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 A VISUALIZATION MODULE FOR AGENT-BASED SOCIAL SIMULATIONS APPLIED TO RADICALISM SPREAD

TASIO MÉNDEZ AYERBE

2018

TRABAJO DE FIN DE GRADO

Título:	Diseño e Implementación de un Módulo de Visualización para Simulaciones basadas en Agentes aplicado a la Difusión del Radicalismo
Título (inglés):	Design and Implementation of a Visualization Module for Agent-based Social Simulations applied to Radicalism Spread
Autor:	Tasio Méndez Ayerbe
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Departamento de 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 A VISUALIZATION MODULE FOR AGENT-BASED SOCIAL SIMULATIONS APPLIED TO RADICALISM SPREAD

Tasio Méndez Ayerbe

Junio 2018

Resumen

Las herramientas de simulación social basadas en agentes fueron desarrolladas como una aplicación de la tecnología de agentes. Constituyen la intersección de tres campos científicos, computación basada en agentes, ciencias sociales y la simulación por ordenador. Debido a que su uso se ha incrementado en el contexto social, en este proyecto se propone una interfaz web para su visualización.

El objetivo de este proyecto es desarrollar una aplicación web que permita visualizar y analizar los resultados de las simulaciones. Esta permite analizar los resultados en tiempo real así como lanzar nuevas simulaciones que pueden ser configuradas directamente desde el navegador.

Para este propósito, se ha desarrollado un modelo que trata de simular la difusión del radicalismo en una sociedad. El modelo tiene por objeto mejorar el entendimiento de la influencia de los vínculos sociales sobre la propagación del radicalismo. Consiste en dos entidades principales, un modelo de difusión y un modelo de red. El modelo de red actualiza las relaciones entre los agentes basándose en la proximidad y en la homofilia, simulando la difusión de información y actualizando la creencia de los agentes.

El modelo ha sido implementado en Python con el simulador social basado en agentes Soil y ha sido evaluado utilizando un análisis de sensibilidad mientras que la aplicación utiliza D3.js, que es una poderosa librería de JavaScript, para renderizar los resultados de la simulación y poder analizarlos.

Este proyecto está dividido en tres fases: el diseño y la implementación del módulo de visualización, el análisis e investigación para adquirir los conocimientos necesarios para llevar a cabo el modelado de la difusión del radicalismo y la implementación del modelo, así como hacer la aplicación auto-configurable para poder realizar la simulación desde el propio navegador.

Palabras clave: simulación social basada en agentes, Soil, D3.js, aplicación web, visualización, análisis, grafos, redes sociales, difusión, radicalización, terrorismo.

Abstract

Agent-based Social Simulation tools have been developed as one of the applications of agent technology. It constitutes the intersection of three scientific fields, namely, agent-based computing, the social sciences, and computer simulation. Due to its increasingly use in social context, in this thesis, a web interface for its visualization is presented.

The aim of this project is to develop a web application which allows the user to visualize and analyze the results of a simulation. It allows the user to analyze data in real time and launch new simulations which can be configured from the web.

For this purpose, a model which tries to simulate the growth of radicalism in a society is developed. The model aims at improving the understanding of the influence of social links on radicalism spread. The model consists of two main entities, a spread model and a network model. The network model updates the agent relationships based on proximity and homophily, it simulates information diffusion and updates the agents' beliefs.

The model has been implemented in Python with the agent-based social simulator Soil and it has been evaluated using a sensitivity analysis while the application uses D3.js, which is a powerful JavaScript library, for rendering the results of the simulation and analyze them.

This thesis is divided in three phases: the design and implementation of the visualization module, the analysis and research necessary to acquire the knowledge for modeling radicalism diffusion and the implementation of the model as well as making the application auto-configurable to simulate the model from it.

Keywords: agent-based social simulation, Soil, D3.js, web application, visualization, analysis, graphs, social networks, diffusion, radicalization, terrorism.

Agradecimientos

Me gustaría dar las gracias a mis padres, Miren y Tino, por haberme motivado y ayudado durante todos los años de mi vida. Gracias a ellos he podido llegar a donde estoy y ser lo que soy. También me gustaría dar las gracias a toda mi familia por haber estado apoyándome durante toda la carrera.

Gracias al Grupo de Sistemas Inteligentes por haberme dado la oportunidad de realizar este trabajo y, en especial, a mi tutor Carlos A. Iglesias por haberme orientado y ayudado a lo largo de todo el desarrollo.

Contents

List of Figures

List of Tables

CHAPTER 1

Introduction

1.1 Context

Research works on political terrorism began in the early 1970s. These works were focused on collecting empirical data and analyzing it for public policy purposes. Terrorist activity was usually attributed to personality disorders or "irrational" thinking [1]. However, later research paint a richer picture, and suggest that there are many additional factors that should be considered.

Many scholars, government analysts and politicians point out that since the mid 1990s terrorism has changed. This "new" form of terrorism is often motivated by religious beliefs and it is more fanatical, deadly, and pervasive. It also differs in terms of goals, methods and organization [1, 2].

However, the drivers of current terrorism involve not only political or religious interests but also include fanaticism. Consequently, terrorism is the result of a complex process of radicalization. i.e., a progressive adoption of extreme political, social or religious ideals.

Nevertheless, this process does not always lead to violence acts such as terrorism [3]. It is of vital importance to understand the properties of radicalization in order to anticipate said

violence. The main challenge with regard to understanding how these organizations work is that information is not always available. And, when it is available, it is often incomplete or inaccurate.

One common approach to face terrorism is trying to understand its roots, motivation and practices. In particular, it is of vital importance nowadays to understand how terrorist organizations recruit new members and isolate them. Moreover, terrorist organizations have effectively used social media and social networks to expand their networks through real-time information exchange.

As society and new forms of communications evolve, terrorists are developing new forms of organization for their purposes. Organizations can thus flatten out their pyramid of authority and control. The resulting structure can take different forms, from a dense network to a group of more or less autonomous, dispersed entities, linked by communications and perhaps nothing more than a common purpose [4]. Thus, terrorist organizations can be modeled as Social Networks (SNs) where vertices represent members of the organization and links represent communication between members.

Regardless of their structure, terrorist organizations are by definition SNs, and can be modeled as such. Hence, a research based on Agent-based Social Simulation (ABSS) could be a good starting point for understanding the information flow within the network. Thus, we are going to review some simulators that are currently published.

A lot of ABSS simulators are written in Java as we can see in Table 1.1. As terrorist organizations can be modeled as SNs, the best ones are that ones that have a specific domain in it. Those are three: HashKat, Krowdix and Soil.

HashKat [6] is a C++ ABSS platform specifically designed for the study and simulation of social networks. It includes facilities for network growth and information diffusion, based on a kinetic Monte Carlo model. It exports information to be processed by machine learning libraries such as NetworkX [7] or R's iGraph [8]. The simulator is highly performant, but has two major drawbacks. Firstly, simulations are expressed in a descriptive language and agents are created by specifying a set of highly configurable parameters. As a result, adding behaviors beyond those already included in the platform involves adding new capabilities to the framework. Secondly, and most importantly, modifications to these behaviors are very tied to the architecture of the platform, rather than being isolated for every type of agent. This makes customization costly, especially for someone without a C++ background.

On the other hand, Krowdix [9] is built on Java ABSS. It uses JUNG [10] for network functions and JFreeChart [11] for visualization. The simulation model considers users, their

Name	Domain	Language	SNs	SNA	OS
Cormas	Generic	VisualWorks	✓	✗	✓
NetLogo	Generic	NetLogo, Scala & Java	✓	✗	✓
Swarm	Generic	Objective-C, Java	✓	✗	✓
MadKit	Generic	Java	✓	✗	✓
MASON	Generic	Java	✓	✗	✓
Repast	Generic	Java	✓	✓	✓
SeSam	Generic	Java	✗	✗	✓
MASeRaTi	Generic	Java	✗	✗	✓
Mesa	Generic	Python	✗	✗	✓
UbikSim	AmI	Java	✗	✗	✓
EscapeSim	Evacuation	Java	✗	✗	✓
HashKat	Social Networks	C++	✓	✓	✓
Krowdix	Social Networks	Java	✓	✓	✗
Soil	Social Networks	Python	✓	✓	✓

Table 1.1: Review of ABSS platforms [5]

relationships, user groups and interchanged contents. Its main drawback is that it is not open source.

