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
    for p in datajson['fov']:
        validate(p, template2)
```

```
Testing GET /api/v1/occupants/{id}/fov
```

(continues on next page)

(continued from previous page)

```
Response:  {'fov': [{'y': 0, 'x': 9}, {'y': 0, 'x': 10}, {'y': 0, 'x': 11}, {'y': 0,
↪'x': 12}, {'y': 0, 'x': 13}, {'y': 0, 'x': 14}, {'y': 0, 'x': 15}, {'y': 0, 'x': 16}
↪, {'y': 0, 'x': 17}, {'y': 0, 'x': 18}, {'y': 1, 'x': 9}, {'y': 1, 'x': 10}, {'y':␣
↪1, 'x': 11}, {'y': 1, 'x': 12}, {'y': 1, 'x': 13}, {'y': 1, 'x': 14}, {'y': 1, 'x':␣
↪15}, {'y': 1, 'x': 16}, {'y': 1, 'x': 17}, {'y': 1, 'x': 18}, {'y': 2, 'x': 9}, {'y
↪': 2, 'x': 10}, {'y': 2, 'x': 11}, {'y': 2, 'x': 12}, {'y': 2, 'x': 13}, {'y': 2, 'x
↪': 14}, {'y': 2, 'x': 15}, {'y': 2, 'x': 16}, {'y': 2, 'x': 17}, {'y': 2, 'x': 18},
↪{'y': 3, 'x': 9}, {'y': 3, 'x': 10}, {'y': 3, 'x': 11}, {'y': 3, 'x': 12}, {'y': 3,
↪'x': 13}, {'y': 3, 'x': 14}, {'y': 3, 'x': 15}, {'y': 3, 'x': 16}, {'y': 3, 'x': 17}
↪, {'y': 3, 'x': 18}, {'y': 4, 'x': 9}, {'y': 4, 'x': 10}, {'y': 4, 'x': 11}, {'y':␣
↪4, 'x': 12}, {'y': 4, 'x': 13}, {'y': 4, 'x': 14}, {'y': 4, 'x': 15}, {'y': 4, 'x':␣
↪16}, {'y': 4, 'x': 17}, {'y': 4, 'x': 18}, {'y': 4, 'x': 19}, {'y': 5, 'x': 9}, {'y
↪': 5, 'x': 10}, {'y': 5, 'x': 11}, {'y': 5, 'x': 12}, {'y': 5, 'x': 13}, {'y': 5, 'x
↪': 14}, {'y': 5, 'x': 15}, {'y': 5, 'x': 16}, {'y': 5, 'x': 17}, {'y': 5, 'x': 18},
↪{'y': 5, 'x': 19}, {'y': 6, 'x': 9}, {'y': 6, 'x': 10}, {'y': 6, 'x': 11}, {'y': 6,
↪'x': 12}, {'y': 6, 'x': 13}, {'y': 6, 'x': 14}, {'y': 6, 'x': 15}, {'y': 6, 'x': 16}
↪, {'y': 6, 'x': 17}, {'y': 6, 'x': 18}, {'y': 6, 'x': 19}, {'y': 7, 'x': 9}, {'y':␣
↪7, 'x': 10}, {'y': 7, 'x': 11}, {'y': 7, 'x': 12}, {'y': 7, 'x': 13}, {'y': 7, 'x':␣
↪14}, {'y': 7, 'x': 15}, {'y': 7, 'x': 16}, {'y': 7, 'x': 17}, {'y': 7, 'x': 18}, {'y
↪': 8, 'x': 9}, {'y': 8, 'x': 10}, {'y': 8, 'x': 11}, {'y': 8, 'x': 13}, {'y': 8, 'x
