UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR
DE INGENIEROS DE TELECOMUNICACIÓN

GRADO EN INGENIERÍA DE TECNOLOGÍAS Y SERVICIOS DE TELECOMUNICACIÓN

TRABAJO FIN DE GRADO

DESIGN AND IMPLEMENTATION OF AN AGENT-BASED CROWD SIMULATION MODEL FOR EVACUATION OF UNIVERSITY BUILDINGS USING PYTHON

GUILLERMO FERNÁNDEZ ESCOBAR

2017

TRABAJO FIN DE GRADO

Título:	Diseño e implementación de un Modelo de Simulación Social para Evacuación en un Edificio de la Universidad usando Python
Título (inglés):	Design and Implementation of an Agent-based Crowd Simulation Model for Evacuation of University Buildings Using Python
Autor:	Guillermo Fernández Escobar
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:

Vocal:

Secretario:

Suplente:

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO FIN DE GRADO

DESIGN AND IMPLEMENTATION OF AN AGENT-BASED CROWD SIMULATION MODEL FOR EVACUATION OF UNIVERSITY BUILDINGS USING PYTHON

Guillermo Fernández Escobar

Junio de 2017

Resumen

Hoy en día vemos muchas noticias sobre tragedias que involucran evacuaciones, por lo que es muy importante estudiar este tema y definir protocolos de evacuación eficaces. Para definir protocolos efectivos de evacuación, es esencial comprender el comportamiento de la gente en estos casos. Muchos estudios se centran en analizar diferentes tipos de comportamientos y definir reglas para manejar a la multitud en dichos casos. Por lo tanto, usaremos estas reglas y estos comportamientos para simularlo en escenarios reales y sacar conclusiones.

Para resolver este problema usamos la simulación social basada en agentes (ABSS, en inglés). ABSS consiste en simulaciones sociales que se basan en el modelado de agentes, y son implementadas usando tecnologías de agentes artificiales. ABSS es una disciplina científica relacionada con la simulación de fenómenos sociales, utilizando modelos multiagentes basados en computadoras. En estas simulaciones, las personas o grupos de personas están representados por agentes. ABSS modela los diferentes elementos de los sistemas sociales utilizando agentes artificiales y los coloca en una sociedad simulada por ordenador para observar su comportamiento. A partir de estos datos es posible conocer las reacciones de los agentes artificiales y traducirlos en resultados de agentes no artificiales.

Este proyecto consistirá en simular el edificio de una universidad a través de la plataforma Mesa, con el que tendremos el plano más real posible de uno de los edificios de la ETSIT. Además del uso de Mesa, en las simulaciones los agentes serán modelados usando Python.

El plano creado será el entorno de trabajo donde los agentes pueden simular la evacuación del edificio haciendo que estos intenten abandonar el edificio tan pronto como sea posible. Por lo tanto, este proyecto generará diferentes informes de acuerdo con los criterios que se aplican a los escenarios simulados. Por ejemplo, el caso más simple sería en el que los agentes buscan la salida más cercana. Por otro lado también se pueden hacer otros tipos de políticas; que los agentes intenten salir en grupos, porque son familia, equipo de trabajo, etc ... También se tendrá en cuenta que el fuego va aumentando según pasa el tiempo.

Palabras clave: Simulación Social basada en agentes, Mesa, Evacuación

Abstract

Nowadays we see a lot of news about tragedies involving crowd evacuations, so it is very important study this topic and define effective evacuations protocols. In order to define effective evacuation protocols, understanding disasters and crowd emergency evacuation behavior is essential. Many studies focus on analyzing different types of behaviors and defining rules to handle crowd in case of evacuation. Therefore we will use these rules and these behaviors to simulate it in real scenarios and make conclusions.

To resolve this problem we use Agent-based Social Simulation (ABSS). ABSS consists of social simulations that are based on agent-based modeling, and implemented using artificial agent technologies. ABSS is scientific discipline concerned with simulation of social phenomena, using computer-based multiagent models. In these simulations, persons or group of persons are represented by agents. ABSS models the different elements of the social systems using artificial agents and placing them in a computer simulated society to observe the behaviors of the agents. From this data it is possible to learn about the reactions of the artificial agents and translate them into the results of non-artificial agents and simulations.

This project will consist of simulating a school building through the Mesa platform, with which we will have the most real possible plan of one of the ETSIT buildings. In addition to the use of Mesa, in the simulations the agents (people) will be modelled through Python.

The plan created will be the work environment where agents can simulate the evacuation of the building causing agents to try to leave the building as soon as possible. Therefore this project will generate different reports according to the criteria that apply to the simulated scenarios. For example the simplest case would be in which these agents look for the nearest exit. On the other hand also can be made other types of policies as it is that the agents look for to leave in groups, because they are family, team of work, etc... It will also take into account that in case of fire the fire is progressing so you will get reliable reports of the different scenarios that arise.

Keywords: Agent-Based Social Simulation, Mesa, Evacuation

Agradecimientos

En este apartado me gustaría mencionar a aquellas personas que han estado ahí apoyándome para que este momento haya sido posible.

Primero quiero agradecer a Carlos toda la ayuda que me ha ofrecido durante toda la carrera y durante este proyecto en el que siempre que he necesitado cualquier cosa ha estado ahí.

Al GSI por ayudarme en todo momento cuando necesitaba algo. En particular a Pablo y a Eduardo con los que he trabajado codo con codo.

Después quiero agradecerles a mi familia, a mis padres y a mis hermanos el gran apoyo que son siempre para mí y que durante esta carrera me han dado. Hacen que conseguir mis sueños sea más fácil.

Gracias a mis amigos tanto de la Escuela como de fuera. No voy a nombrar a nadie porque ellos saben quiénes son.

Y por último gracias a ti por estar siempre a mi lado.

Contents

List of Figures

CHAPTER 1

Introduction

This chapter provides an introduction to the problem which will be approached in this project, what the project's main goals are, and finally a brief description of this document and its chapters.

1.1 Context

Nowadays we see a lot of news about tragedies involving crowd evacuations, in these cases many people were injured by the chaos produced because they did not know how manage these situations, so it is very important study how people behave in those difficult moments and define effective evacuations protocols. Emergency evacuation, known as egress, is a critical component of emergency response and requires developing in advance evacuation preparation activities that ensure that people can get to safety in case of emergency.In order to define effective evacuation protocols, understanding disasters and crowd emergency evacuation behavior is essential [1].

There are many research of theories about mass psychology with the objective of understand crowd behaviors in emergencies from several points of view: decision making, exit times, clinical issues and crowd behavior [1] [2].

For example, one of the most influential is exit selection and the time it takes to evacuate. Aspects, such as their familiarity with exits or their visibility are very important to choose the exit way. People personality has become in other relevant aspect because it also influences to follow or cooperate with other users when they have to select the exit [3].

From the other points of view there are a lot of models and theories. Some researchers have analyzed clinical issues, such as freezing or becoming disassociated from reality, which are also potentially dangerous [2]. However, those researchers found other interesting findings; around 50 percent of emergency survivors referred unambiguously to a sense of unity or togetherness with the rest of the crowd during the emergency. Also there is one model which explains why family groups often escape or die together, this is the affiliation model [2].

Based on these concepts, there are tools to model and simulate emergency evacuation using social simulation. We have used Agent-Based Social Simulation(ABSS) to implement some of these previously named concepts. ABSS is detailedly explained in the next chapter together with Mesa [4] which is a new open-source, Apache 2.0 licensed Python package that allows users to quickly create agent-based models.

In order to achieve the project's objectives, this project will consist of simulating a school building through the Mesa platform, with which we will have the most real possible plan of one of the ETSIT buildings. In addition to the use of Mesa, in the simulations the agents (people) will be modeled through Python with the objective of use the models and behaviors studied to simulate it in real scenarios and make conclusions.

1.2 Project goals

The main goal of this project is develop a Crowd Evacuation Simulator Based on Agents (CESBA) making use of Mesa [4] . This project will generate different reports according to the criteria that apply to the simulated scenarios. These scenarios would be the most basic being possible to implement and analyze other more complex scenarios for further investigation and thus establish certain evacuation protocols to successfully achieve the exit of the building as soon as possible.

This main goal includes some tasks such as:

- Design scenarios and policies.
- Implementation scenarios and policies which are previously designed.
- Simulate and testing the differents situations we have studied.

- Analyze the results of the simulation.
- Technical report writing.

1.3 Structure of this document

In this section we provide a brief description of each chapter included in this document. The structure is the following:

Chapter 1. Introduction. provides an introduction to the problem which will be approached in this project. It provides an overview of the context of the project.

Chapter 2. Background. This chapter gives a description of the main standards and technologies on which this project rely on.

Chapter 3. Architecture. This chapter describes the architecture.

Chapter 4. Case Study. This chapter offers an overview of the main use case. The running of the modules and their functionalities are explained, and the steps that a user has to follow to use this system.

Chapter 5. Conclusions. This chapter sums up the findings and conclusions found throughout the document and gives a hint about future development to continue the work done for this project.

CHAPTER 2

Background

In this chapter, the enabling technologies are introduced. Enabling technologies are the various already functional platforms that are used in order to implement this project.The main technology we have used is Mesa, an agent-based modeling framework.

2.1 Overview

Social simulation is a research field that applies computational methods to study issues in the social sciences. The issues explored include problems in psychology, organizational behavior, sociology, political science, economics, anthropology, geography, engineering, archeology and linguistics [5]. This research field aims to cross the gap between the descriptive approach used in the social sciences and the formal approach used in the natural sciences, by moving the focus on the processes/mechanisms/behaviors that build the social reality.

The idea of simulation is to construct a computer program that has some of the properties of a 'real world' social process and observe what happens when the program runs.

Three regional associations promote and coordinate the social simulation, these associations are ESSA for Europe, CSSS for North America and PAAA Pacific Asia.

Four are the historical modalities of computational social simulation [6]: the simulation of population flows (System Dynamics, SD), the simulation of events flows (Stochastic Processes, SP), the simulation of Interactive individual behavior (Cellular Automata, CA) and the simulation of multi-agent based systems (MABS).

For this project the more interesting is simulation of multi-agent based systems which has three principal parts [6]:

- Environment: The environment models the physical or environmental context of the simulation, through a set of own rules that govern the dynamic of the model.
- Agents: Agents model individuals by their particular attributes. There may be several types of agents whose behavior is determined by differents rules of behavior.
- Rules: It makes agents and environment interact according to them.

Among the different existing types of simulations, the use of agent-based social simulation (ABSS) is one of the most representative research streams. ABSS [7] consists of social simulations that are based on agent-based modeling, and implemented using artificial agent technologies. Agent-based social simulation is scientific discipline concerned with simulation of social phenomena, using computer-based multi-agent models. In these simulations, persons or group of persons are represented by agents. ABSS models the different elements of the social systems using artificial agents and placing them in a computer simulated society to observe the behaviors of the agents. From this data it is possible to learn about the reactions of the artificial agents and translate them into the results of non-artificial agents and simulations.

Building a multi-agent social simulation is a long and complete task if you use basic programming techniques without any help. There are a minimum of four options, with incremental complexity in terms of their computational technology, to be able to construct a social simulation model:

- Program the entire code.
- Use "libraries" of prefabricated code to write the code itself.
- Use of "platforms" or integrated environments, with programming aids without directly writing the code.
- Meta-model tools that automatically generate executable code from a formal abstract specification and model.