Conversely, Soil is open source and built using Python and custom agent models are also developed using Python. This has the advantage of Python's increased popularity, its very gradual learning curve, readability, clear syntax and availability of libraries for network processing and machine learning. The network features of Soil are based on NetworkX, which is the defacto standard library for Social Network Analysis (SNA) of small to medium networks.

The main drawback is that once you run the simulation you need to use a Jupyter Notebook to analyze data and for visualizing it you need an external software. The solution, comes from having a web application which could be accessed from the browser and provides you the option of running a simulation, visualize and analyze it using the SNA tools that Soil provides.

As a summary, the aim of the project is to develop the model using SNA tools which help us to understand the performance of this type of organizations with the purpose of decrease the impact of them in the current society. Nonetheless, the aim is not only to develop the model but also to provide the necessary tools to visualize and analyze the results.

1.2 Project goals

The aim of this project is to develop a web interface which allows the user to visualize and analyze the result of different simulations based on graphs. Once the simulation is finished, the user will be able to interact with the simulation in real time.

For this purpose, a model which tries to simulate the spread of information to comprehend the radicalism diffusion in a society will be developed. Thus, the main project goals could be divided in four as it is shown in the following.

- Simulate the spread of information through a network applied to radicalism.
- Visualize the results of the simulation and interact with the simulation in real time.
- Extract data from the results.
- Show data correctly for its research.

1.3 Structure of this document

In this section we provide a brief overview of the chapters included in this document. The structure is as follows:

Chapter 1 is the introduction of the project. A description of the context in which the project is developed and its main goals are presented.

Chapter 2 provides the technologies which have been used throughout the project and why have been chosen.

Chapter 3 describes the architecture of the web application based on different modules. A global overview about the architecture is presented and all its components detailed.

Chapter 4 explains the aim of the model and its design to achieve the objectives defined.

Chapter 5 provides the implementation required to match its design based on research and presents the project outcomes.

Chapter 6 discuss the conclusions, the achieved goals and future work.

CHAPTER 2

Enabling Technologies

In this chapter, the technologies that have been used throughout the project are introduced and detailed. It includes several programming languages because of the client-server architecture, Python to deploy the server and to develop the agent, and JavaScript for the visualization.

First of all, Soil Simulator is introduced in Sect. 2.1. Secondly, Python libraries for developing the model and for deploying the server are introduced in Sect. 2.2 and Sect. 2.3. JavaScript and Python libraries used for visualization are explained in Sect. 2.4. Finally, in Sect. 2.5 the tools used for evaluating the model are described.

2.1 Soil: Agent-based Social Simulator

Soil[1] is a modern Agent-based Social Simulator for modeling and simulation of SNs developed by GSI-UPM [5]. It has been applied to a number of scenarios, ranging from rumor propagation to emotion propagation and information diffusion. It also provides you the tools for developing new models and agents based on Social Networks and everything you need to analyze the results.

[1] http://soilsim.readthedocs.io/en/latest/

This software extends and uses NetworkX which will be detailed in Sect. 2.2, so a lot of tools that NetworkX provides can be used. There are three main elements in a simulation:

- **Network topology**. You can generate one or use an existing one of NetworkX.
- **Agents**. There are two types of agents:
  - network agents, which are linked to a node in the topology.
  - environment agents, which are freely assigned to the environment.

  The one that we will use for modeling a terrorist organization is the network agent. An agent will represent a member of the organization or a civilian and the links will represent flow of information.
- **The environment**. It is the place where the shared state of the simulation is stored. Any agent has unrestricted access to the environment.

Soil is based on simpy [12], which is an event-based network simulation library. It provides several abstractions over events to make developing agents easier.

2.1.1 Modeling behavior

When you model an agent, it should inherit `soil.agents.BaseAgent`, and define its behavior in every step of the simulation by implementing a `run(self)` method.

Instead of define a run method, a Finite State Machine (FSM) could be used which will be based on `soil.agents.FSM`. On it, every step is defined as a function and to change to another state, the function have to return the new state.

Agents are a way of modeling behavior. They can be characterized with two variables: an agent type and its state. When you inherit the `soil.agents.BaseAgent` class, the agent has access to its state and the environment parameters. Through the environment, it can access the network topology and the state of other agents.

- `agent.state`, a dictionary with the state of the agent where you can save some constants that defines the agent in that step.
- `agent.env`, a reference to the environment. With this variable you have access to the environment parameters, even to the network topology.

As you can access the environment from an agent, you can modify it. However, this is not recommended. For the sake of simplicity, it is recommended limiting environment updates to environment agents.

2.1.2 Run simulation

Once you have defined the necessary agents, you are able to run the simulation and you will only need to load configuration. For this purpose, you can load it from Python dictionaries as well as JavaScript Object Notation (JSON) and YAML files.

To configure the network agents, the configuration file allows you to specify how agents will be mapped to topology. It is also possible to add more than one type of agent to the simulation, and to control the ratio of each type using the `weight` property. In addition, you can add a custom initial state to each agent.

Environment agents are not that different, the main difference is that they are not assigned to network nodes.

Finally, as global settings, you could specify the network topology as was explained before, you could define the number of total steps and the number of trials. The trials will be run one behind the other but the results will be stored in different files with the purpose of contrast them.

Four types of objects are saved by default: a pickle of the simulation; a YAML representation of the simulation (which can be used to re-launch it); and for every trial, a CSV file with the content of the state of every network node and the environment parameters at every step of the simulation, as well as the network in Graph Exchange XML Format (GEXF) format.

Another way of running a simulation is importing the package `soil.simulation` into a Python script. Then you are not only able to run the simulation and export the results, but you could also get the results as a variable in the script to interact with them using NetworkX methods.

2.2 NetworkX

NetworkX[2] is a Python language package for exploration and analysis of networks and network algorithms [7]. It includes data structures for representing many types of networks, or graphs, including undirected graphs, directed graphs and flexible graphs that allows multiple undirected or directed links between pairs of nodes.

NetworkX is the defacto standard library for SNA analysis of small to medium net-

[2]https://networkx.github.io/documentation/stable/

works. It is interoperable with a great number of graphs formats, including GEXF, Graph Modelling Language (GML) and GraphML.

It also supports adding attributes to graph, nodes, and edges. Attributes such as weights, labels, colors, or whatever Python object you like, can be attached to graphs, nodes or edges.

Each graph, node, and edge can hold key/value attribute pairs in an associated attribute dictionary. By default, these are empty, but attributes can be added or changed on the fly.

It is a powerful tool for using it with small and medium networks and to apply algorithms of search. Some examples of methods that have been applied to the model are the following.

- **shortest_path_length**, computes the shortest path lengths in the graph from the source node to the target.
  Applying this method, I have been able to do a method that returns the shortest path between two nodes in the graph and if they are not connected, it returns infinity.

  Listing 2.1: The shortest path length between two nodes

```
def shortest_path_length(self, G, source, target):
    try:
        return nx.shortest_path_length(G, source, target)
    except nx.NetworkXNoPath:
        return float('inf')
```

- **ego_graph**, returns induced subgraph of neighbors centered at node n within a given radius. Node, edge and graph attributes are copied to the returned subgraph.
  Using this function, I have been able to do a function that searches all the nodes reachable starting from one passed as an argument in a determined number of steps.

  Listing 2.2: Social search

```
def social_search(self, G, node, steps):
    nodes = list(nx.ego_graph(G, node, radius=steps).nodes())
    nodes.remove(node)
    return [ G.nodes()[index]['agent'] for index in nodes ]
```

In addition, using other methods from this library, a function that searches for nodes depends on their position has been made. This method could be interesting for graphs that depends on the positions of the nodes, so you could get all the nodes reachable starting from one passed as an argument in a determined radius.