↪': 14}, {'y': 8, 'x': 15}, {'y': 8, 'x': 16}, {'y': 8, 'x': 17}, {'y': 8, 'x': 18},
↪{'y': 9, 'x': 9}, {'y': 9, 'x': 10}, {'y': 9, 'x': 11}, {'y': 9, 'x': 12}, {'y': 9,
↪'x': 13}, {'y': 9, 'x': 14}, {'y': 9, 'x': 15}, {'y': 9, 'x': 16}, {'y': 9, 'x': 17}
↪, {'y': 9, 'x': 18}, {'y': 10, 'x': 8}, {'y': 10, 'x': 9}, {'y': 10, 'x': 10}, {'y
↪': 10, 'x': 11}, {'y': 10, 'x': 12}, {'y': 10, 'x': 13}, {'y': 10, 'x': 14}, {'y':␣
↪10, 'x': 15}, {'y': 10, 'x': 16}, {'y': 10, 'x': 17}, {'y': 10, 'x': 18}, {'y': 11,
↪'x': 6}, {'y': 11, 'x': 7}, {'y': 11, 'x': 8}, {'y': 11, 'x': 9}, {'y': 11, 'x': 10}
↪, {'y': 11, 'x': 11}, {'y': 12, 'x': 4}, {'y': 12, 'x': 5}, {'y': 12, 'x': 6}, {'y
↪': 12, 'x': 7}, {'y': 12, 'x': 8}, {'y': 12, 'x': 9}, {'y': 12, 'x': 10}, {'y': 12,
↪'x': 11}, {'y': 13, 'x': 3}, {'y': 13, 'x': 4}, {'y': 13, 'x': 5}, {'y': 13, 'x': 6}
↪, {'y': 13, 'x': 7}, {'y': 13, 'x': 8}, {'y': 13, 'x': 9}, {'y': 13, 'x': 10}, {'y
↪': 13, 'x': 11}, {'y': 14, 'x': 1}, {'y': 14, 'x': 2}, {'y': 14, 'x': 3}, {'y': 14,
↪'x': 4}, {'y': 14, 'x': 5}, {'y': 14, 'x': 6}, {'y': 14, 'x': 7}, {'y': 14, 'x': 8},
↪ {'y': 14, 'x': 9}, {'y': 14, 'x': 10}, {'y': 15, 'x': 0}, {'y': 15, 'x': 1}, {'y':␣
↪15, 'x': 2}, {'y': 15, 'x': 3}, {'y': 15, 'x': 4}, {'y': 15, 'x': 5}, {'y': 15, 'x
↪': 6}, {'y': 15, 'x': 7}, {'y': 15, 'x': 8}, {'y': 15, 'x': 9}, {'y': 15, 'x': 10},
↪{'y': 16, 'x': 0}, {'y': 16, 'x': 1}, {'y': 16, 'x': 2}, {'y': 16, 'x': 3}, {'y':␣
↪16, 'x': 4}, {'y': 16, 'x': 5}, {'y': 16, 'x': 6}, {'y': 16, 'x': 7}, {'y': 16, 'x
↪': 8}, {'y': 16, 'x': 9}, {'y': 16, 'x': 10}, {'y': 17, 'x': 0}, {'y': 17, 'x': 1},
↪{'y': 17, 'x': 2}, {'y': 17, 'x': 3}, {'y': 17, 'x': 4}, {'y': 17, 'x': 5}, {'y':␣
↪17, 'x': 6}, {'y': 17, 'x': 7}, {'y': 17, 'x': 8}, {'y': 17, 'x': 9}, {'y': 18, 'x
↪': 0}, {'y': 18, 'x': 1}, {'y': 18, 'x': 2}, {'y': 18, 'x': 3}, {'y': 18, 'x': 4}, {
↪'y': 18, 'x': 5}, {'y': 18, 'x': 6}, {'y': 18, 'x': 7}, {'y': 18, 'x': 8}, {'y': 18,
↪ 'x': 9}]}
```