Continuing with meta-model tools there are several that they are being introduced briefly:

- MASON [8] is a fast discrete-event multiagent simulation library core in Java, designed to be the foundation for large custom-purpose Java simulations, and also to provide more than enough functionality for many lightweight simulation needs. MASON contains both a model library and an optional suite of visualization tools in 2D and 3D.

- Repast [9] is a family of advanced, free, and open source agent-based modeling and simulation platforms that have collectively been under continuous development for over 15 years. Repast Simphony 2.4.0, released on 30 September 2016, is a richly interactive and easy to learn Java-based modeling system that is designed for use on workstations and small computing clusters. Repast for High Performance Computing 2.2.0, released on 30 September 2016, is a lean and expert-focused C++-based modeling system that is designed for use on large computing clusters and supercomputers.

- NetLogo [10] is a multi-agent programmable modeling environment. It is used by tens of thousands of students, teachers and researchers worldwide. It also powers HubNet participatory simulations. It is authored by Uri Wilensky and developed at the CCL.

- Mesa [4] is explained detailedly in the following section because it is the principal tool that this project use to model and simulate.

- UbikSim [11] is a framework used to develop social simulation which emphasizes the construction of realistic indoor environments, the modeling of realistic human behaviours and the evaluation of Ubiquitous Computing and Ambient Intelligence systems. UbikSim is written in Java and employs a number of third-party libraries such as Sweet Home 3D [12] and MASON [8].

2.2 Mesa

Mesa [4] is a new open-source, Apache 2.0 licensed Python package that allows users to quickly create agent-based models using built-in core components (such as agent schedulers and spatial grids) or customized implementations; visualize them using a browser-based interface; and analyze their results using Python's data analysis tools.

Preiously we have mentioned that there are several tools and frameworks in use for agent-based modeling, such as Netlogo [10], Repast [9] and MASON [8]. All of them share

that they do not use Python whereas Mesa do it. Python allows interactive analysis of model output data, through the IPython Notebook [13].

IPython provides the following features:

- A powerful interactive shell.
- A browser-based notebook with support for code, text, mathematical expressions, in line plots and other media.
- Support for interactive data visualization and use of GUI toolkits.
- Flexible, embeddable interpreters to load into one's own projects.
- Easy to use, high performance tools for parallel computing.

Mesa can implements several agent schedulers and require the modeler to specify which one is being used. Also it implements several useful tools to accelerate common model analysis tasks: a data collector (present only in Repast) and a batch runner (available in Repast and NetLogo), both of which can export their results directly to pandas data frame format for immediate analysis. On other side this framework facilitates such live visualization as well.

Now we introduce a brief of Mesa's core features:

2.2.1 Arquitecture

Mesa's architecture is modularity. It divide the modules into three overall categories: model, analysis and visualization.

- **Model**
  The model components are: Agents, modeling by a Model class, scheduler which controls the agent activation regime and handles time in the model in general and components describing the space.
- **Analysis**
  Those modules provide useful tools for getting data out of your model runs to study more systematically. The two more important modules are: data collectors, are used to record data from each model run, and batch runners, automate multiple runs and parameter sweeps – running the model with different parameters, to see how they change its behavior.

- **Visualization**

  Visualization system is divided into two parts: the server side, and the client side. The server runs the model, and at each step extracts data from it to visualize, which it sends to the client as JSON via a WebSocket connection. The client receives the data, and uses JavaScript to actually draw the data onto the screen for the user. The client front-end also includes a GUI controller, allowing the user to start a model run, pause it, advance it by one step, reset the model, and set the desired frame-rate.

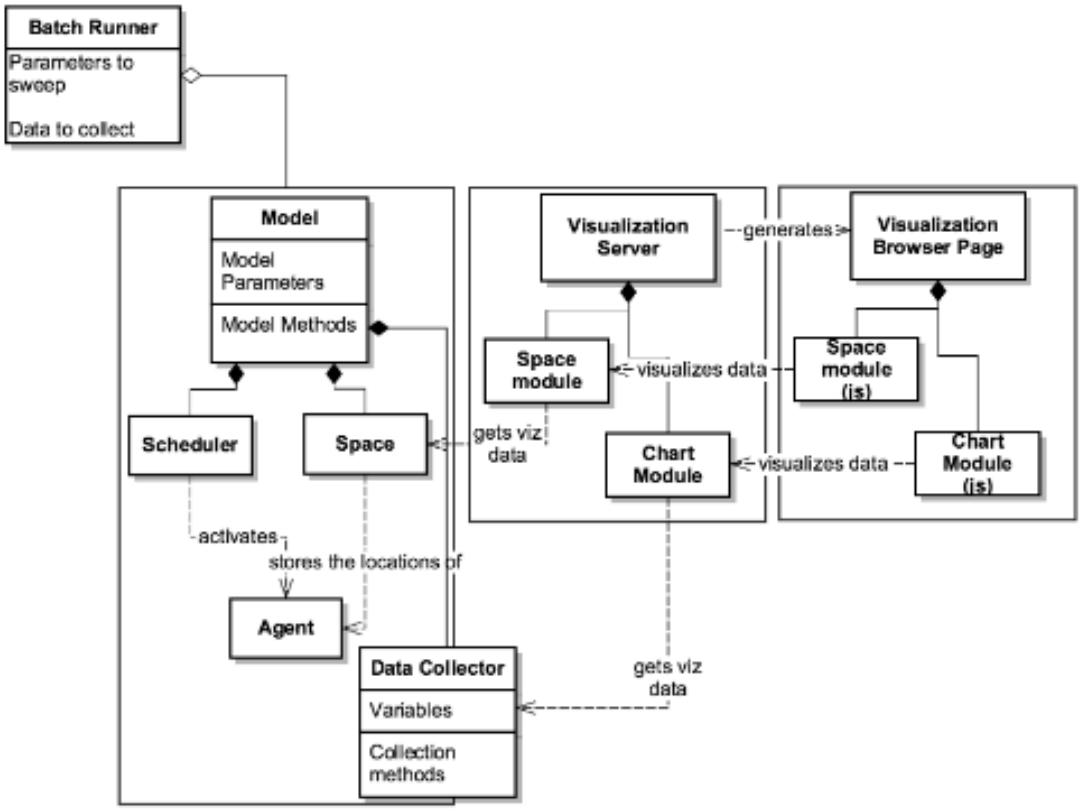


Figure 2.1: Mesa

2.3 Soba

SOBA [14] is a Simulator of Occupancy Based on Agent useful for develop studies and research based on social simulation, mainly in buildings. The simulations are configured by declaring a type of occupant, with specific and definable behavior, a physical space (rooms of the building) and agents that interconnect with each other and with the occupants. The simulation and results can be evaluated both in real time and post-simulation.

2.3.1 Arquitecture

The architecture of SOBA is divided into several levels.

- *The Model and the Agents*: Agents will perform actions and interactions within a model. The model will control the agents by means of 'steps' in the time and positions (x,y) in a grid as space.
- *Space*: In the simulations the agents move by rooms of a definable building.
- *Visualization and Results modules*: Simulations have two options, one of them is running in background or running with visualization in a browser by using Canvas.js
- *Configuration files*: DefineOccupancy and DefineMap. Using JSON's, we can define the scenario and the agents.

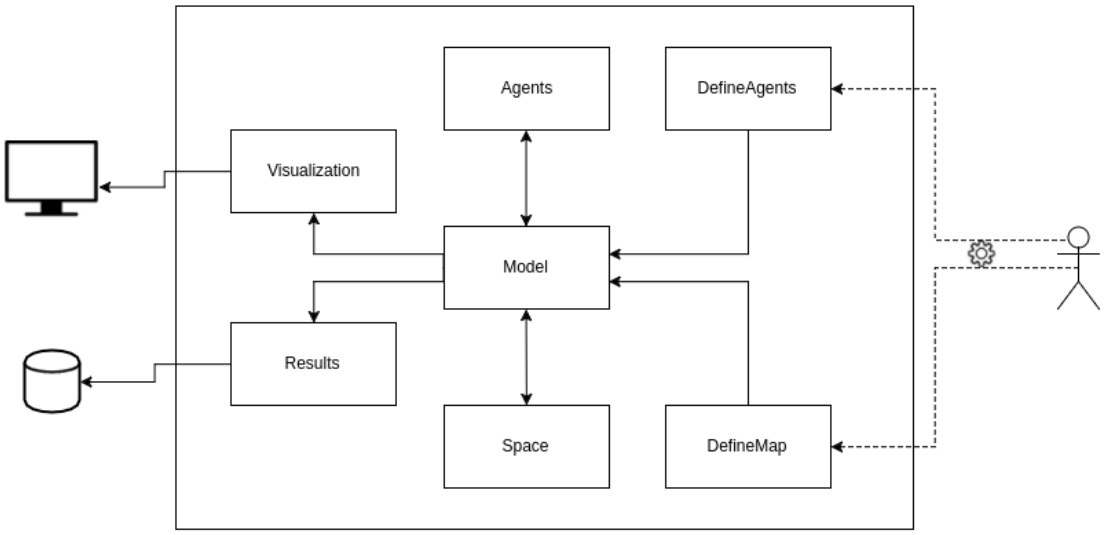


Figure 2.2: Soba

This project uses Soba to implement some functionalities for example transitions,which is a lightweight, object-oriented state machine implementation in Python. This package allows us to define states and change between them depends of the time or others variables, such as there is a fire and everybody are on the emergency state. Also we have used this architecture to model our agents and our map creating a new grid class, BuildingGrid which extends MultiGrid class of Mesa and new agents classes.

2.4 Ramen

Ramen [15] is a new project, developed in ETSIT's GSI DIT department, which consists in the design and development of a tool that allows users to visualize a 3D environment in which there are people interacting with each other. This tool provides us a helpful way to analyze different situations to obtain valuable information. With this objective, ThreeJS has been used, which is a Javascript library to visualize and to animate 3D environments.

Ramen is created because in the ETSIT's GSI DIT department there is a software called UbikSim, for visualizing social simulations in a 3D environment, that it is obsolete because it is based on SweetHome3D, which in turn is based on Java3D. Nowadays Java3D is outdated and abandoned.

In our project we will use Ramen to visualize the simulation of the different scenarios that we will analyze to take conclusion about which policy is better.

Ramen receives data from our project, CESBA, this information needs a specific format to gets it and represents it on Ramen.

Here we can see how the format is, where '"type":1' represents the type of coordinates and we have to define in each step where all the agents are and if they are going to move, where they will go and how many steps are needed to get that position:

Listing 2.1: "JSON Ramen"

```
{
  "type": 1,
  "steps": [
  [
  {
    "agent": 0,
    "position": "x,y"
  },
  {
    "agent": 1,
    "position": "x,y"
  }
  ],
  [
  {
    "agent": 0,
    "moveTo": "x,y",
```

```
      "toStep": 15
    }
    ]
    ]
}
```

All of this information is given to Ramen in a JSON which has this structure.

Below the image represents how GSI Laboratory would be seen using Ramen.

Figure 2.3: Ramen

CHAPTER 3

Architecture

3.1 Overview

In this chapter, the architecture of this project will be explained, both design and implementation phases details. First the global system will be explained and after that, each module details will be given.