Listing 2.3: Link search

```
def link_search(self, G, node, radius):
    pos = nx.get_node_attributes(G, 'pos')
    nodes, coords = list(zip(*pos.items()))
    kdtree = KDTree(coords)  # Cannot provide generator.
    edge_indexes = kdtree.query_pairs(radius, 2)
    _list = [ edge[int(not edge.index(node))] for edge in edge_indexes if
        node in edge ]
    return [ G.nodes()[index]['agent'] for index in _list ]
```

2.3 Tornado

Tornado[3] is a Python web framework and asynchronous networking library [13]. Tornado can scale to tens of thousands of open connections, making it ideal for WebSockets and other applications that require a long-lived connection to each user.

Tornado can be roughly divided into four major components:

- A web framework, including subclasses to create web applications.
- Client and server-side implementations of HTTP.
- An asynchronous networking library including the classes `IOLoop` and `IOStream`, which serve as the building blocks for the HTTP components and can also be used to implement other protocols.
- A co-routine library which allows asynchronous code to be written in a more straightforward way than chaining callbacks.

The `tornado.web` package provides a web framework with asynchronous features. Importing this package allows you to render HTML templates in the browser depending on the URL. It also supports such a list of methods to interact with requests as you can receive post or get requests.

The package from this framework that routes HTTP requests to appropriate handlers is the `tornado.routing` package. You have to configure this routes and passed them to the `init` method of the `tornado.web.Application`. That provides you render the HTML template and to import JavaScript or CSS files. For example, the necessary routes for the web application are these.

[3] http://www.tornadoweb.org/en/stable/

Listing 2.4: Tornado routes

```
page_handler = (r'/', PageHandler)
socket_handler = (r'/ws', SocketHandler)
static_handler = (r'/(.*)', tornado.web.StaticFileHandler, {'path': '
    templates'})
local_handler = (r'/local/(.*)', tornado.web.StaticFileHandler, {'path': ''
    })
```

The first one is the one needed for rendering the HTML template. The second one is the one needed for handling sockets as it is explained below. The other ones are the ones needed for rendering the correct files in the directory and for importing the JavaScript and CSS files from the right path.

Tornado implements the *WebSocket protocol* which is supported in the current versions of all major browsers. The WebSocket protocol accept bidirectional communication between the browser and server over a single TCP socket. This bidirectional communication and low latency that provides [14], makes it good to use for delivering data to the client's web browser in real time [15].

Tornado had a module that implements the final version of the WebSocket protocol defined in RFC 6455. That module is `tornado.websocket` and it is the one that we will use for controlling the application on the browser.

To use it, you have to use the WebSocket Handler as it was shown in Lst. 2.4. For using this handler you have to override the `on_message` method to handle incoming messages, and use `write_message` to send messages to the client. You can also override `open` and `on_close` methods to handle opened and closed connections.

Once you have defined the handler in your application, you can invoke it in JavaScript as it is shown in Lst. 2.5.

Listing 2.5: Tornado connection from JavaScript

```
var ws = new WebSocket('ws://' + window.location.host + '/ws');
ws.onopen = function() { console.log('Connection opened!'); };
ws.onmessage = function(message) { console.log('Message received!'); };
```

2.4 Visualization libraries

In this section, some visualization libraries which have been used among the project are presented. These libraries include visualization methods for performing charts in JavaScript and Python.

2.4.1 D3.js

D3.js[4] refers to Data-Driven Documents [16] and it is defined according to its official Wiki as following.

> *D3.js is a JavaScript library for manipulating documents based on data. D3 helps you bring data to life using HTML, Scalable Vector Graphics (SVG) and CSS. D3's emphasis on web standards gives you the full capabilities of modern browsers without tying yourself to a proprietary framework, combining powerful visualization components and a data-driven approach to Document Object Model (DOM) manipulation.*

D3 is extremely fast, supporting large datasets and dynamic behaviors for interaction and animation. Thus, this library will be used to show data from the simulation in the browser, it will help us to represent the network as a graph [17].

It allows us to manipulate nodes separately such as setting attributes or styles, registering event listeners, adding, removing or sorting nodes. It also allows us to *enter* and *exit* for creating new nodes for incoming data and remove outgoing nodes that are no longer needed.

Furthermore, many graphics libraries do not provide a scene graph that can be inspected for debugging. For example, Processing [18] renders the content into a Canvas what makes it hard to debug, and Raphaël [19] encapsulates SVG and Vector Markup Language (VML). In addition, the documentation for these libraries is not as extensive as D3 documentation due to its demand.

D3 is such an extended JavaScript library that there are a lot of other libraries that extends this one and some of them are specifically for representing graphs.

As is summarized in Table 2.1 it exists some JavaScript libraries for representing graphs [20] and some of them are based in D3 such as Alchemy.js and JSNetworkX. This last is a port

[4]https://github.com/d3/d3/wiki

Name	Support Categories	JSON	SVG	OS
D3.js	Networks / Charts / Hierarchies	✓	✓	✓
Alchemy.js	Networks	✓	✓	✓
Sigma.js	Networks	✓	✗	✓
Vis.js	Networks / Charts	✓	✗	✓
Linkurious	Networks	✓	✓	✗
Cytoscape.js	Networks	✓	✗	✓
JSNetworkX	Networks	✓	✓	✓

Table 2.1: Review of JavaScript libraries for representing Graphs

of the Python graph library NetworkX but it is still in an early stage of development.

Sigma.js is a lightweight JavaScript library dedicated to graph drawing; Vis.js is a dynamic, browser based visualization library, and Cytoscape.js is a graph network library for analysis and visualization. Finally, Linkurious is a graph visualization and analysis software that helps to identify hidden insights but it is not open source.

For the purpose of this thesis, the most suitable library seems to be JSNetworkX because of the use of NetworkX library in Python but it is still in development. Of the others, Sigma, Vis or Cytoscape are suitable. However, it was implemented without using any of those libraries for the non-complexity of the graph. Notwithstanding, for the analysis of more complex models, one of those could be implemented in the application.

2.4.2 C3.js

C3.js[5] is a D3-based reusable chart library. It makes it easy to generate D3-based charts by wrapping the code required to construct the entire chart.

It gives some classes to each element when generating, so you can define a custom style by the class and it is possible to extend the structure directly by D3. C3 provides a variety of APIs and callbacks to access the state of the chart. By using them, you can update the chart even after it's rendered.

Thanks to this library, show charts for analyzing the results from a simulation is easier. It only needs some configuration for costuming the chart and get the data as an array. Thus, we will be able to represent statistics and analysis in a proper way to its comprehension.

[5] http://c3js.org/reference.html

2.4.3 Seaborn

Seaborn[6] is a library for making attractive and informative statistical graphics in Python. It is built on top of matplotlib and tightly integrated with the PyData stack, including support for numpy and pandas data structures and statistical routines from scipy and statsmodels [21].

The plotting functions try to do something useful when called with a minimal set of arguments, and they expose a number of customize options through additional parameters. This library has made easier the fact of analyzing the results from the Sensitivity Analysis (SA) made with SALib.

2.5 Sensitivity Analysis

SA is defined as "the study of how the uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input" [22].

The first historical approach to SA is known as the local approach. The impact of small input perturbations on the model output is studied. This small perturbations occur around nominal values (the mean of a random variable for instance). This deterministic approach consists in calculating or estimating the partial derivatives of the model at a specific point [23].

From the late 1980s, to overcome the limitations of local methods (linearity and normality assumptions, local variations), a new class of methods has been developed in a statistical framework. In contrast to local sensitivity analysis, it is referred to as "global sensitivity analysis" because it considers the whole variation range of the inputs [24]. The objectives of SA are numerous, these may include:

- identify and prioritize the most influential inputs
- identify non-influential inputs in order to fix them to nominal values
- map the output behavior in function of the inputs by focusing on a specific domain of inputs if necessary
- calibrate some model inputs using some available information (real output observations, constraints, etc.)