```
print(str('Testing {}').format('PUT /api/v1/occupants/{id}'))
template = {
    "type": "object",
        "properties": {
            "avatar":{
                "type": "object",
                "properties": {
                    "position":{
                        "type": "object",
                        "properties": {
                            "x": {
```

(continues on next page)

(continued from previous page)

```
                                "type": "number",
                            },
                            "y": {
                                "type": "number"
                            }
                        },
                        "required": ["x", "y"]
                    },
                    "id":{
                        "type": "number"
                    }
            },
            "required": ["position", "id"]
        }
    },
    "required": ["avatar"]
}

dataBody = {"x": 10, "y": 10}

for i in range(N):
    url = URLBASE + URISOBA + "/" + str(0)
    data = requests.put(url, json=dataBody, headers={'Content-Type': "application/json
↪", 'Accept': "application/json"})
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
```

```
Testing PUT /api/v1/occupants/{id}
Response:  {'avatar': {'position': {'y': 10, 'x': 10}, 'id': 100000}}
```

```
print(str('Testing {}').format('POST /api/v1/occupants/{id}/position'))
template = {
    "type": "object",
        "properties": {
            "avatar":{
                "type": "object",
                "properties": {
                    "position":{
                        "type": "object",
                        "properties": {
                            "x": {
                                "type": "number",
                            },
                            "y": {
                                "type": "number"
                            }
                        },
                        "required": ["x", "y"]
                    },
                    "id":{
                        "type": "number"
                    }
            },
            "required": ["position", "id"]
        }
```

(continues on next page)

(continued from previous page)

```
    },
    "required": ["avatar"]
}

dataBody = {"x": 5, "y": 7}

for i in range(N):
    url = URLBASE + URISOBA + "/" + str(100000) + "/position"
    data = requests.post(url, json=dataBody, headers={'Content-Type': "application/
↪json", 'Accept': "application/json"})
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
```

```
Testing POST /api/v1/occupants/{id}/position
Response:  {'avatar': {'position': {'y': 7, 'x': 5}, 'id': 100000}}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/{id}/route/{route_id}'))
template = {
    "type": "object",
    "properties": {
        "positions": {
            "type": "array"
            }
    }
}

template2 = {
    "type": "object",
    "properties": {
        "x": {
            "type": "number"
            },
        "y": {
            "type": "number"
            }
    },
    "required": ["x", "y"]
}

for i in range(N):
    url = URLBASE + URISEBA + "/" + str(100000) + "/route/1"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
    for m in datajson["positions"]:
        validate(m, template2)
```

```
Testing GET /api/v1/occupants/{id}/route/{route_id}
Response:  {'positions': [{'y': 7, 'x': 4}, {'y': 7, 'x': 3}, {'y': 7, 'x': 2}, {'y':␣
↪6, 'x': 1}, {'y': 6, 'x': 0}]}
```

```
print(str('Testing {}').format('PUT /api/v1/occupants/{id}'))
template = {
```

(continues on next page)

(continued from previous page)

```
    "type": "object",
    "properties": {
        "avatar": {
            "type": "object",
            "properties":{
                "position":{
                    "type": "object",
                    "properties":{
                        "x": {
                            "type": "number"
                        },
                        "y": {
                            "type": "number"
                        }
                    },
                    "required": ["x", "y"]
                },
                "id": {
                    "type": "number"
                }
            },
            "required": ["position", "id"]
        }
    },
    "required": ["avatar"]
}

dataBody = {"x": 13, "y": 13}

for i in range(N):
    url = URLBASE + URISEBA + "/" + str(1)
    data = requests.put(url, json=dataBody, headers={'Content-Type': "application/json
↪", 'Accept': "application/json"})
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
```

```
Testing PUT /api/v1/occupants/{id}
Response:  {'avatar': {'id': 100001, 'position': {'y': 13, 'x': 13}}}
```

```
print(str('Testing {}').format('GET /api/v1/occupants/{id}/fire'))
template = {
    "type": "object",
    "properties": {
        "positions": {
            "type": "array"
            }
    },
    "required": ["positions"]
}

template2 = {
    "type": "object",
    "properties": {
        "x": {
            "type": "number"
```

(continues on next page)

(continued from previous page)

```
            },
        "y": {
            "type": "number"
            }
    },
    "required": ["x", "y"]
}

for i in range(N):
    url = URLBASE + URISEBA + "/" + str(2) + "/fire"
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
    for m in datajson["positions"]:
        validate(m, template2)
```