Figure 3.1 shows how is divided the whole system, which is composed by two modules described below. For this project the main module is Mesa, divided into several submodules.

- **Mesa**: this module is the responsible for the entire simulation, defining all the parameters to analyze and saving all the data obtained. This module is divided into two submodules:
  - Social Module: Affiliation and old people models.
  - Evacuation Module: Routing strategies, Policies and metrics.

  As we can see in 3.1, also these submodules are divided into several submodules, all of them will be explained in detail later.

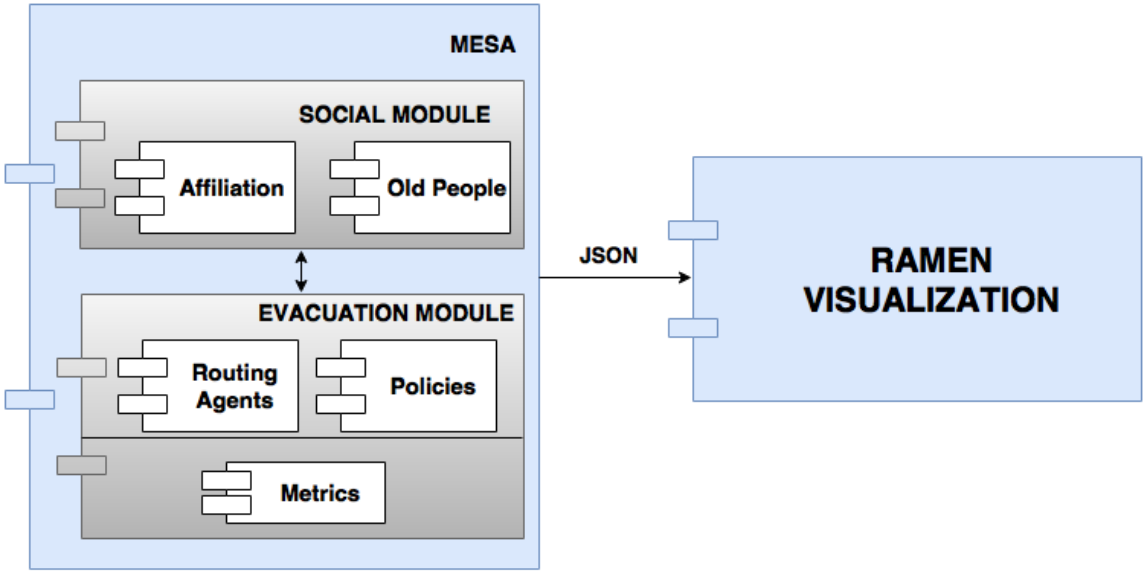


Figure 3.1: Architecture

- **Ramen Visualization**: this module supports 3D visualization to the project, so it allows users to analyze the simulation watching it in a 3D environment.

This project is mainly focused on the Mesa module and also in a minor way the Ramen Visualization because we only use this last module to visualize the simulations, whereas the first one creates and develops the scenarios and models the behaviors of the agents.

3.1.1 Python Architecture

Figure 3.2 shows how is the python architecture and the folder organization.

On one hand the project has several folders, the figure shows the main ones.

- *Agents*: contains the files which model the agents. 'fire.py', 'occupant.py' and 'behaviourMarkov.py'. 'occupancy.py' make used of 'behaviourMarkov.py' to change the agents' state.
- *Space*: includes the files required to define objects related with the map, for example, 'room.py', 'door.py' and 'wall.py'. Also the file 'aStar.py' is in this folder. This file includes the A* algorithm and the methods used to evacuate the building according to the differents policies.
- *Configuration*: contains the files which model the scenario. 'BuildingGrid.py' uses 'defineMap.py' to create the walls and the doors. 'defineOccupancy.py' is used by

the model to create the agents and know how they behave depending on the time. And finally 'settings.py' is used to set the valor to several variables of the model, for example the fire hour.

- *Visualization*: includes the files in charge of the visualization in Mesa.

The main files will be introduced in the following pages at the same time that the CESBA architecture is explained. Finally when these files where presented there is a explication of the entire operation of CESBA and how each file is connected with other one.

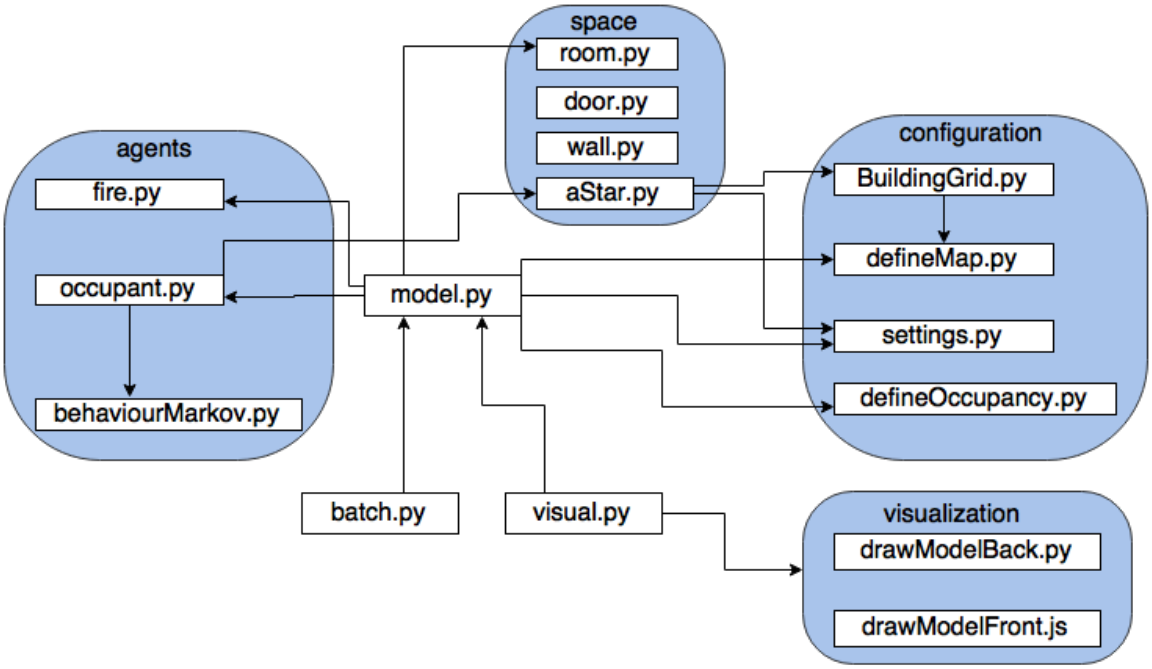


Figure 3.2: Diagram of Python Architecture

'batch.py' and 'visual.py' are the files used to simulate. The first one does it in background and the second one uses the visualization files to see the simulation step by step.

3.2 Mesa module

We use Mesa, described in chapter 2.2, as the main tool to design and implement our project. This module is connected with Ramen Visualization Module 3.3 by a JSON file which contains all needed information to represent the simulation with Ramen.

Mesa module includes all code that we have used to develop this project. Although our project uses and extends the Mesa framework for the evacuation scenario, we have created new classes to model our agents, their behavior and the simulation map.

There are two main submodules, which includes another submodules. Now these modules are introduced.

3.2.1 Social Module

This is the most important module of this project, because it is the responsible of agents' behavior. This module deals with how agents move into the map, their velocity and how they react when there is a emergency.

In this part we are going to introduce how agents have been modeled. The evaluation of the module has been carried out in the ETSIT scenario. In particular, we have modeled the 2nd floor of Building B. So this scenario has offices, classes and laboratories. Later this scenario will be explained in detail 3.2.2.

First of all we have modeled normal agents, these agents have a normal behavior. They move in the map according to a state machine. This state machine works with transitions, it works as Markov Chains, which are probabilistic processes which depend only on the previous state and not on the complete history. So there is a method that moves agents between several states depending on the hour of the day and a Markov matrix which has different probabilities depending of the previous state. This method returns a new matrix that is ready to move the agent to a new state. When a agent changes his state he sets the new position where he has to go using A* algorithm, which is developed in next subsection 3.2.2.

Following with agents, we have created three types of them according to the university scenario. Agents have been modeled as professors, students and researchers, since they have different routines along the day. For every agent type, a probabilistic state machine has been defined, based on the final work of Eduardo Merino [14]. Also here we define the position and the number of agents there are in each room depending on the state.

It is important to mention that there is only one class of agent, named Occupant and

this class distinguishes between the three types named above. In addition, an agent has been defined for modeling the Fire, but this we will see later.

Below they are introduced the different type of agents and their states:

- Professors

Listing 3.1: "statesProfessors"

```
statesProfessors = [
{'name':'leave', 'position': 'outBuilding'},
{'name':'working in my office', 'position': {'Office1': 2, 'Office2':
    3, 'Office3': 3, 'Office4': 4, 'Office5': 3, 'Office6': 1, '
    Office7': 3, 'Office8': 5, 'Office9': 2, 'Office10': 4, 'Office11
    ': 3, 'Office12': 3, 'Office13': 2, 'Office14': 2}},
{'name':'having a break', 'position': {'Hall': 16}},
{'name':'at restroom', 'position':'Restroom'},
{'name':'in a meeting', 'position': {'Class4':10 ,'Lab10':10,'Lab12
    ':10,'Lab16':10}},
{'name':'lunch', 'position': 'outBuilding'},
{'name':'giving class', 'position':{'outBuilding': 24, 'Class1': 4, '
    Class2': 4, 'Class3': 4, 'Class4': 4}},
{'name':'emergency', 'position':'outBuilding'}
]
```

- Researchers

Listing 3.2: "statesResearchers"

```
statesResearchers = [
{'name':'leave', 'position': 'outBuilding'},
{'name':'working in my laboratory', 'position': {'Lab1': 1, 'Lab2': 1,
     'Lab3': 1, 'Lab4': 1, 'Lab5': 1, 'Lab6': 3, 'Lab7': 2, 'Lab8': 2,
     'Lab9': 2, 'Lab11': 3, 'Lab13': 3, 'Lab14': 3, 'Lab15': 5, 'Lab17
    ': 4, 'Lab18': 2, 'Lab19': 3, 'Lab20': 3}},
{'name':'having a break', 'position': {'Hall': 26}},
{'name':'at restroom', 'position':'Restroom'},
{'name':'lunch', 'position': 'outBuilding'},
{'name':'emergency', 'position':'outBuilding'}
]
```

- Students

Listing 3.3: "statesStudents"

```
statesStudents = [
{'name':'leave', 'position': 'outBuilding'}, #initial state(the first)
{'name':'in class', 'position': {'Class1': 40, 'Class2': 35, 'Class3':
    30, 'Class4': 20}},
{'name':'emergency', 'position':'outBuilding'}
]
```

To set the agents there is a file (defineOccupancy.py) which is in charge of creating the array 'occupancy_json' that will contain for each type of agents (professor, research and student) a JSON. This file is initialized in the main file, and the each JSON mentioned before contains: type, number of agents, states, markov matrix and the schedule of each type of agents.

