# RAMEN Documentation

***Release 1.0***

**Pablo Aznar**

**Jan 30, 2018**

# Contents


**1 What is RAMEN? 3**
- 1.1 Architecture . . . 4

**2 Installation 7**

**3 How to use 9**
- 3.1 Creating the floor plan . . . 9
- 3.2 Setting Movement . . . 11
- 3.3 Starting the simulation . . . 13

**4 Objects 15**
- 4.1 Type 1: Floor Item . . . 16
- 4.2 Type 2: Wall Item . . . 19
- 4.3 Type 3: In Wall Item . . . 20
- 4.4 Type 7: In Wall Floor Item . . . 20
- 4.5 Type 8: On Floor Item . . . 21

**5 Sentiments 23**

Contents:

CHAPTER 1

# What is RAMEN?

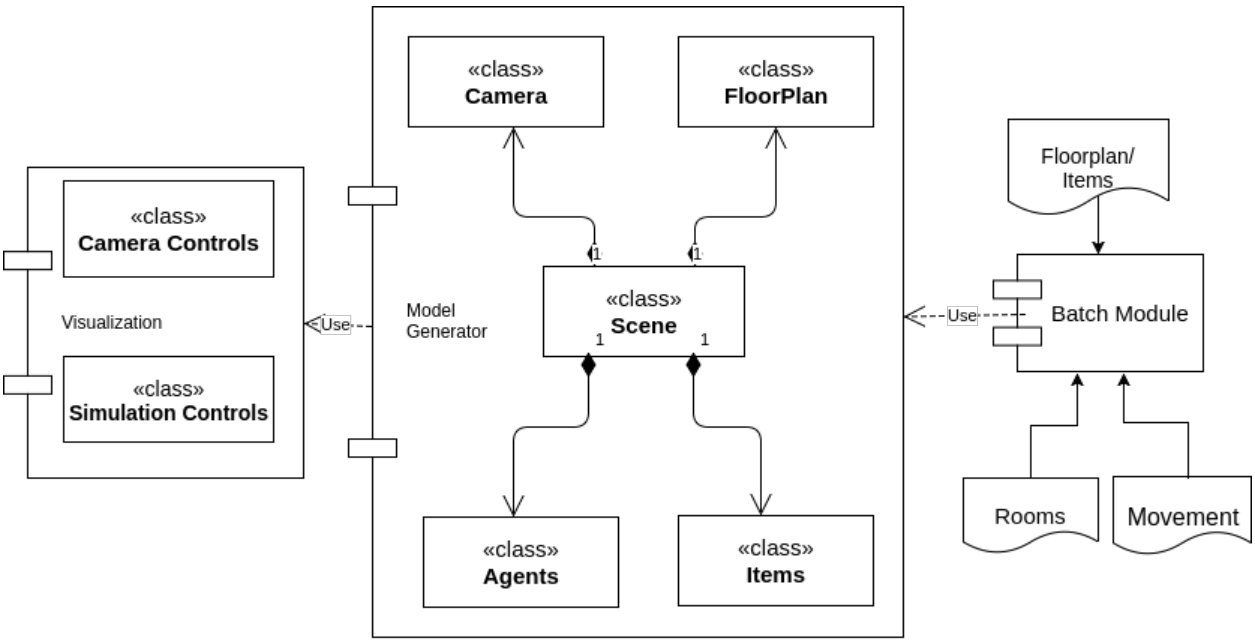
Representation of Animated Multitudes in ENvironments.

RAMEN 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.

## 1.1 Architecture

RAMEN is divided in three main modules:

- Batch module: this module is in charge of collecting the data proportionated by the user, such as the floor plan, the rooms assignation and the movement of the agents.
- Model generator: thanks to the data collected by the batch module, it is able to generate the scene in which there are the different elements that are going to be represented. The main elements are the floor plan, the agents, the items and the camera. All of them make use of the renderer which proportionates the data to represent the visualization.
- Visualization module: it manages the final result of the visualization on the browser.