[6] https://seaborn.pydata.org/

There are many approaches to performing a Sensitivity Analysis, some of them are compared in Table 2.2.

Type	Morris	Variance	Factorial	Monte Carlo	Local SA
Model independent?	yes	yes	yes	yes	yes
Sample source	levels	distributions	levels	distributions	levels
Number of factors	$20 - 100$	< 20	> 100	< 20	< 100
Factor range	global	global	global	global	local
Multi-factor variation	yes	yes	yes	yes	no
Correlated factors?	no	no	yes	yes	no
Cost (for k factors)	$10(k+1)$	$500(k+1)$	$k \rightarrow 2k$	$500+1$	$2(k+1)$
Estimated CPU time[7]	1 day	11 days	3 hours	2 days	1 hour

Table 2.2: Review of Sensitivity Analysis techniques [25]

SALib[8] is an open source library written in Python for performing sensitivity analysis. SALib provides a decoupled workflow, meaning it does not directly interface with the mathematical or computational model. Instead, SALib is responsible for generating the model inputs, using one of the `sample` functions, and computing the sensitivity indices from the model outputs, using one of the analysis functions [26].

SALib provides several sensitivity analysis methods, such as Sobol, Morris, and FAST. However, regardless of which method is chosen, you need to use only two functions: `sample` and `analyze`. Thus, a typical sensitivity analysis using SALib follows four steps:

1. Determine the model inputs (parameters) and their sample range.
2. Run the `sample` function to generate the model inputs.
3. Evaluate the model using the generated inputs, saving the model outputs.
4. Run the `analyze` function on the outputs.

[7] Assuming five minutes per simulation and 30 groups of factors
[8] http://salib.readthedocs.io/en/latest/

CHAPTER 3

Soil Web Application

In this chapter, the web application is described. Firstly, in Sect. 3.1 a brief analysis is presented reviewing some other platforms. Sect. 3.2 explain the global architecture with aim of discussing the main modules that compose it. In the following sections, modules are detailed. Finally, in Sect. 3.5 a brief conclusion about the architecture is provided.

3.1 Analysis

Most of the ABSS platforms discussed in Sect. 1.1 does not provide a user interface and if they do it sometimes it ends up being more difficult than it is supposed to be.

From those, Netlogo [27] includes facilities for network representations although not for network analysis. It provides a web interface with a few configuration options but not as much as the installable version. Furthermore, Netlogo uses Logo as its main programming language for developing new agents.

Soil does not provide a visualization module although it provides the necessary files for visualizing the results with external software as Gephi [28], which is an open source software for graph and network analysis. It provides the necessary files for the visualization and tools

for SNA although it does not provide the environment to carry this out.

The solution to this problem comes from developing a web interface that groups these features: simulate, visualize and analyze the network using a single tool.

Netlogo developed a desktop application for this purpose as its web version is a simplified version. However, desktop applications are decreasing due mostly to that they are designed for a specific operative system, virtual machines, or they need external programs to be installed in the user's desktop computer [29]. Web applications are totally independent of these features, what makes them attractive for Internet users, and now are very popular.

Thus, in Fig. 3.1 the use case of the application is presented. There is only one actor, which is the user who uses the application and one external actor which is the Soil simulator. The user is able to perform four different actions which are a use case each of them: (1) run simulation, (2) change simulation parameters, (3) visualization and (4) analyze.

There is one added from the features presented above that is change the simulation parameters. This is an interesting use case because the user is able to run several simulations varying the input parameters with the purpose of realizing a deeper analysis.

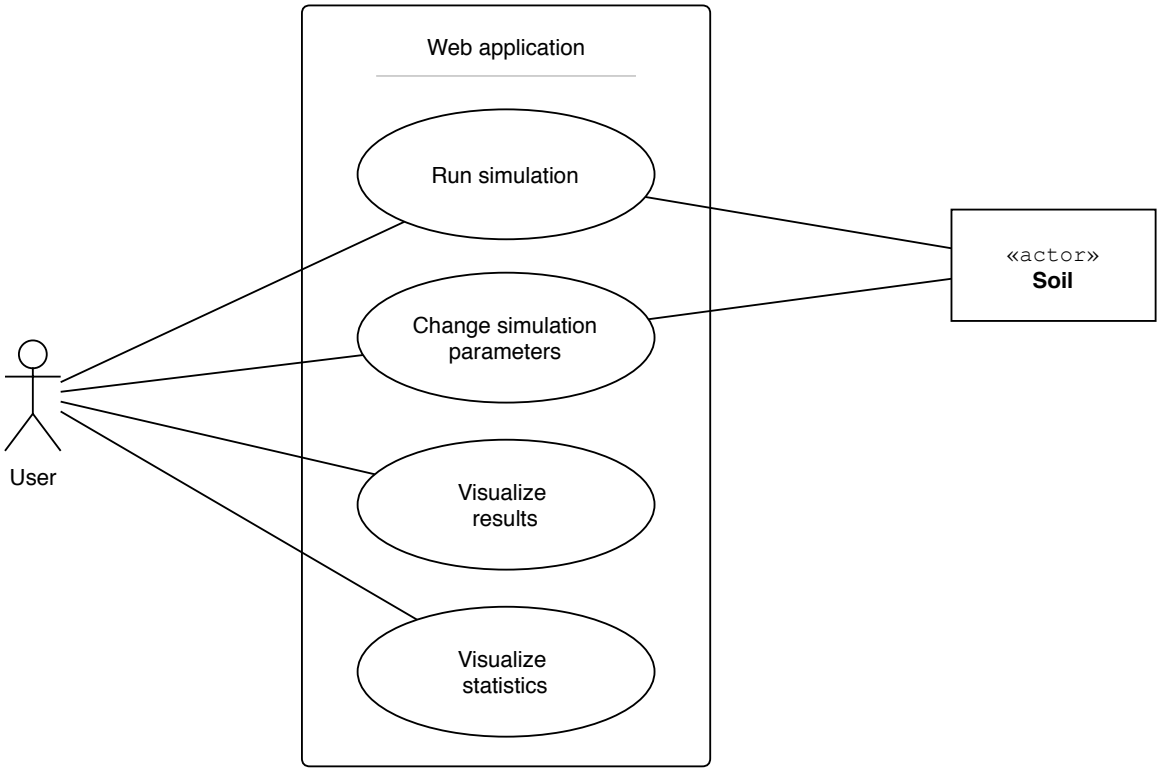


Figure 3.1: Use case

Running the simulation will be done in background, so the interface has to focus the other three ones. In Fig. 3.2 the mock-up of the first view if presented. The idea is to render the graph in a main component which has enough space for the user to interact with it. Emulating a video player, at the bottom of the page a progress bar has been placed so the user can play or pause the simulation.