Listing 3.4: "Example of Students JSON inserted in occupancy_json"

```
occupancyStudents = {'type':'students' , 'N':NStudents, 'states':
    statesStudents, 'matrix': markov_matrixStudents, 'lifeWay':
    controlBehaviourStudents}

occupancy_json.append(occupancyStudents)
```

As we can see in the last listing, all parameters required are defined. 'States' is an array like it has been presented above, 'matrix' an 'liveWay' are like in the next example:

Listing 3.5: "Example of parameters inserted in occupancy_json"

```
markov_matrixStudients = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]

controlBehaviourStudients = {'arriveTime': 15.55, 'leaveWorkTime': 18.05}
```

Then when agents are created the main file (Model) go through 'occupancy_json' and set type, number and place to the agents. Also 'defineOccupancy.py' has two methods that control the transitions (changing the probabilities) between states and the time of permanence in each state by means of the time.

Listing 3.6: "Example of method that control the transitions"

```
elif agent.type == 'studients':
  if time > configuration.settings.activationFire:
    new_matrix = [[0, 0, 100], [0, 0, 100], [0, 0, 100]]
  elif time < behaviour['arriveTime']:
    new_matrix = [[0, 0, 0], [0, 0, 0], [0, 0, 0]]
  elif (behaviour['leaveWorkTime']) >= time >= behaviour['arriveTime']:
    new_matrix =[[30, 70, 0], [0, 100, 0], [0, 0, 0]]
  elif time >= (behaviour['leaveWorkTime']):
    new_matrix = [[100, 0, 0], [70, 30, 0], [0, 0, 0]]
  return new_matrix
```

Listing 3.7: "Example of method that control the time activity"

```
if time < behaviour['arriveTime']:
  timeActivity_matrix = [13.0, 0, 0]
elif (behaviour['leaveWorkTime']) >= time >= behaviour['arriveTime']:
  timeActivity_matrix = [0.01, 2, 0]
elif time >= behaviour['leaveWorkTime']:
  timeActivity_matrix = [10, 0.01, 0]
return timeActivity_matrixs...
```

Then in these extracts there are two examples of how transitions work. In case of other type of agent the code is similar but it has more states as we can see in 3.1 or 3.2 listing.

Therefore there are normal agents and they move in the map according their states but when an emergency happens they react and try to go out of the building as soon as possible. But according to [16], during the impact time the first minutes when a emergency occurs, there are:

- 10-25 % of people keep together and be in calm, they study an action plan and possibilities.
- 75 % manifest disorderly behavior, bewilderment.
- 10-25 % confusion, anxiety, paralysis, hysterical screams and panic.

So we have modeled the agents in the worse case, they take sometime to react. When they notice what it happens they try to go out. For this goal we have implemented several ways to evacuate the building, they were seen in 3.2.2 subsection.

To this standard case we have added two submodules like it is shown in Fig. 3.1, affiliation and old people. These two submodules help us to model agents in a different way when they have to evacuate the building. They will be able to simulate together or separated.

3.2.1.1 Affiliation

On one hand, affiliation model has been chosen according to [1] because it is one of the most important crowd behavior together with different kinds of reactions when a emergency happens. This project [1] says that when there is a evacuation, families try to exit together by the same exit, meet at a point or someone take care of the youngest member and the rest leave the building. So in this project we have chosen the last one.

When agents are created they could belong to a family, each family have one child and one parent. Not every agents belong to a family only a 20% have been modeled in this way to analyze the differents results between agents with family and individual agents.

On one hand in the simulation each member of the family has a normal behavior according to the state machine but when the emergency happens each member of the family try to leave the building using the same way as his family whereas the child of each family stay stopped in his position waiting to his parent.This parent mentioned previously look for the child and go to the child's position. When they are together try to leave the building in the same way as their family did. On the other hand individual agents exit the building in their own way.

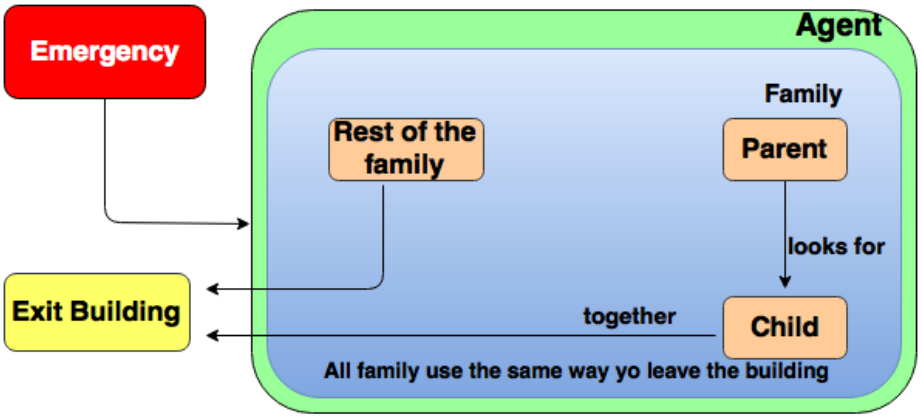


Figure 3.3: Affiliation

3.2.1.2 Old people

On the other hand there is another submodule which models people with mobility disabilities, such as old people. These characteristic allow us treat this kind of agents in a different way because their velocity is slower than normal agents and they could take more time to react when there is a emergency.

This submodule is implemented because we think it is important to analyze how old people manage these cases. Also in this way we can take conclusions about what is the best policy to old people.

3.2.1.3 Agent Operation

Once the social module has been introduced, where agents are defined. Now it is time to explain how agent works, so this section is dedicated to introduce in detail the agent operation. It will be easier if we use the following figure to support our explication.

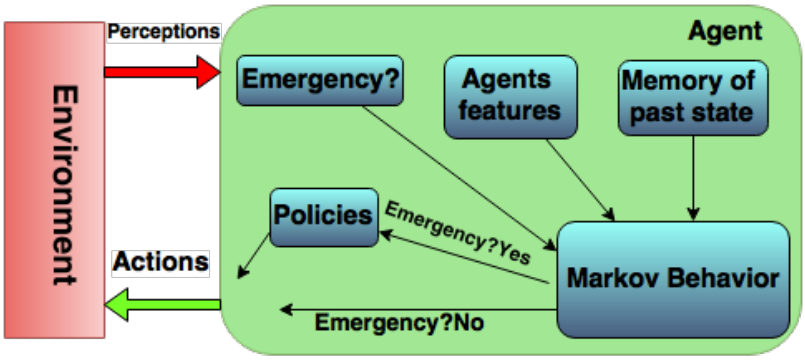


Figure 3.4: Agent Operation

First of all, agents are created with differents features (affiliation, they belongs to a family; or old people, so they move slower than normal agents), also they could be created without any of these characteristics.

Then the simulation starts and agents begin to move depending on the time, for example if it is nine o'clock many of them will go to their work place. When agents have been initialized depends on the time they change their states because of Markov Behavior, which knows what was the last state and change to the next if it is the moment.

As it was explained before in Listing 3.1 each state has certain positions(rooms), so if an agent modifies his state, he will choose one of these rooms, which are formed by several cells, and in this room he sets one cell, where there is not another agent. Once there is a goal,

agent will calculate the way to go there without collide with walls, with the A* algorithm. Also it is important to mention that while agents go towards their goal positions, if there is another agent they do not collide. The conflict has been solved by waiting until the other agent is moved. Figure 3.5 shows how it the map created to simulate.

Just explained the normal case, now it is the moment of emergency situation. When environment notices that a fire happens then environment warns agents. In this moment fire alarm rings, agents take some time to react and start to get out the building when their state change to 'emergency'. Before agents know where they have to go in each state but now in 'emergency' state there are several policies to evacuate the building which will be introduced in Sect. 3.2.2.2. If it is been simulated family evacuations, each family chooses one exit and everybody uses that exit. These policies mentioned previously are chosen by us to obtain and analyze all data. With all the information and simulating each policy with different agents' features we will be able to make conclusions about which policy is the best.

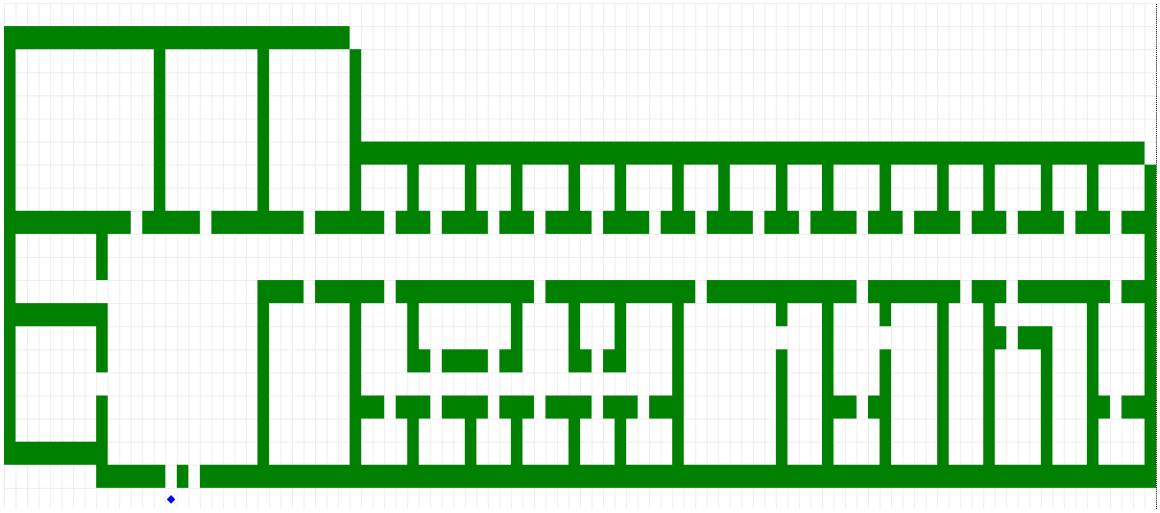


Figure 3.5: Example of how map is seen with Mesa

3.2.2 Evacuation module

Evacuation module is where everything related with environment is defined. The model class includes all the variables and methods to the proper functioning of the project. Aforesaid class imports the variables and methods of other classes that they are required.

The following pages explain in detail how this evacuation model works, with this purpose three submodules will be introduced.

3.2.2.1 Routing

It is formed by several files which make possible represent the map and calculate the path for the agents' movement. Two files are the most important to represent the map, shown in Fig. 3.5, these are 'defineMap.py' and 'BuildingGrid.py'.

Using Ramen it has been created several JSONs which contains all the information it is required to represent the map. All this data is in 'defineMap.py' file which has all jsons:

- **wall_json:** includes each wall and his corner. The corner has an 'id' how it is shown in 3.8.
- **corners_json:** includes the 'ramen' coordinates of each corner 3.9.
- **doors_json:** includes the 'ramen' coordinates of each door.
- **rooms_json:** includes the name, door and possibles 'x' and 'y' coordinates of each room 3.10.

Listing 3.8: "Example of a little part of walls_json"

```
walls_json = [
{
"corner1": "fa8fc859-c1d9-c3ec-53c0-f4cb5c7561dc",
"corner2": "20175766-b0fe-6de5-0770-338996573530"
},
```

Listing 3.9: "Example of a little part of corner_json"

```
corners_json = [
{"fa8fc859-c1d9-c3ec-53c0-f4cb5c7561dc": {
"x": -169.03700000000012,
"y": -308.35600000000005
}},
```

Listing 3.10: "Example of a little part of rooms_json"

```
rooms_json = [

{'name':'Office1', 'door' : (33,14), 'x': {31,32,33,34}, 'y':{15,16}},
{'name':'Office2', 'door' : (37,14), 'x': {36,37,38,39}, 'y':{15,16}},
```

Defining JSONs, it has been mentioned 'ramen' coordinates because when the walls_json and corners_json are gotten they are given with the coordinates used in RAMEN 2.4. So it is necessary to make a conversion and obtain 'mesa' coordinates to define position with a integer number 'x' and 'y'. For that reason it was moved the origin of the 'ramen' coordinates system to the origin of 'mesa' coordinates system(position(0,0)). Also a factor has been used to do the conversion with the objective that each cell in the map represents one meter.