«class»
Camera
«class»
FloorPlan
Floorplan/
Items
«class»
Camera Controls
Visualization
«class»
Simulation Controls
Use
Model
Generator
1
1
«class»
Scene
1
1
Use
Batch Module
Rooms
Movement
«class»
Agents
«class»
Items

CHAPTER 2

# Installation

First of all, you need to clone the GitHub repository:

```
git clone https://github.com/pablo-aznard/blueprint3d
cd ramen
```

Once you have downloaded RAMEN, you have to ensure that you have installed npm and grunt. Then, execute the following commands:

```
npm install
grunt
```

The latter command generates *"executable/js/ramen.js"* from *src*.

The easiest way to run locally is to run a local server from the *executable* directory.

```
cd executable
python visual.py
```

Then, visit *http://localhost:8001* in your browser.

CHAPTER 3

# How to use

## 3.1 Creating the floor plan

First of all, you need to create the floor plan in which the simulation is going to take place. For that purpose you can import it as a JSON file or use the floorplanner tool.

The structure of the JSON file is the following:

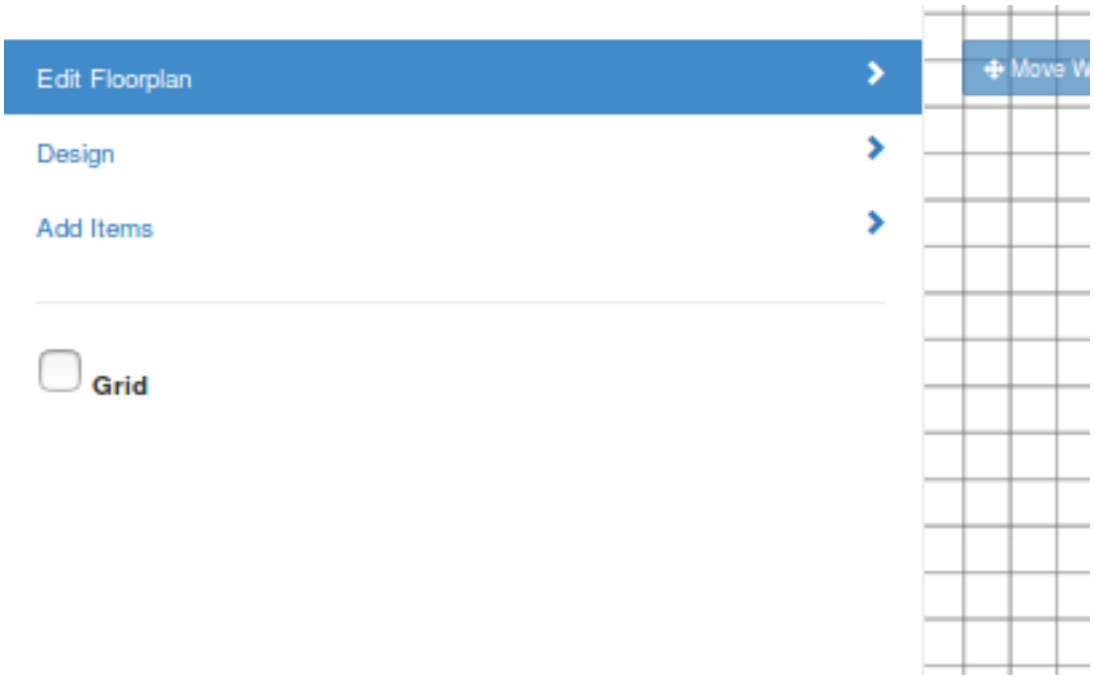
```
{"floorplan": {
        "corners": {
                "C1": {"x": 0, "y": 0},
                "C2": {"x": 200, "y": 0}
        },
        "walls": [{
                "corner1": "C1",
                "corner2": "C2",
                "frontTexture": {
                        "url": "rooms/textures/wallmap.png",
                        "stretch": true,
                        "scale": 0
                },
                "backTexture": {
                        "url": "rooms/textures/wallmap.png",
                        "stretch": true,
                        "scale": 0
                }
        }],
},
"items": [{
        "item_name": "Open Door",
        "item_type": 7,
        "model_url": "../models/js/open_door.js",
        "xpos": 100,
        "ypos": 0,
        "zpos": 0,
```

```
        "rotation": 0,
        "scale_x": 1,
        "scale_y": 1,
        "scale_z": 1,
        "fixed": false
}]
}
```