```
Testing GET /api/v1/occupants/{id}/fire
Response:  {'positions': [{'y': 4, 'x': 12}, {'y': 5, 'x': 13}, {'y': 4, 'x': 13}, {'y
↪': 4, 'x': 11}, {'y': 3, 'x': 11}, {'y': 5, 'x': 12}, {'y': 3, 'x': 12}, {'y': 5, 'x
↪': 11}, {'y': 3, 'x': 13}, {'y': 6, 'x': 14}, {'y': 5, 'x': 14}, {'y': 6, 'x': 13},
↪{'y': 6, 'x': 12}, {'y': 4, 'x': 14}, {'y': 3, 'x': 14}, {'y': 4, 'x': 10}, {'y': 3,
↪ 'x': 10}, {'y': 5, 'x': 10}, {'y': 2, 'x': 10}, {'y': 2, 'x': 11}, {'y': 2, 'x':␣
↪12}, {'y': 6, 'x': 11}, {'y': 2, 'x': 13}, {'y': 6, 'x': 10}, {'y': 2, 'x': 14}, {'y
↪': 7, 'x': 15}, {'y': 6, 'x': 15}, {'y': 7, 'x': 14}, {'y': 7, 'x': 13}, {'y': 5, 'x
↪': 15}, {'y': 4, 'x': 15}, {'y': 7, 'x': 12}, {'y': 7, 'x': 11}, {'y': 3, 'x': 15},
↪{'y': 2, 'x': 15}, {'y': 1, 'x': 10}, {'y': 1, 'x': 11}, {'y': 1, 'x': 12}, {'y': 1,
↪ 'x': 13}, {'y': 7, 'x': 10}, {'y': 1, 'x': 14}, {'y': 1, 'x': 15}, {'y': 8, 'x':␣
↪16}, {'y': 7, 'x': 16}, {'y': 8, 'x': 15}, {'y': 8, 'x': 14}, {'y': 6, 'x': 16}, {'y
↪': 5, 'x': 16}, {'y': 8, 'x': 13}, {'y': 8, 'x': 12}, {'y': 4, 'x': 16}, {'y': 3, 'x
↪': 16}, {'y': 8, 'x': 11}, {'y': 8, 'x': 10}, {'y': 2, 'x': 16}, {'y': 1, 'x': 16}]}
```

```
print(str('Testing {}').format('GET /api/v1/fire'))
template = {
    "type": "object",
    "properties": {
        "positions": {
            "type": "array"
            }
    },
    "required": ["positions"]
}

template2 = {
    "type": "object",
    "properties": {
        "x": {
            "type": "number"
            },
        "y": {
            "type": "number"
            }
    },
    "required": ["x", "y"]
}
```

(continues on next page)

(continued from previous page)

```
for i in range(N):
    url = URLBASE + URIFIRE
    data = requests.get(url)
    datajson = data.json()
    print("Response: ", datajson)
    validate(datajson, template)
    for m in datajson["positions"]:
        validate(m, template2)
```

```
Testing GET /api/v1/fire
Response:  {'positions': [{'y': 4, 'x': 12}, {'y': 5, 'x': 13}, {'y': 4, 'x': 13}, {'y
↪': 4, 'x': 11}, {'y': 3, 'x': 11}, {'y': 5, 'x': 12}, {'y': 3, 'x': 12}, {'y': 5, 'x
↪': 11}, {'y': 3, 'x': 13}, {'y': 6, 'x': 14}, {'y': 5, 'x': 14}, {'y': 6, 'x': 13},
↪{'y': 6, 'x': 12}, {'y': 4, 'x': 14}, {'y': 3, 'x': 14}, {'y': 4, 'x': 10}, {'y': 3,
↪ 'x': 10}, {'y': 5, 'x': 10}, {'y': 2, 'x': 10}, {'y': 2, 'x': 11}, {'y': 2, 'x':␣
↪12}, {'y': 6, 'x': 11}, {'y': 2, 'x': 13}, {'y': 6, 'x': 10}, {'y': 2, 'x': 14}, {'y
↪': 7, 'x': 15}, {'y': 6, 'x': 15}, {'y': 7, 'x': 14}, {'y': 7, 'x': 13}, {'y': 5, 'x
↪': 15}, {'y': 4, 'x': 15}, {'y': 7, 'x': 12}, {'y': 7, 'x': 11}, {'y': 3, 'x': 15},
↪{'y': 2, 'x': 15}, {'y': 1, 'x': 10}, {'y': 1, 'x': 11}, {'y': 1, 'x': 12}, {'y': 1,
↪ 'x': 13}, {'y': 7, 'x': 10}, {'y': 1, 'x': 14}, {'y': 1, 'x': 15}, {'y': 8, 'x':␣
↪16}, {'y': 7, 'x': 16}, {'y': 8, 'x': 15}, {'y': 8, 'x': 14}, {'y': 6, 'x': 16}, {'y
↪': 5, 'x': 16}, {'y': 8, 'x': 13}, {'y': 8, 'x': 12}, {'y': 4, 'x': 16}, {'y': 3, 'x
↪': 16}, {'y': 8, 'x': 11}, {'y': 8, 'x': 10}, {'y': 2, 'x': 16}, {'y': 1, 'x': 16}]}
```

# Python Module Index

## p



## s

# Index

## I



## M



## N



## O



## P



## R

## S



## T



## W



## X