Then, it is time to introduce 'BuildingGrid.py' where these JSONs are used. This file extends 'MultiGrid' class of Mesa which used to move and set the agents in the grid. 'BuildingGrid.py' is created to implement the methods which creates walls and doors.

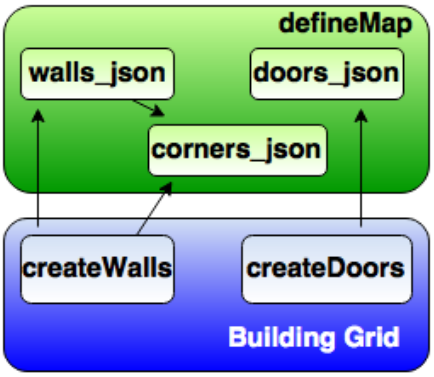


Figure 3.6: Explanation of BuildingGrid

Now each method will be explained using the Fig. 3.6. The module 'createWalls' defines two arrays which are imported from 'defineMap'. These arrays contain walls and corners in JSON format. In addition, an additional array (self.Walls) is created to contain the wall positions.

Once those arrays have been declared the method starts to iterate through walls_json and get one wall. For every wall, the coordinates are obtained from the array corners_json. Then, the method converts between 'ramen' and 'mesa' coordinates. In case the values of xs and ys are the same, this means a corner. If 'x' of corner1 is equal than corner2's 'x' then 'x' is the permanent value and the different values of 'y' between 'y' of corner1 and 'y' of corner2 are iterated through and saved as Wall(x,y) object in 'self.Walls'. It will be the same process if values of xs are differents and ys are equals. When the walls_json is gone through, we would have 'self.Walls' array with all the positions where the is a wall.

Furthermore, the implementation of 'createDoors' is similar to 'createWalls'. First, it iterates through doors_json and the conversion to 'mesa' coordinates is made, after that in 'self.Doors' it is saved object 'Door' which has 'x' and 'y' position. Finally we iterate through both arrays, 'self.Doors' and 'self.Walls', and from the last one we remove where there is a door.

With these two methods the map are defined and they are called when the Model is initialized together with a method which belongs to the Model and use the rooms_json to creates another array with 'Room' objects array. These 'Room' objects contain, as shown in Listing 3.10, name, door position, and possibles 'x' and 'y' which are in the room.

Although RAMEN 2.4 makes the visualization, before his integration CESBA used the Mesa visualization. This visualization [4] is done in a browser window, using JavaScript to draw the different things being visualized at each step of the model. To do this, Mesa launches a small web server, which runs the model, turns each step into a JSON object (essentially, structured plain text) and sends those steps to the browser.

A visualization is built up of a few different modules: for example, a module for drawing agents on a grid, and another one for drawing a chart of some variable. Each module has a Python part, which runs on the server and turns a model state into JSON data; and a JavaScript side, which takes that JSON data and draws it in the browser window. Mesa comes with a few modules built in, and let you add your own as well.

Then, MESA visualization has been configured to use these modules. The files DrawModelFront.js and DrawModelBack.py contains the methods to draw the map. Being the first one the JavaScript side.

Just explained the part of the map it is time to introduce other type of agent in addition to Occupant agent. This new class of agent is the fire, this agent is very simple. Fire agents are created when the time is equal to global variable 'activationFire', this variable is created randomly when Model is initialized. Once that time exceeds the fire hour define by 'activationFire' then the first fire agent is created in a randomly room. From that point every 30 seconds the fire spreads in all possibles directions if there are not walls. Creating new fire agents in all neighbor position, the next time that this process would be realized the fire would spread from the last fire positions created.

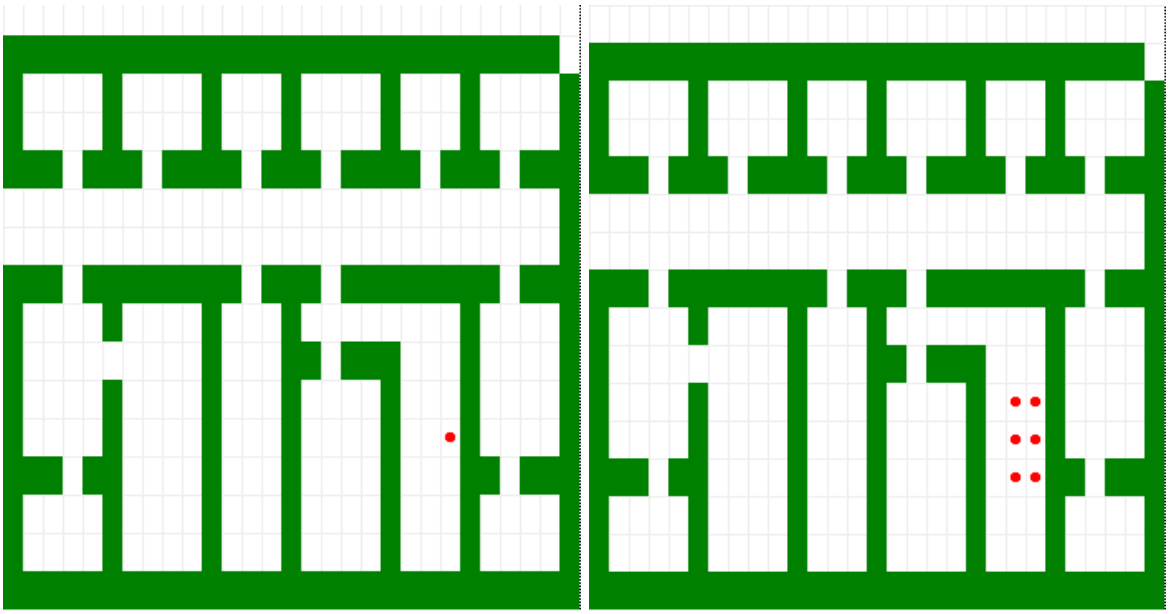


Figure 3.7: Example of fire spreading

Finally, the Routing module contains an additional file, 'aStar.py', which implements the A Star algorithm [17]. This algorithm will be used by agents the calculate the path between positions. The main method is 'getPath(model, start, finish)', it is in charged to calculating the best path between start position and finish position. This path is calculated with the A* algorithm, the pseudocode is shown below in Listing 3.11. In our case the cost(g) between one node to other node is one and 'h' cost is zero. When the algorithm has finished, it returns the path and the path's cost.

Listing 3.11: "Pseudocode of A* algorithm"

```
  initialize the open list
  initialize the closed list
  put the starting node on the open list (you can leave its f at zero)

  while the open list is not empty
    find the node with the least f on the open list, call it "q"
    pop q off the open list
    generate q's 8 successors and set their parents to q
    for each successor
      if successor is the goal, stop the search
      successor.g = q.g + distance between successor and q
    successor.h = distance from goal to successor
    successor.f = successor.g + successor.h

    if a node with the same position as successor is in the OPEN list \
      which has a lower f than successor, skip this successor
    if a node with the same position as successor is in the CLOSED list \
      which has a lower f than successor, skip this successor
    otherwise, add the node to the open list
  end
  push q on the closed list
end
```

Agents' movement has been implemented with the condition that if the position or cell is it occupied by another agent, the agent will wait until the cell is empty. But it would happen that two agents are put face to face, then these agents never move again because the cell they want to occupy is not empty. For that case we have used a collision mechanism resolution, which consists in when the agents are face to face one of them try to move to a neighbor cell if this cell is empty and recalculate the goal path. Also if agents are blocked during three steps then they will try to reach a neighbor cell if it is possible and recalculate the goal path.

3.2.2.2 Policies

Now in this part the differents policies are going to be introduce. These are the main ones that this project has used to simulate in the differents cases of study.

Before beginning to use the policies when agents' state was 'emergency' they left the building going to the principal gate and nobody uses the 'building C' gate. In the map this last gate it is also another room but in the real blueprint it is a door which is communicated with a corridor of the adjacent building. This policy could be 'go to the familiar gate' considering that it is the mainly gate in this building. How it is said in [1], familiar exits, routes and places are used by people to evacuate before using new routes. This make the evacuation slower so this project try to analyze other policies. These policies are:

- *Nearest gate*: It consists in go to the nearest gate when the emergency occurs. This technique could be the fastest way to evacuate the building but it is not the safest considering that the fire could be in the way to this exit. If the fire progresses quickly and it is near from one of the exit, this fact could be dangerous if the agent tries to evacuate in that direction.

  In this way, the bottle necks are avoided because the agents are distributed between the two exits and the evacuation takes less time. Although it could be dangerous how it was mentioned above because of the fire position.

  This policy is implemented in the same file of A* algorithm considering that it consists in a method which returns the path to the nearest exit. This method only needs one parameter, the agent position, and with this parameter calculates the distance to the main exit and 'Building C gate'. The exit with the low distance will be the goal. Agents will follow that path calculated with the A* algorithm.

- *Safest gate*: as its name suggests, this policy makes the agents leave the building by the farthest exit in relation of the initial position of the fire. This path to this safest exit could be dangerous if this path contains the fire position. Then when this policy has been implemented the method looks for the initial position of the fire and if that position is in the path the agent changes the gate and goes to another one.

  Also it is developed in the file of A* algorithm because it uses the A* to calculate the distance to the fire and how it was mentioned previously if fire's initial position is not in the path to the exit, the method will return the farthest exit. This could have a problem and it is the fire spreading because although fire's initial position will not in the path when fire is spreading could be that when the agents reach certain cell, this cell is occupied by the fire.

When an emergency happens the agents decide policy for leaving the building when the state changes. Also it is in that moment when they calculate the path to the new position defined by the new state although that state is different from the emergency ones.

3.2.2.3 Metrics

Metrics involve the parameters analyzed to obtain results and draw conclusions. In this type of simulation there are many interesting parameters, fire's velocity and position, where and why the fire starts, agents positions, reaction time. But this project is focus on:

- *Exit time*: This time covers the time since agents change their state to 'emergency' until they leave the building. This metric is very important because it shows us which policy is the best. To calculate the time agents have a variable which registers the time when agents change the state to 'emergency' and another variable to register when variable they reach the exit goal. The difference between both of them will be the exit time.

- *Number of burned agents*: how its own name indicates this metric calculates the number of agents that have died during the simulation. They will die if they reach a cell which also contains fire or backwards, fire reach a cell which contains an agent. Then to estimate this number in each step of the 'occupant.py' file there is a method which checks if the agent is burned.

  The method mentioned previously uses the parameter given (agent's position) and checks if this position is equal to one of the cells which contain fire.

With these main metrics we can analyze and draw conclusions about which are the best or the worst policy in several cases. For example when agents are old people or they belongs to a family. These cases will be explained detailedly in the following chapter 4.