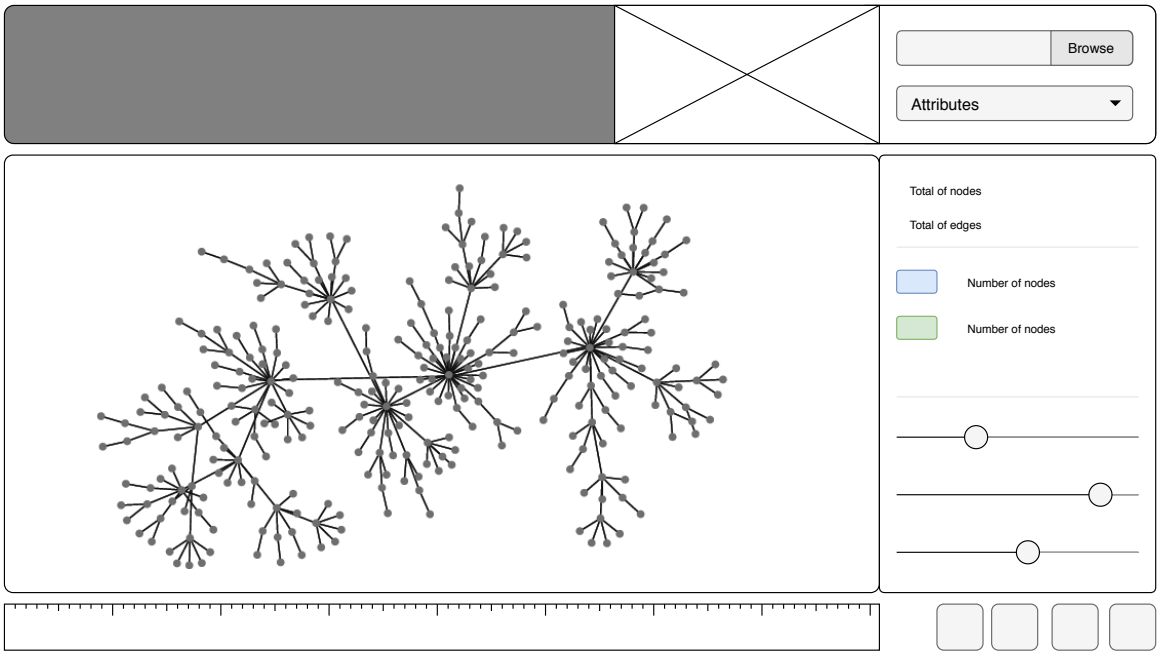


Figure 3.2: Mock-up of the main view

The right sidebar it is thought to show statistics in real time, so they change depending on the instant of time selected. However, another view that focuses the statistics is supposed to be developed as it shown in Fig. 3.3

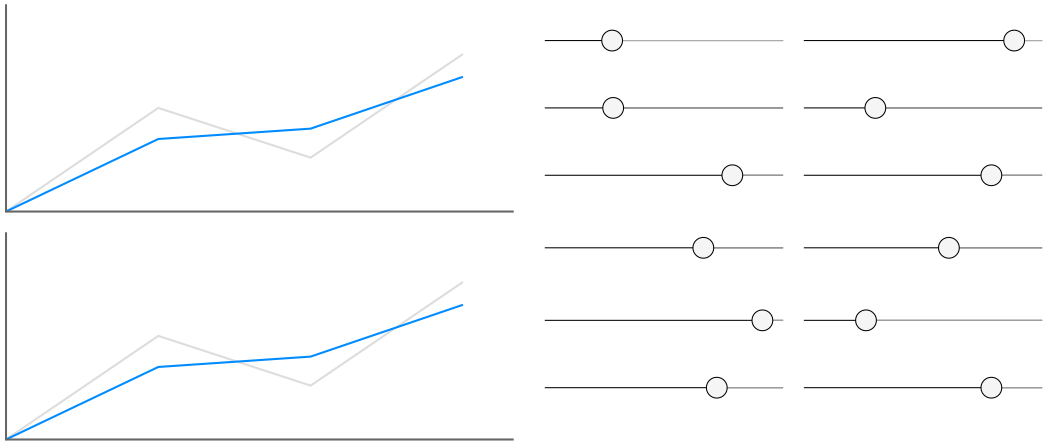


Figure 3.3: Mock-up of the statistics and configuration view

The second view is focused on statistics in the left and changing the parameters on the right. Furthermore, a select menu will give the user the option of choosing the trial to visualize.

The statistics are presented with C3.js as it was explained in Sect. 2.4. It tries to summarize the main points of the simulation such as the comparison between the states of the nodes.

The application should also give the user the option of exporting the simulation. For this purpose, as Soil gives the option of exporting the results, the application, will give the user this feature. Thus, the flow of Soil is not modified. The application only gives an environment that centralize all the features that Soil provides.

3.2 Overview

The web application has been implemented using a client-server architecture. The server side has been implemented using Python as it was explained in Sect. 2.3 while the client side uses JavaScript as its main language.

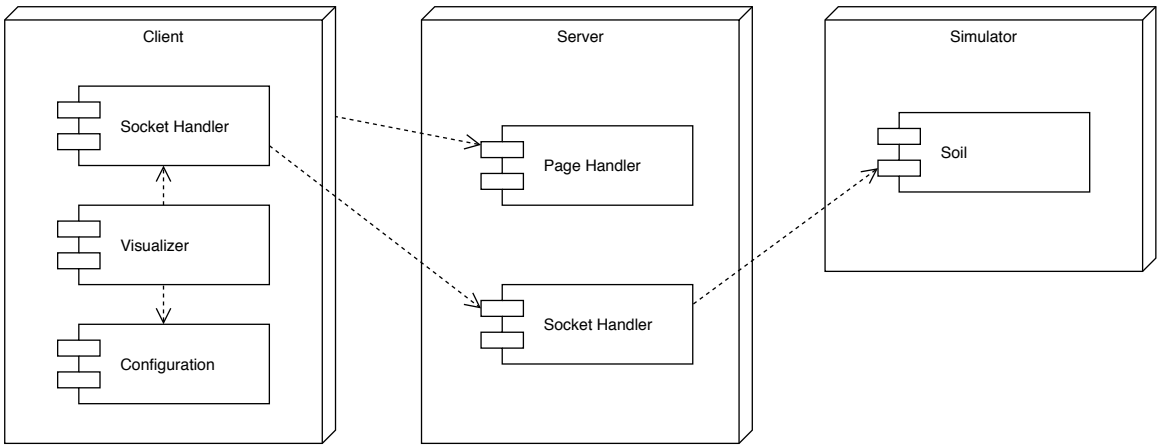


Figure 3.4: Modular architecture

In Fig. 3.4 a layered architecture is presented. As it can be noticed, server and client sides are divided into different modules that have dependencies between each other.

The modular server is in charge of handling the template which holds the visualization and the HTTP requests while the simulator will receive the data from the server and it will run as many simulations as the user have configured. The simulator uses Soil as it was explained in the previous chapter.

The client side is the one in charge of the visualization. It only receives data from the server and it has to process that information with the aim of representing the simulation in the browser. The visualizer module uses D3.js to represent the graph but that is not going to be its only role.

Another technologies have been considered as it was described in Sect. 2.4.1 for representing the network and provides interactive, dynamic data visualizations. However, D3 seemed to be the most suitable for this project.

3.3 Back-end

The server itself uses the Tornado library for launching the main process. It has as arguments the handlers that it needs to operate, that are the following ones.

- The page handler that renders and controls the HTML template which holds the visualization.
- The socket handler that is in charge of replying the HTTP requests transparently to the user.

The server could be launched locally as it listen in the indicated port by the user being `8080` the default port, although it could also be launched in a server and be accessed from any other computer.

3.3.1 Socket Handler

The Socket Handler will receive a message, parse and act accordingly to the user actions. These messages are exchanged in JSON format as it is shown in Lst. 3.1 and are divided in two main parts. The first one indicates message type which tell the handler what to do when the message is received. The other part is the data that the client side sent and its content and structure depends on the message type.

The decision of the socket handler depends on the functionalities of the application which are listed below.

- Configure and run the simulation.
- Request a trial to visualize the results.
- Run a new simulation with different input values.
- Download the results of the simulation. The application gives to the user the option of downloading the results of the last simulation in GEXF and JSONGraph format.

Listing 3.1: Socket message example

```
{
    'type': 'get_trial',
    'data': 0
}
```

The user is only able to download the last simulation because the simulations are not stored in the server. However, they are available because they are saved in the socket buffer which is different for each user and is emptied when socket is closed. Nonetheless, the server gives the option of dumping all the results from the simulations but are not accessible from the front-end.

Each connection to the server opens a new socket so that the simulations do not interfere with each other. That is the reason why the simulator is called from the Socket Handler and all the data of the session is stored also in the socket buffer.

3.3.2 Simulator

The simulator module is in charge of running the simulations with Soil but it does not store the results as it was explained above. Once the simulation is finished, the results are sent backwards to the user through the socket handler.