In this JSON is defined all the floor plan and the items in it. The code above is an example of a wall of 2 metres long with a door in the middle. In order to define it, you have to follow this steps:

- Corners: to declare a corner, it has to be proportionated the id of the corner and its coordinates x and y.
- Walls: The textures indicate the color of the wall and it is proportionated by a PNG image. Each wall has two different textures instead of one because a wall could be shared between two different rooms and those rooms could have different wall's color.
- Items: to declare an item, it has to be defined the item's name, the item's type, the model's url to the 3D model file and the position, rotation and scale.

It is also possible to generate the scenario using the floor planner tool. For that purpose, you have to click *Edit Floorplan* in the left menu and draw the floorplan using the interface. Once the floor plan is created, you can place the furniture clicking in *Add Item* in the left menu. When you have finished, you can save the floorplan clicking the button *Save Plan*.




In order to load an specific plan when loading RAMEN, it has to be located in *"/executable/js/maps/map.json"* or it can be loaded using the *Load Plan* button.

Furthermore, to identify a room by a specific name it has to be created the rooms JSON in *"/executable/js/maps/rooms.json"* or loaded using the *Load Rooms* button. It has to be declared the name and the coordinates of the center of the room with the following structure:

```
{"room":
        [{
                "name": "Hall.1",
                "x": 500,
```

```
                "y": 400
        }]
}
```

## 3.2 Setting Movement

Once the floor plan is created and loaded, we have to define the actions that are going to occur in the simulation. The different actions that can be executed are the following ones:

- Add new agent: it is needed an agent id, its position and its sentiment. The ids have to be defined in order, starting in 0.
- Turn on/off lights: it is needed a parameter that indicates if the light has to be turn on or off and the room in which this action is wanted to be executed.
- Turn on/off TV: it is needed a parameter that indicates if the TV has to be switched on or off and the room in which this action is wanted to be executed.
- Move an agent: it is declared in different ways depending on which type of simulation are we running. In the following examples is shown how it has to be declared.
- Add fire: the possibility of adding fire is implemented in order to represent fire evacuations. It is needed a parameter that indicates if there is fire and its position.
- Change agent's sentiment: it is needed the agent id and the sentiment. This provides the possibility of changing the sentiment of the agent in any step. It can also be changed when the movement is declared, so the agent's sentiment will change before the movement starts. To do so, it is necessary to add the attribute *sentiment* in the declaration.
- Remove an agent: in order to represent an agent leaving the building, the *outBuilding* attribute is used. It is needed the agent id and the parameter that indicates that it is out of the building.

These actions can be executed in real time or in batch mode.

In order to execute them in batch mode, a JSON file is declared. It is divided in steps and in each one, all the actions that happen in that moment are defined. The steps are a way of measuring time and everyone has the same duration, **100 ms**. There are three different ways to declare them:

- Type 0: it defines the position of the agents by rooms.

```
{
        "type" : 0,
        "steps": [
        [
                {
                "agent": 0,
                "position": "C.10",
                "sentiment": "anger"
                },
                {
                "agent": 1,
                "position": "C.1",
                "sentiment": "happiness"
                }
        ],
        [
                {
                "light": false,
```

```
                "room": "Lab1"
                },
                {
                "video": true,
                "room": "Office3"
                },
                {
                "fire": true,
                "room": "Office1"
                },
        ],
        [
                {
                "agent": 0,
                "moveTo": "C.2",
                "toStep": 15
                },
                {
                "agent": 1,
                "sentiment": "sadness",
                }
        ]
        ]
}
```

- Type 1: it defines the position of the agents by coordinates.

```
{
        "type" : 1,
        "steps": [
                [
                        {
                        "agent": 0,
                        "position": {"x": 800, "y": 500},
                        "sentiment": "hapiness"
                        }
                ],
                [
                        {
                        "agent": 0,
                        "moveTo": {"x": 2000, "y": 400},
                        "toStep": 10
                        }
                ]
        ]
}
```