3.3 Ramen Visualization Module

This module is in charge of the visualization in Ramen. First of all, when the simulation has finished, we obtain a JSON. This JSON contains all the data about the simulation, agents's position, their movements, the fire progress, etc.

This information needs to follow the following structure:

Listing 3.12: "JSON Ramen"

```
{
"type": 1,
"steps": [
[
{
"agent": 0,
"position": "x,y"
},
{
"agent": 1,
"position": "x,y"
}
],
[
{
"agent": 0,
"moveTo": "x,y",
"toStep": 15
}
]
]
}
```

When the data is with this structure then it is used by the Model Generator as Fig. 3.8 shows. Model Generator analyses all this information and creates the corresponding objects of the Ramen classes Camera, Floorplan, Agents, Scene and Items. Scene is the main one that uses the other to represent all the data and creates the visualization where there are two types of controls: camera controls to see the scene by differents angles and the simulation controls to stop and start the simulation.

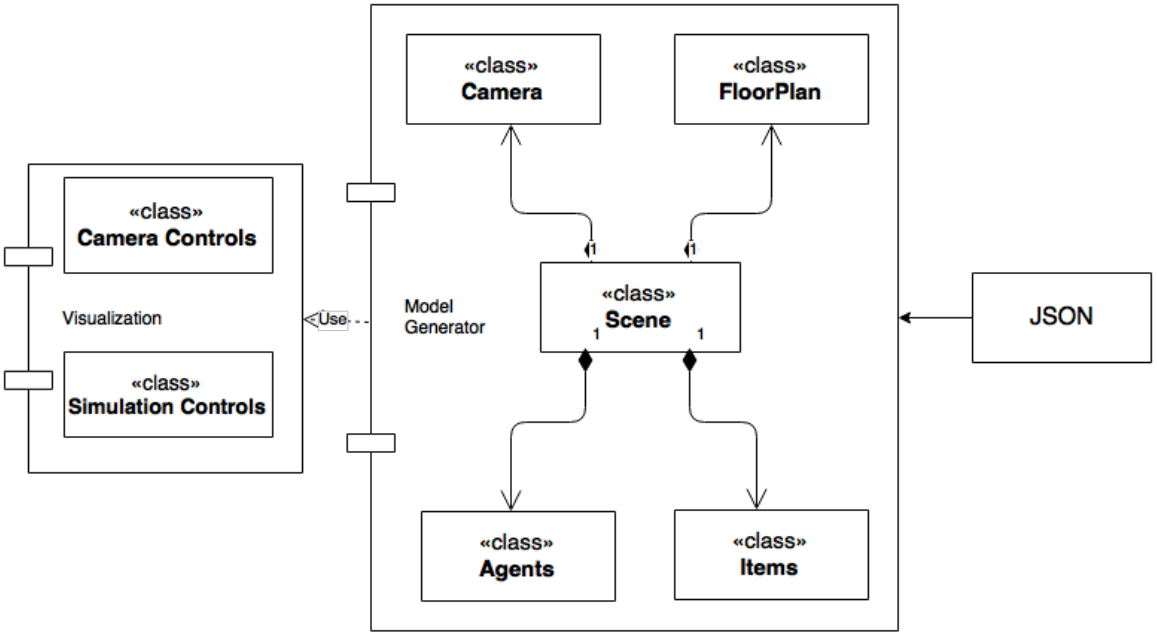


Figure 3.8: Ramen Architecture

3.4 Operating Mode

As it was mentioned in Sect. 3.1.1 when all the modules are introduced it is time to explain detailedly how the project works and the connections between the files.

Before starting it is important to remind the project architecture shown in Fig. 3.2.

For this explication we are going to simulate with 'visual.py' file which helps us to see step by step the simulation. This file calls the model ('model.py') and it is the model object is in charge of creating the Scene object.

First of all model initializes the configuration files and some control variables. After that the model creates the rooms, sets the agents and calls the methods of 'BuildingGrid.py' to create the map. Before this, it sets the agents because when agents are initialized they are placed into the map. As we have mentioned previously 'BuildingGrid.py' uses

'defineMap.py' to create walls and doors. Also the model calls 'defineMap.py' to create the rooms and 'defineOccupancy.py' to create the agents.

When everything is created the model starts running and by each step of the model there is one step by each agent. In each step of the model we check if it is the fire time, when it is this time the model creates the fire agent and place it into the map. If we have passed this time each 30 seconds the fire advances to the neighbor cells and in this way the fire gradually spreads. Here in this model step we also check if the fire time was one minute ago and the model changes the agents' state to the emergency ones. This minute is the reaction time since the alarms sounds and the evacuation starts.

For each model step there is one step by each agent then we are going to explain how 'occupant.py' works. First of all when an agent is created there is initialization where it is defined the states, the position by state and several control variables. Each time when the state changes it is called the method 'start_activity' which is in charge of deciding where the agents have to go depending on the state. Each state has one position, and calculates the path using 'aStar.py'. It is in this method where in case of emergency agents choose one of the differents policies.

In each agent step we check if it is time to change the state, this file uses 'behaviour.py' to this objective. If it is not time to change the state, agent has several options depending of the situation:

- Normal situation: if agent has not reached the goal then tries to move to the next cell if this cell is empty. In case of agent has reached the goal, the time in this state starts to decrease and if this time by state is over the state machine is called to change the state.
- Fire situation: in this case first of all we check if the agent has been reached by the fire and tries to move like in the normal situation

It is important to mention that each step represents one second. This time by step is defined in 'settings.py', which is imported by 'aStar.py'.

The simulation finishes when all the agents are out of the building and we have stored all the data by each model step to know how it is the simulation in each moment.

CHAPTER 4

Case study

In this chapter it will be suggested five research questions and with all data obtained they will be answered.

Many simulations have been made to try to get a reply. These simulations will be differents features depending on the research questions.

4.1 RQ1: Does emergency time affect the evacuation?

To answer this question we have simulated in three differents hours with these features:

- Fire position: Office 6, (54,16), this parameter did not change between simulations.
- Fire hour: 10:20, 12:00, 15:00

We have obtained the following information:

- 10:20: 22 agents in the building and 1 burned
- 12:00: 41 agents in the building and 2 burned

- 15:00: 10 agents in the building and 0 burned

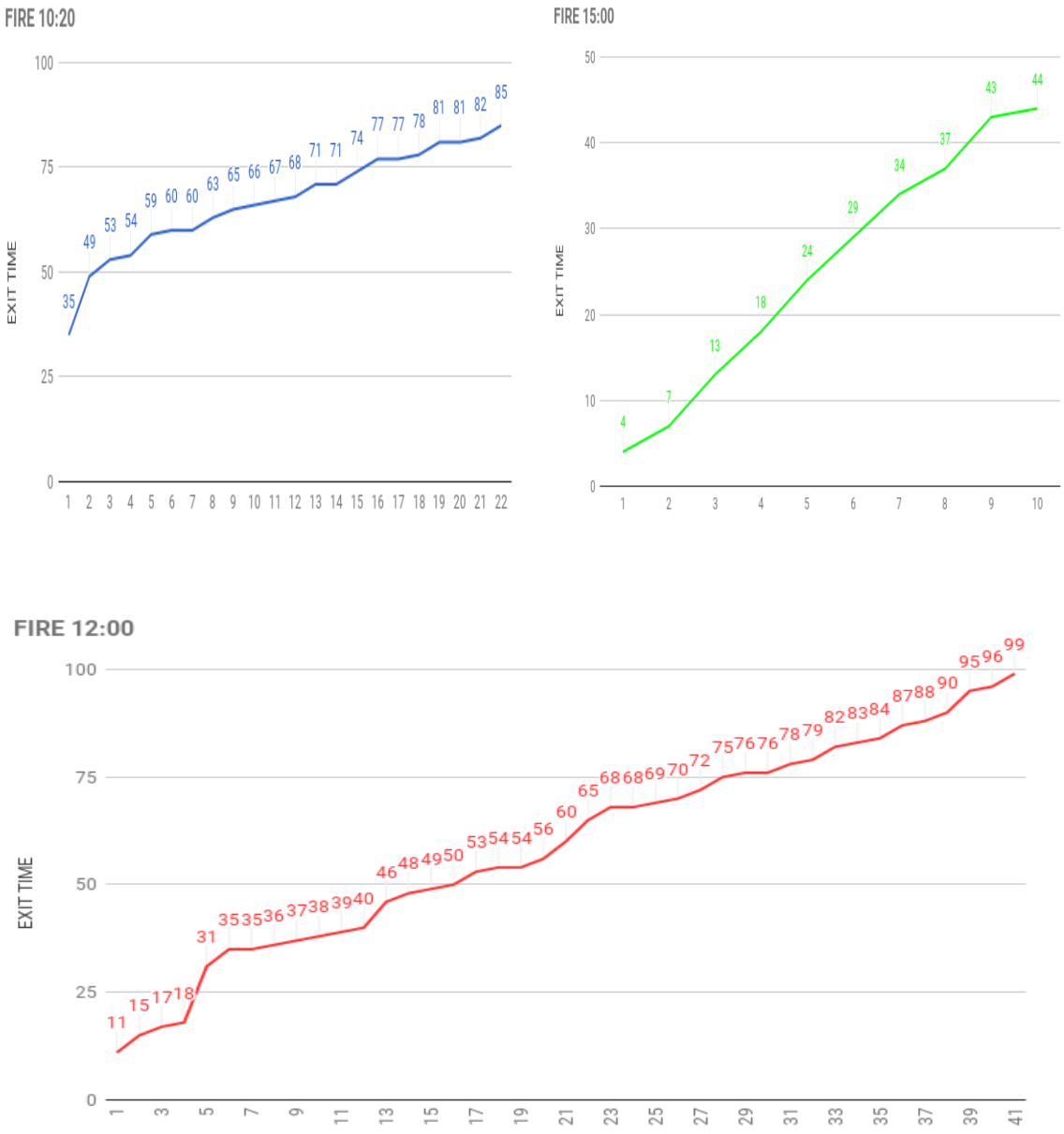


Figure 4.1: Graphic of data obtained

We can see the differents results in the charts, being 'x' axis each agent and 'y' axis their exit time. The number of dead agents depends on the time the fire occurs. Considering this data we can answer with total certain that the emergency hour is very important. Depending of the hour there will be more or less agents in the building. Being the worst hour at midday and the best hour at lunch time in these cases of study.

4.2 RQ2: What policy is more effective saving people?

To resolve this research question we are going to simulate with the following data:

- Fire hour: 12:00
- Fire position: Office 6, (56,15). This position changes in relation to research question 1 because in the previous position the fire needed more time to spread and reach the corridor.

In this case three simulations have been made because there are three differents policies: familiar exit, nearest exit and safest exit. The following chart shows the number of agents alive and died in each situation. The total number differs due to the probabilistic model of the state machine.