With the purpose of informing the user of the state of the simulation, the logs from the simulation are captured from the socket handler and sent to the user as it is shown in Fig. 3.5. This functionality has been implemented using a context manager and a string buffer in Python.

```
If you don't know how to write the file, please visit this page:
http://soilsim.readthedocs.io/en/latest/quickstart.html

$ INFO.soil: Using config: TerroristNetworkModel_sim
$ INFO.soil: Trial: 0
$ INFO.soil: Running
$ INFO.soil: Finished trial in 2.72758150100708 seconds
$ INFO.soil: NOT dumping results
$ INFO.soil: Finished simulation in 2.7977089881896973 seconds
```

Figure 3.5: Simulation logs shown to the user

The context manager defines a runtime context where the simulations are executed. Thus, when the simulation starts the buffer where the logs are going to be stored is opened and when it finishes, the buffer is closed. While the simulation is running, a background process will be in charge of reading the buffer and every time a log is written, it is sent to the user console. It is a way that the user knows in every moment the simulation state and if there has been any error.

3.4 Front-end

The front-end is completely written in JavaScript. The visualizer is supported by D3.js. The socket handler is the same as the one explained in Sect. 3.3 but using JavaScript instead of Python. It makes a request to the server with the data introduced by the user and it acts accordingly to the server reply.

Finally, the configuration module is in charge of adding as many inputs as the simulations parameters depending on the parameter type. The type parameters supported are *boolean*, *int* and *float*.

The application is divided in two different views: (1) the simulation results and (2) statistics and configuration, as it can be noticed in Fig. 3.6.

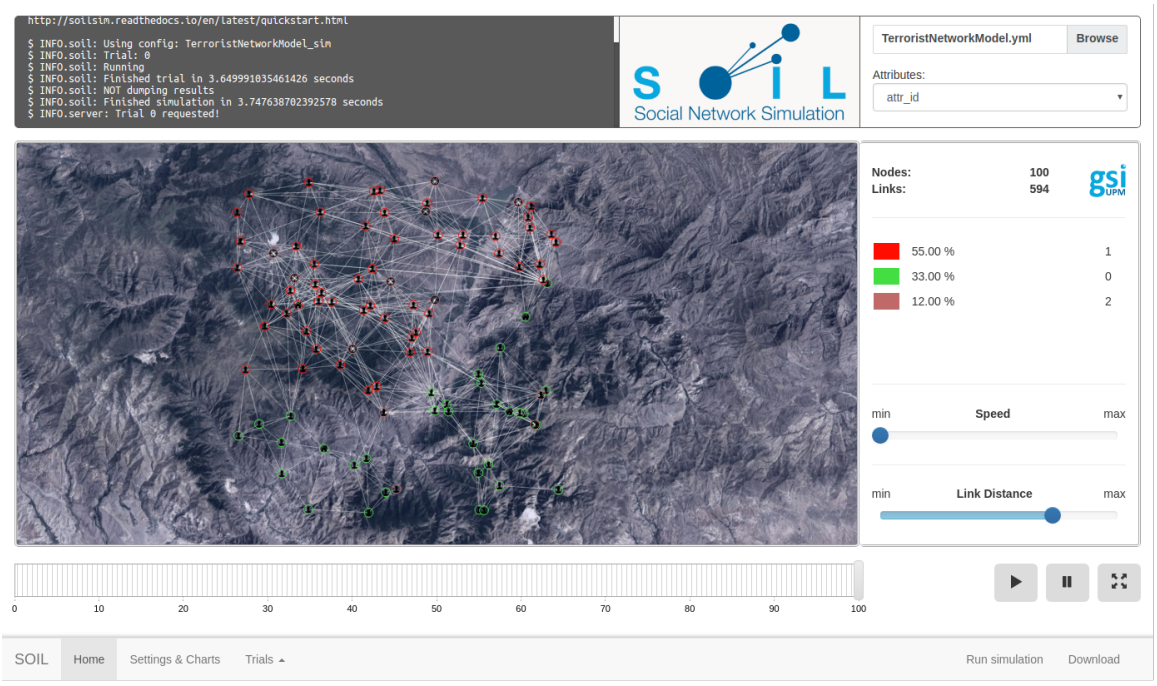
The first view shown in Fig. 3.6a where the simulation results are displayed, three main elements are distinguished. The first one is the main view that is a SVG item and it will be controlled by the visualizer module as it is discussed in Sect. 3.4.1. The second one in the right is an information panel that it also gives the option to adjust the speed of the simulation and the distance between the nodes. The last one is the progress bar which allow the user to start and pause the simulation as well as fit the graph to the SVG area. It also lets the user to move through the simulation and view the state of the graph in each instant of time.

The second view shown in Fig. 3.6b displays the statistics as well as the configuration for launching more simulations. The charts displayed on the left are computed for the trial selected as well as the attribute selected in the top bar. The sliders on the right are the environment parameters of the simulation that are detailed in the configuration file. These inputs vary depending on the type of variable, being sliders for float variables, number inputs for int variables and checkboxes for boolean variables.

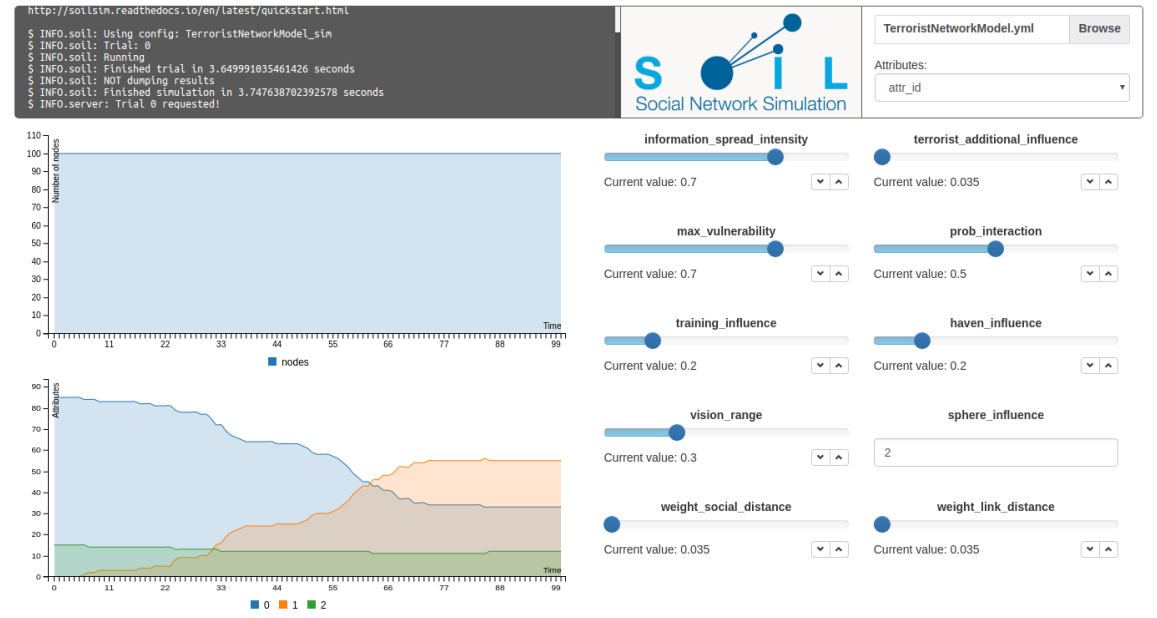
3.4.1 Visualizer

This module renders the graph received from the server into a SVG item. The module uses some private functions what makes up the *core* to compute the graph as well as some internal variables where all the necessary data is stored.

The main functions as well as some internal values that may be relevant to the user are accessible through its public API. These functions are listed and described below as well as their input parameters.