- Type 2: it defines the movement of the agents by direction and speed.

```
{
        "type": 2,
        "steps": [
        [
                {
                        "agent": 0,
                        "position": {"x": 250, "y": 100},
                        "sentiment": "happiness",
                        "rotation": "E"
```

```
                }
        ],
        [
                {
                        "agent": 0,
                        "direction": "E",
                        "speed": 1
                }
        ],
        [
                {
                        "agent":0,
                        "stop":true
                }
        ]
        ]
}
```

This file has to be loaded using the interface that appears when the play button is selected. In the other hand, when the play button is pressed, the option of executing the actions in real time is also available. If we click on it, the socket will be opened and ready to receive the actions. These actions has to be defined with the same structure as above, but only one step. In order to send the data, it can be done in the following way:

```
curl -d @data.json -X POST http://localhost:8001/api
```

Being *data.json* the data that is wanted to be send.

## 3.3 Starting the simulation

Finally, when the floor plan and the movements are declared, we are ready to play the simulation. The camera can be controlled using the mouse or the camera buttons placed at the bottom of the display. Furthermore, the simulation can be started, paused or played faster using the simulation controls placed at the top of the display.

CHAPTER 4

# Objects

In order to make the process of adding objects to the scene easier, different types of objects are defined:

## 4.1 Type 1: Floor Item

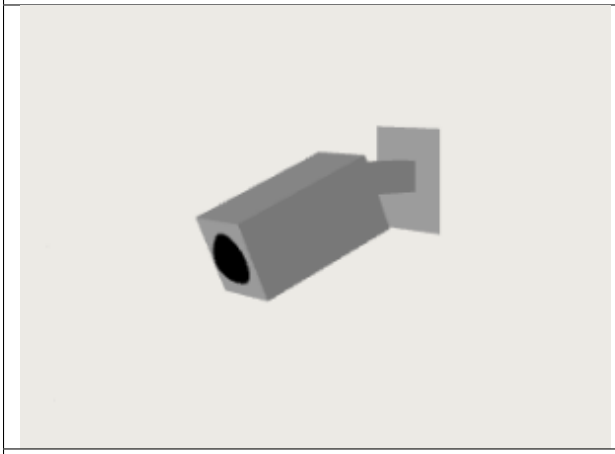
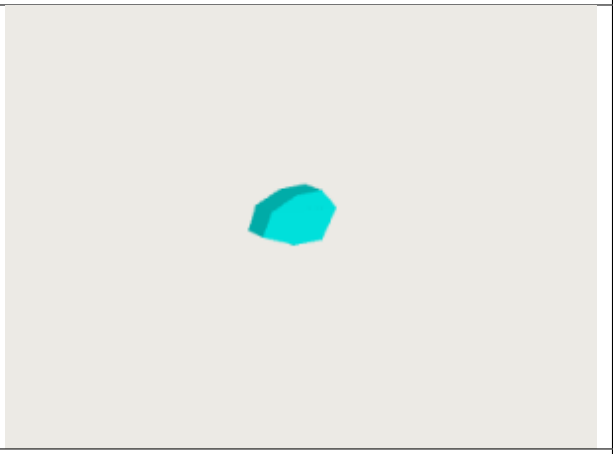
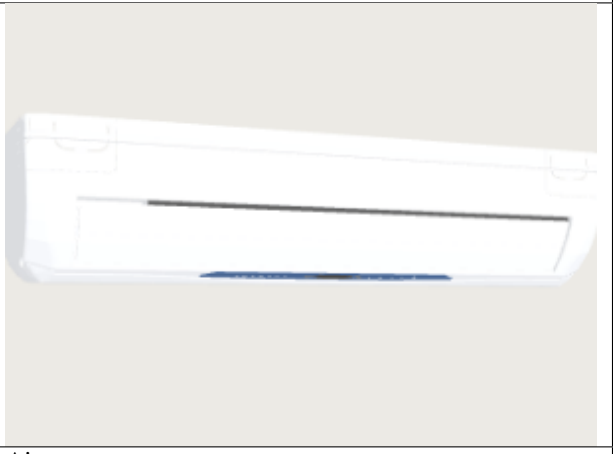
### 4.1.1 Chairs

| | |
|---|---|
|  |  |
| Chair | Blue Chair |
|  |  |
| Red Chair | Sofa 1 |
|  | |
| Sofa 2 | |