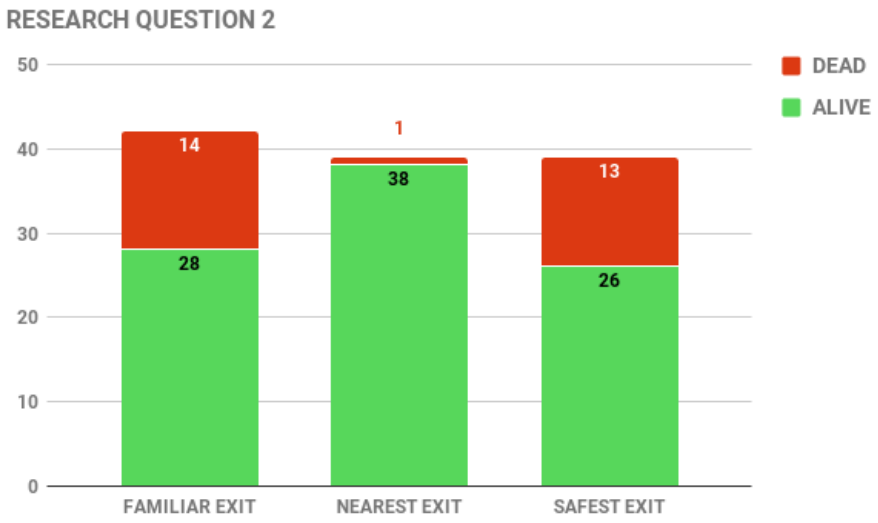


Figure 4.2: Number of agents

Observing the chart we can conclude that the best policy to save people is 'Nearest exit' considering that only one agent died in this simulation whereas in the other two simulations many agents died because the fire position is almost in the middle of the map and the agents which are far when they try to exit by the familiar exit or by the safest exit the fire reaches the corridor and blocks the agents.

The following figures shows how 'Nearest exit' policy is seen with Ramen.

Figure 4.3: Fire starts

Figure 4.4: Agents leaving the building and fire spreading

4.3 RQ3: Which policy gets the best evacuation time?

In this case study, we carry out an evaluation of which policy leads to the best evacuation time. With this objective the simulation features are the same than in RQ2, described in Sect. 4.2. But now we are going to analyze the exit time.

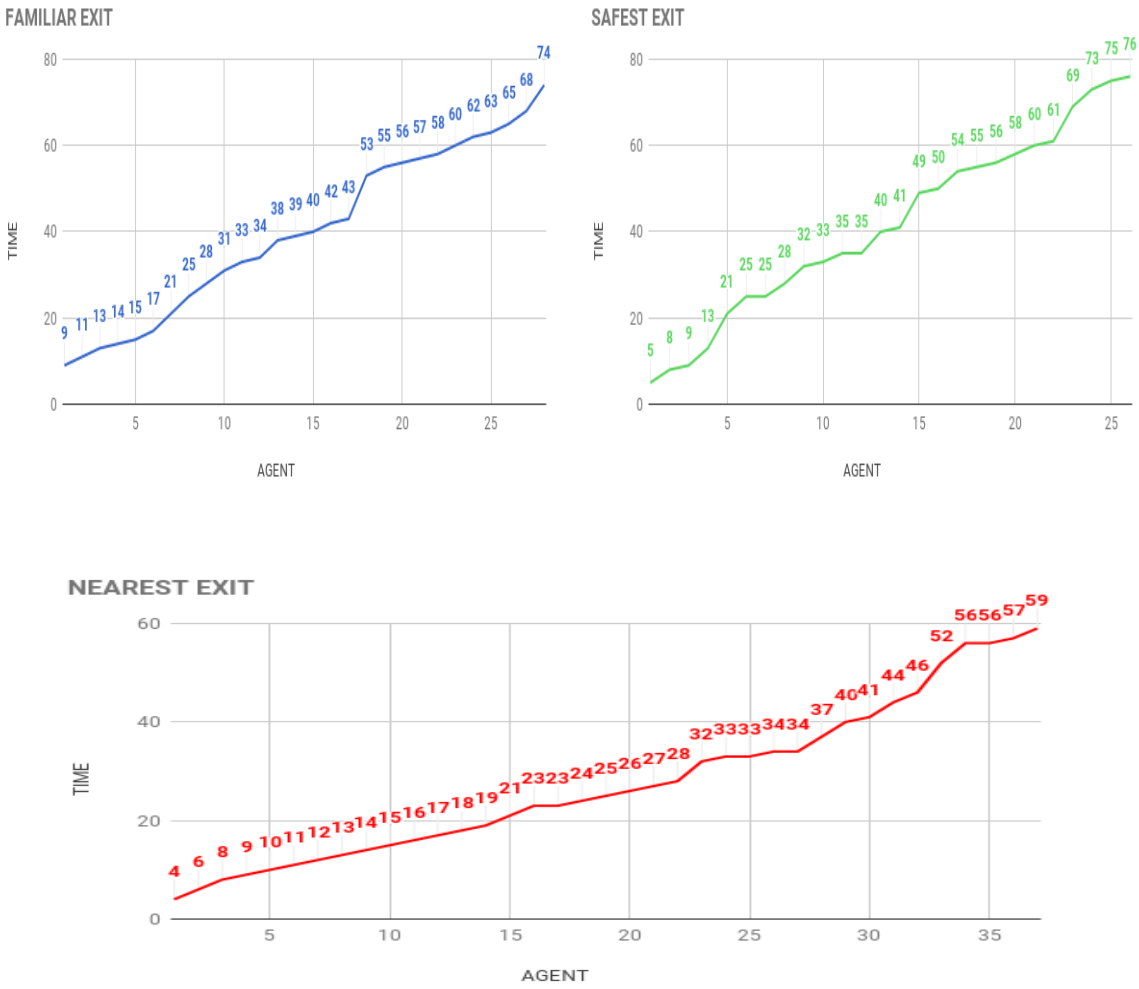


Figure 4.5: Graphic of exit times

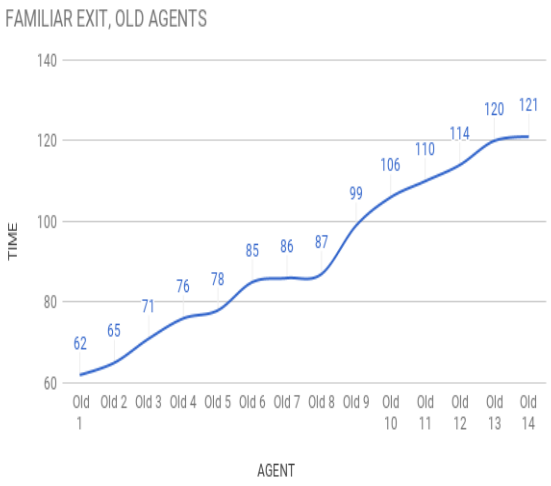
For each simulation we have obtained a different media: 'Familiar exit' 40,14 seg, 'Nearest exit' 27,65 seg and 'Safest exit' 41,77 seg.

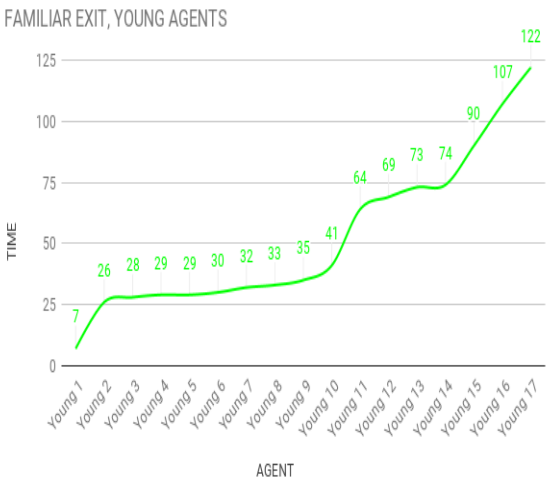
On one hand with this information we can affirm that the best policy to get the best exit time is the 'Nearest exit' policy. On the other hand the other policies have a similar exit time considering that the fire position is almost in the middle of the map how we can see in Fig. 4.4a. Thus, agents use the same exit in those cases.

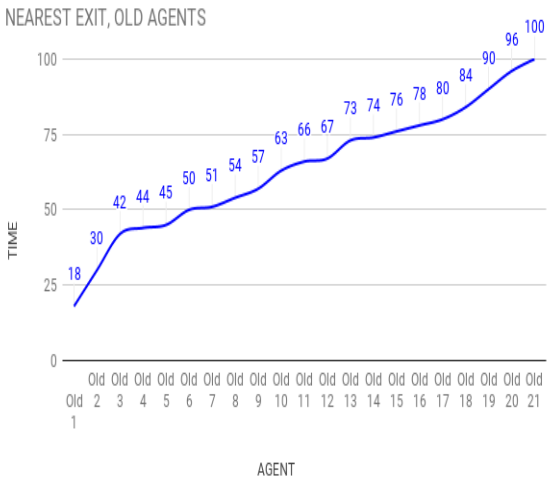
4.4 RQ4: How do mobility problems impact on the evacuation?

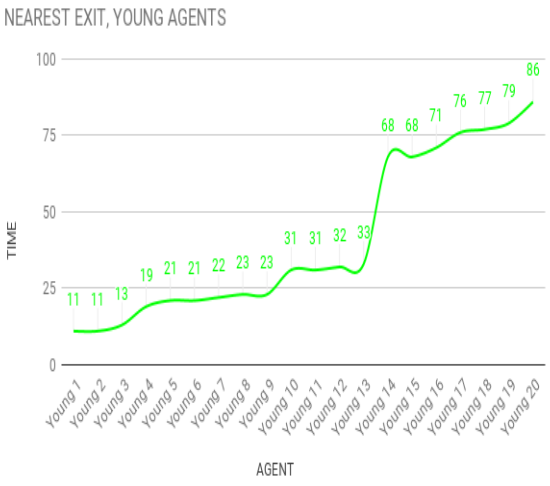
To resolve this research question we are going to simulate with the following data:

- Fire hour: 12:00
- Fire position: Office 6, (55,15).
- Agents features: In these simulations we have included some agents which are old and walk slower than the normal ones. Their velocity will be reduced by half, that is two model steps.









Also we have obtained the following medias:

- Familiar exit:
  - Old: 91,43 seg
  - Young: 52,29 seg
- Nearest exit:
  - Old: 63,71 seg
  - Young: 40,80 seg

'Safest exit' has the same results as 'Familiar exit'. As we can see in the charts old agents need more time to leave the building although some young agents are stopped by the old agents and follow them to the exit. This fact made young people more vulnerable to be burned as we can see in the following chart, where in the second simulation 'Nearest exit' two young agents died but in general old agents have more probability of die.

Therefore we can conclude that mobility problems affect directly in the time of evacuation and in the probability of be reached by the fire.