(a) Visualization



(b) Statistics and configuration

Figure 3.6: View of the application after running a simulation

- Create the SVG space where the graph will be drawn.
  - `id` of the SVG item
  - height of the item
  - width of the item
- Import the graph and attributes of all the nodes. The function is called with the results of the simulation as its only parameter.
  - the json structure of the graph
- Set the link distance between the nodes if the nodes have not coordinates.
  - distance between nodes
- Represent the graph depending on the property and time given.
  - property to show
  - instant of time
- Set a background image in the SVG item. This is part of the configuration as it is explained in Sect. 3.4.2.
  - background image
  - opacity of the image
  - color for filtering the image
- Set shapes and colors of the nodes depending on their status. As the previous one, is part of the configuration.
  - shape property
  - dictionary of shapes
  - dictionary of colors
- Adjust the graph to the SVG item giving the option of setting a padding and a transition time.
  - padding size
  - transition time
- Reset the SVG item what makes possible to import again different data without having to destroy the SVG item.

- Get the color of a node or a group of nodes depending on the attribute value given.
  - property
  - value
- Get a dictionary with all nodes and its attribute value at a specific instant of time.
  - property
  - instant of time
- Get the number of nodes at a specific time when the graph is dynamic not only its attributes but also the nodes.
  - instant of time

From this user accessible functions, the general workflow of a graph simulation is shown in Fig.3.7 taking into account that before resetting the graph, getters functions can be applied to get information about the graph.

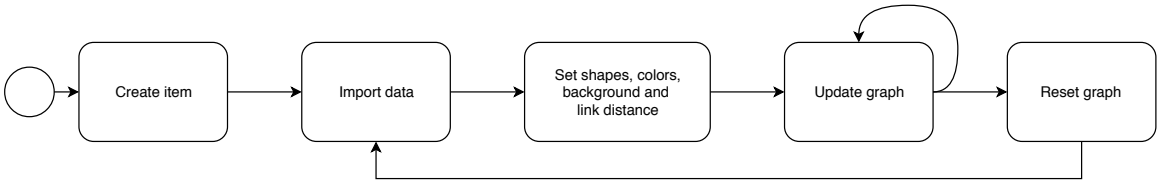


Figure 3.7: General workflow of a graph visualization

3.4.2 Configuration

The configuration of the visualization is done in the same way as the configuration in Soil. In the same YAML file where the simulation is configured, some parameters can be included to customize the visualization.

The three main customization elements are: (1) the shapes of the nodes depending on a static attribute, (2) the color depending on the value of an attribute and (3) the image background. Every node can be attached to a shape that is a SVG pattern. Following the model that will be explained in Chapter 4 an example of a shape customization is shown in Lst. 3.2.

Listing 3.2: Shape customization example

```
visualization_params:
  shape_property: agent
  shapes:
    TrainingAreaModel: target
    HavenModel: home
    TerroristNetworkModel: person
```

This means that for the property *agent* different shapes will be shown depending on the values which are described in `shapes` tag.

If the user wants to change the color of the nodes for some properties, it could be also done in a similar way as it is shown in Lst. 3.3. If the shape customization has also been applied, the color of the shape will change.

Listing 3.3: Color customization example

```
visualization_params:
  colors:
    - attr_id: civilian
      color: "#40de40"
    - attr_id: terrorist
      color: red
    - attr_id: leader
      color: "#c16a6a"
```

Finally, for setting an image background, the configuration file should look like Lst. 3.4. The filter color property and the opacity are combined to allow the user change the contrast between the background and the graph without the need of editing the picture. Thus, with some good values, the graph will not be hidden because of the color of the image background.

Listing 3.4: Background customization example

```
visualization_params:
  background_image: 'map_4800x2860.jpg'
  background_opacity: '0.9'
  background_filter_color: 'blue'
```

3.5 Conclusion

Once all the modules from the architecture are detailed, in Fig. 3.8 the main modules are presented in a sequence diagram with the purpose of detailing the exchanged messages during a regular flow. All these messages are transparent to the user, what means that the user will not notice about all the transitions.

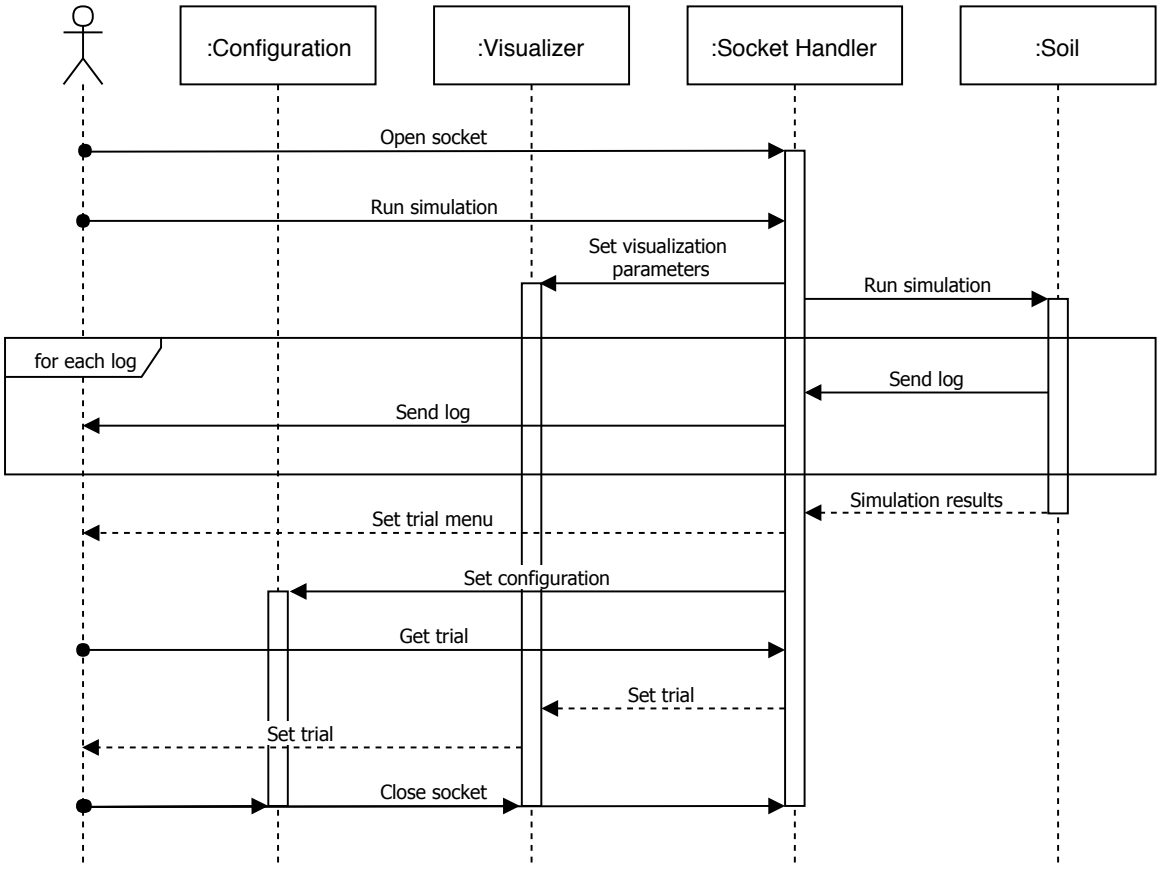


Figure 3.8: Sequence diagram

While the socket is opened, the user will be able to make requests to the server and the socket will reply to the module in charge of processing the action requested. They socket will be closed when the user closes the browser or the connection is interrupted.

CHAPTER 4

Simulation model

In this section, the agent-based model of radicalization is introduced. First of all, in Sect. 4.1 the problem faced is described with the purpose of understanding the aim of this thesis. Secondly, in Sect. 4.2 the considerations that it has been made to design the model are introduced. Sect. 4.3 describes an overview of the model itself. Sect. 4.4 and Sect. 4.5 introduces the agent-based model of radicalization. Finally, Sect. 4.6 describes the configuration of the model and visualization.

4.1 Problem

As previously discussed, in the last years, the way people communicate has changed, becoming more relevant social networks, where everyone can exchange messages, images and videos. Terrorist organizations also have moved forward by setting up radio stations, TV channels or Internet websites. These activities allow them to increase their strength, their funds and better recruit new people.