### 4.1.2 Furniture

| | |
|---|---|
|  |  |
| Bedside table - White | Bedside Table |
|  |  |
| Table | Side Table |
|  |  |
| Dresser - White | Dresser - Dark Wood |
|  |  |

## 4.2 Type 2: Wall Item

| | |
|---|---|
|  |  |
| GSI Poster | NYC Poster |
|  |  |
| Camera | Beacon |
|  |  |
| Board | Air |

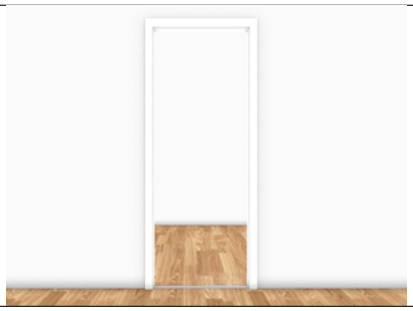
## 4.3 Type 3: In Wall Item

### 4.3.1 Window

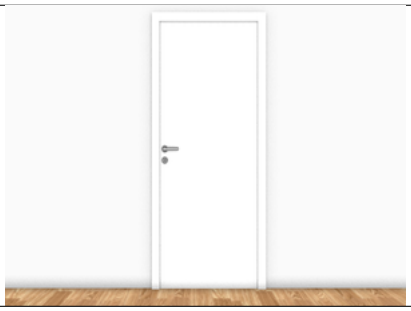
Window

## 4.4 Type 7: In Wall Floor Item

### 4.4.1 Doors



Open Door



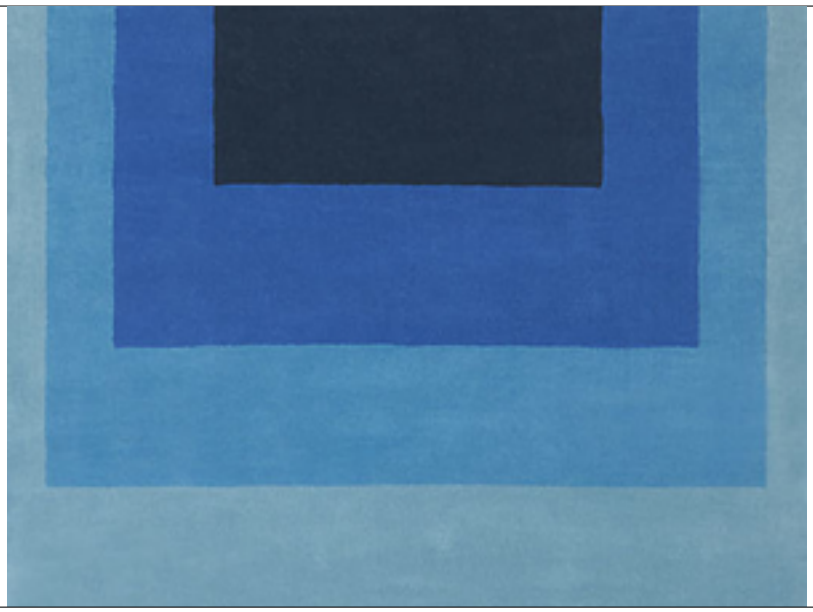
Closed Door

Out Door

The Out Door is like the Open Door but it is used to identify the exit of the map.

## 4.5 Type 8: On Floor Item

### 4.5.1 Rug



Rug

# CHAPTER 5

## Sentiments

In order to be able to identify the sentiment of the agent, it has been chosen to change the agent's color. The colors that represent each sentiment are:

- Happiness: yellow.
- Anger: red.
- Fear: green
- Disgust: purple.
- Surprise: light blue.
- Sadness: dark blue.


Happiness
Anger
Fear
Disgust
Surprise
Sadness