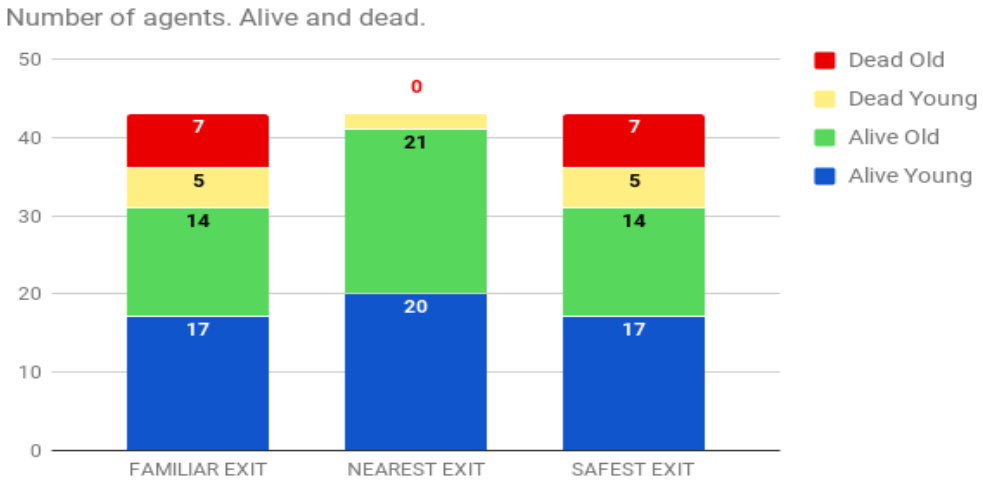


Figure 4.6: Graphic of number of agents. Alive and dead

4.5 RQ5: How do family ties affect in the evacuation?

In this case we have modeled two families which are formed by one child, one parent and another three agents. In addition to these families, it have been created agents like in other cases. In this research question we have used the following features:

- Fire hour: 10:15
- Fire position: Office 8, (64,15).
- Agents: Two families and individuals agents.

In these case we are going to analyze data without charts because this question requires a deeper analysis. Also we have simulated all the policies and we obtain similar data when we are in 'Familiar exit' and 'Safest exit', this happens because of the fire position.

On one hand we have that family 1 obtains the following exit times with the 'Familiar exit' policy:

Familiar Exit			
Family 1		Family 2	
Member	Exit time	Member	Exit time
Parent	41 seg	Parent	75 seg
Child	44 seg	Child	74 seg
Member 3	42 seg	Member 3	77 seg
Member 4	46 seg	Member 4	Dead
Member 5	51 seg	Member 5	Dead

Table 4.1: Familiar Exit Table.

On the other hand we have that family 1 obtains the following exit times with the 'Nearest exit' policy:

Nearest Exit			
Family 1		Family 2	
Member	Exit time	Member	Exit time
Parent	43 seg	Parent	Dead
Child	45 seg	Child	Dead
Member 3	41 seg	Member 3	30 seg
Member 4	46 seg	Member 4	36 seg
Member 5	51 seg	Member 5	Dead

Table 4.2: Nearest Exit Table.

With this information we can affirm that family ties affect in the evacuation time and also increase the probability of been reached by the fire because when parent goes to look for the child they delay the exit time and the fire could have advance and block the exit way. Also it is important to mention that it is very important where the fire takes place because in this simulation the fire was in one of family agent's room.

CHAPTER 5

Conclusions

This chapter explains the conclusions drawn of the development of the project, and also describes problems encountered , accomplished achievements and lines of future work.

5.1 Conclusions

In this project, an evacuation simulator has been developed, in order to define effective evacuation protocols. This simulator allows the users analyze the best policy to leave the building and also models agents with differents features as agents with mobility problems or family tie.

The whole simulation could be represent in a 3D visualization tool, Ramen. This tool helps us to see the crowd behavior and make conclusions about which is the best protocol in emergency cases.

This project is based on a number differents technologies:

- Mesa 2.2, this project uses Mesa to create agent-based models using built-in core components (such as agent schedulers and spatial grids) or customized implementations.
- Soba 2.3, the project is based on it to create the state machine that agents use to

move according to their state, in addition we have implemented the 'emergency' state to make agents leave the building.

- Ramen 2.4, how it is mentioned previously this tool allows users visualize the simulation in 3D.

As described previously, we have stated a number of research questions regarding the evacuation scenario. We summarize here our main conclusions drawn from the experiments.

Two are the most important factors, the emergency time and the fire position. The first one means how many agents are in the building, that is, at midday almost everybody are in the building however at lunch time the number of agents is very low. On the other hand we can affirm that the position fire is very important, above all if this position is near the main corridor and fire could reach the corridor in few minutes because of it the fire will block the exits. However if the fire position is in a big room and needs more time to reach the corridor, agents have enough time to leave the building.

Therefore in cases where the fire is in the middle of the map the most effective policy is 'Nearest exit' because with this policy we have obtained the best exit times and the number of dead agents is the lowest respect other policies. According to this fact, in RQ2 4.2 the more effective policy to save people have been analyzed and the exit times have been studied in RQ3 4.3. As we have seen in those cases almost every agents uses the nearest exit and do not pass near the fire. But in cases where the fire position is nearby the best way to leave the building will be going to the safest exit which one is farther to the fire.

Also we have studied if mobility problems and family ties affect to the evacuation and the result is of course yes. These factors make the evacuation time increase. In the case of mobility problems this time raises almost to the double, this supposes that the probability of been reached by the fire is greatest. Regarding family ties, two scenarios have been analyzed. When parents look for their children in the same direction than the exit path, there is not effect on the evacuation time. Nevertheless, when they have to look for them in another direction, the evacuation time is increased, as their probability of being reached by the fire.

In conclusion we can affirm that the best protocol to a successful evacuation will be when agents know where is the fire and choose the exit way that do not cross the possible zones affected by the fire.

Next sections will describe what goals were achieved by this project, what problems were encountered and also future lines of work.

5.2 Achieved goals

These are the goals achieved during the development of this project:

- **Design scenarios and policies**. This is the main goal of the project together the next one, the development and implementation of a simulator with Mesa using Python.
- **Implementation scenarios and policies which are previously designed**. This is the most difficult objective where the scenarios and policies have been developed.
- **Simulate and testing the differents situations we have studied**. Simulations which allows us test the agents, their features and their policies to leave the building. In this goal is where we have used Ramen to visualize the case of study.
- **Analyze the results of the simulation**. Using the data obtained the project generate several charts to study the differents scenarios and in this way could achieve the next goal.
- **Technical report writing**. After all, this project make a conclusion in the previous section about which the best protocol to evacuate the building.

5.3 Problems faced

The list of the problems encountered during the development of this project is shown bellow:

- Map: The map generation has been a big problem because at the beginning of the project is not clear the way to do it, with an array, using a JSON, etc... Finally we adopted Ramen to create the map. After this, the project has worked with a JSON file and has implemented the class 'BuildingGrid.py', that extends Mesa's Grid, to create the map according to Mesa coordinates which are different respect Ramen coordinates.
- Blocks: This problem occurs when there are many agents and between them they block the steps, trying to get a cell which is not empty. To manage this situations the project implements two methods, the first one follows this approach: if an agent that is blocking me wants to move to my position, I will move to a neighbor cell which is empty. The other method checks if the agent have been blocked during three step and try to move in the same way as the previous method.

- Ramen: Integration with Ramen has been difficult because Ramen needs a particular structure of the data that it is given. Therefore the project tries to extract the data of the simulations in the correct manner and gives the data the adequate structure.

5.4 Future work

Finally, this section lists the various improvements and future lines of work related to this project.

- **Add intelligent to the agents**. It would be a interesting fact if the project could make the agents recalculate the exit path when their way is blocked by the fire recalculate the exit path using another exit.
- **Add Smart Building to the scenario**. An interesting integration for the future is adding Smart Buildings where there are illuminated signals to show the agents the way to evacuate the building. Smart Building could decrease the reaction time and the exit time in evacuation situations.
- **Add new policies**. In the future it would be useful to add new policies and analyze the new ones to compare with the other policies. For example the least affluence exit to avoid bottlenecks.
- **Add new agents features**. The project would be able to model agents with differents feelings and depending on the agent's feeling uses different reaction time and policy.
- **Ramen integration in real time**. Now the project shows us the simulation in Ramen when it has finished but it would be very useful watch the simulation step by step in real time with Ramen.

Bibliography

[1] Cabinet Office, "Understanding Crowd Behaviours-Guidance and lessons identified," Tech. Rep., 2009. [Online]. Available: http://library.college.police.uk/docs/cabinetoffice/guidancelessons1.pdf

[2] J. Drury, "The Mass Psychology of Disasters and Emergency Evacuations : A Research Report and Implications for Practice The mass psychology of disasters and emergency evacuations : A research report and implications for practice Dr John Drury and Dr Chris Cocking," no. November, 2015. [Online]. Available: http://sro.sussex.ac.uk/14386/

[3] S. J. Guy, S. Kim, M. C. Lin, and D. Manocha, "Simulating heterogeneous crowd behaviors using personality trait theory," in *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation - SCA '11*, 2011, p. 43. [Online]. Available: http://gamma.cs.unc.edu/personality/Personality.pdfhttp://dl.acm.org/citation.cfm?doid=2019406.2019413

[4] D. Masad and J. Kazil, "MESA: An Agent-Based Modeling Framework," *Proceedings of the 14th Python in Science Conference (SCIPY 2015)*, no. Scipy, pp. 53–60, 2015. [Online]. Available: http://conference.scipy.org/proceedings/scipy2015/pdfs/jacqueline{_}kazil.pdf

[5] S. Takahashi, D. Sallach, and J. Rouchier, "Advancing Social Simulation: The First World Congress: The First World Congress (Google eBook)," p. 370, 2008. [Online]. Available: http://books.google.com/books?id=jvkxp1QTDpkC{&}pgis=1

[6] Francisco J.Miguel Quesada, "Simulación Social: Una introducción — Laboratori de Simulació de Dinàmiques Socio-Històriques."

[7] X. Li, W. Mao, D. Zeng, and F. Y. Wang, "Agent-based social simulation and modeling in social computing," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2008, vol. 5075 LNCS, pp. 401–412.

[8] C. Cioffi-Revilla, "Computational social science," pp. 259–271, may 2010. [Online]. Available: http://doi.wiley.com/10.1002/wics.95

[9] A. Nouman, A. Anagnostou, and S. J. Taylor, "Developing a Distributed Agent-Based and DES Simulation Using poRTIco and Repast," in *2013 IEEE/ACM 17th International Symposium on Distributed Simulation and Real Time Applications*. IEEE, oct 2013, pp. 97–104.

[10] U. Wilensky, "NetLogo Home Page," 2012. [Online]. Available: https://ccl.northwestern.edu/netlogo/http://ccl.northwestern.edu/netlogo/index.shtml

[11] F. Campuzano, I. Doumanis, S. Smith, and J. A. Botia, "Intelligent environments simulations, towards a smart campus," in *2nd International Workshop on Smart University*, 2014.

[12] SweetHome3D, "Sweet Home 3D - Draw floor plans and arrange furniture freely." [Online]. Available: http://www.sweethome3d.com/

[13] C. Rossant, *Learning IPython for interactive computing and data visualization.*

[14] E. Merino Machuca, "Design and implementation of an agent-based social simulation model of energy related occupant behaviour in buildings," Master's thesis, ETSI Telecomunicación, June 2017.

[15] P. Aznar Delgado, "Design and development of an agent-based social simulation visualization tool for indoor crowd analytics based on the library three.js," ETSI Telecomunicación, June 2017, bachelor thesis.

[16] M. Fidalgo Vega, "NTP 390: La conducta humana ante situaciones de emergencia: análisis de proceso en la conducta individual," *Instituto Nacional de Seguridad e Higiene en el Trabajo*, pp. 1–11, 1993. [Online]. Available: http://www.insht.es/InshtWeb/Contenidos/Documentacion/FichasTecnicas/NTP/Ficheros/301a400/ntp{_}390.pdf

[17] P. Sanders, "Time Dependent Contraction Hierarchies – Basic Algorithmic Ideas," apr 2008.