Since terrorist organizations can be modeled as social networks we can study how information is shared and how people become members of groups or even new relationships. Within the proposed model, terrorist groups will be represented as graphs where vertices

represent members and edges represent communication between those members.

However, radicalism is not only sustained by flow information. Multiple causes, rather than a single cause should be considered, including social and spacial relations which evolve over time. Estimating their evolution is important for management, command and control structures, as well as for intelligence analysis research purposes. By knowing future social and spacial distributions, analysts can identify emergent leaders, hot spots, and organizational vulnerabilities [30].

In order to approach to the radicalism spread, a spatial distribution is used based on Geometric Graph Generators [31], which provides geographical positions to agents, being able to manage real environments.

The physical space aims to produce more insightful results when considering the spread of terrorism [32]. Properties of space and place are vital components of terrorist training, planning, and activities.

Besides, based on the principle of homophily, as a contact between similar people occur at a higher rate than among dissimilar people, it is more likely to have contact with those who are closer to us in geographic location than those who are distant [33]. It is theorized that, in general, proximity in geographic space strongly influences closeness in social space [32].

As it was discussed above, the proposed model will try to approach to the fact of the rise of radicalism within a specified geographic area considering real geographical connections between members.

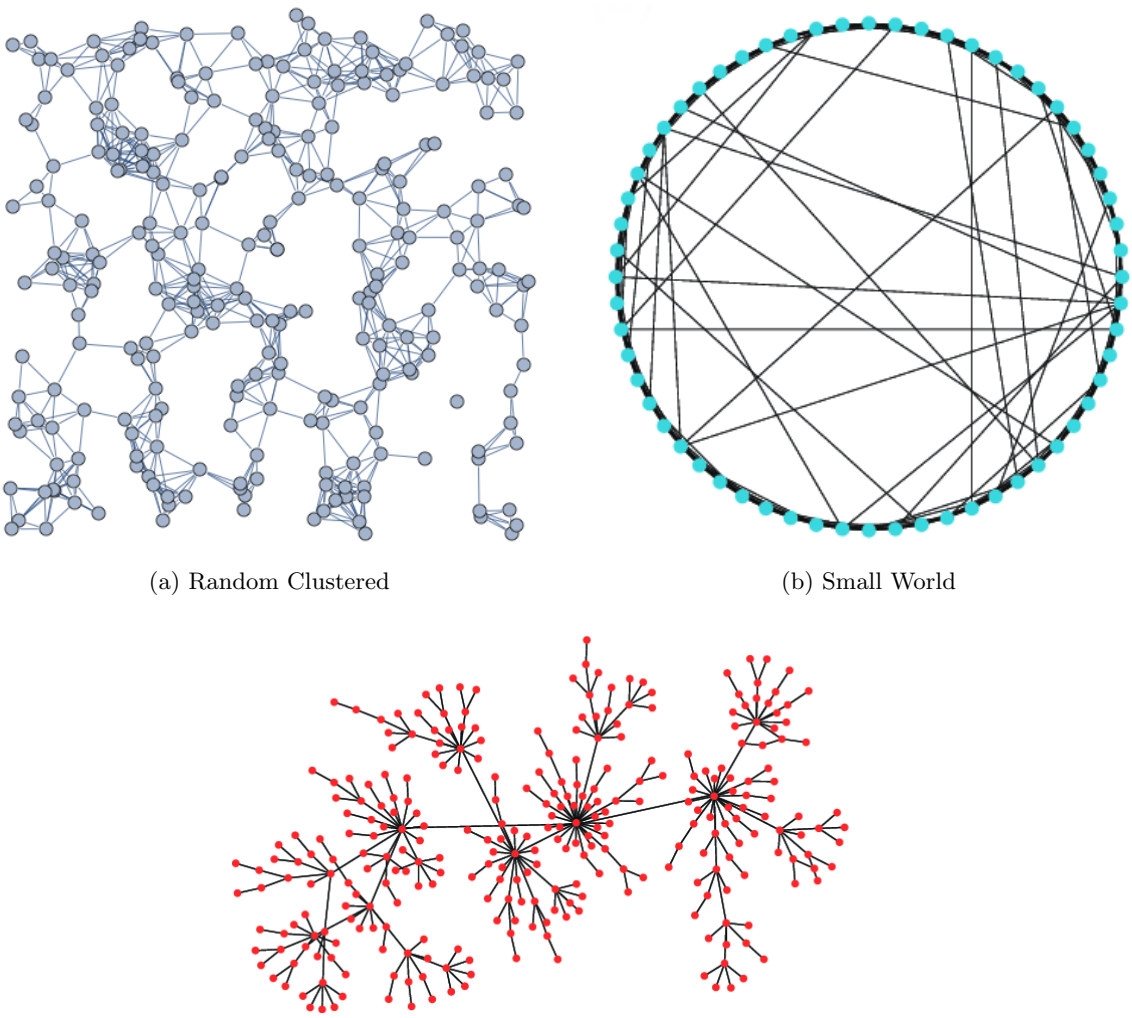
4.2 Background

The proposed model will explore the rise of radicalism within a specified geographic area. To carry this out, several aspects should be considered before going into detail such as network topologies and SNA techniques used to characterize the network structure.

The network consists on N nodes linked with edges as an undirected graph $G = (N, E)$ which can take different topologies based on the algorithm applied. In the context of influence spread, N can be viewed as the users of the social network. As it is explained in [34] complex networks statement where terrorist or criminal networks can be categorized into three types: random, small-world and scale-free.

- Random Clustered network is created based on proximity (Euclidean distance) between nodes.

- Scale Free network is a network type where a few nodes have lots of connections and all others have a few.
- Small World is type of mathematical graph in which most nodes are not neighbors of one another, but most nodes can be reached from every other by a few hops or steps.



(a) Random Clustered

(b) Small World

(c) Scale Free

Figure 4.1: Network Topologies

With the aim of consider the whole network within the model instead of each agent individually, some SNA techniques have been applied to compute metrics such as centrality which indicates the structural importance or prominence of a node in the network [35]. Within the model two different measures of node centrality are computed: (1) degree centrality and (2) betweenness centrality.

On one hand, degree centrality is defined as the number of adjacencies upon a node, which is the sum of each row in the adjacency matrix representing the network. It can be interpreted within social networks as a measure of immediate influence – the ability to infect others directly or in one time period [36]. This measure can be described as shown in Eq. 4.1.

$$C_D(i) = \sum_{j}^{N} x_{ij} \tag{4.1}$$

where i is the focal node, j are all other nodes and N is the number of nodes within the network [37].

On the other hand, betweenness centrality is defined of a node v as the sum of the fraction of all-pairs shortest paths that pass through v which is the same as the degree to which a node lies on the shortest path between two other nodes. It can be defined as shown in Eq. 4.2.

$$C_B(v) = \sum_{s,t \in G} \frac{\sigma(s,t|v)}{\sigma(s,t)} \tag{4.2}$$

where G is the set of nodes, $\sigma(s,t)$ is the number of shortest (s,t)-paths, and $\sigma(s,t|v)$ is the number of those paths passing through some node v other than s,t.

In the field of social networks, betweenness centrality represents a measure of the extent to which a node has control over information flowing between others. This means that nodes with the highest measure are those that facilitate communication to the nodes that it links.

Besides, with the purpose of decreasing the impact of radicals in the actual society, these nodes can be a target for breaking the radicalism diffusion within the network.

4.3 Overview

Three levels of analysis are widely accepted for the radicalization process [38]: *micro-level* (i.e. the individual level involving feelings of grievance, marginalization, etc.), *meso-level* (i.e. the social environment surrounding radicals and the population and lead to the formation of radical groups), and *macro-level* (i.e. impact of government policies, religion, media, including radicalization of the public opinion and political parties).

The model here proposed is focused on analyzing the macro-level, including limited aspects of the micro level (such as the vulnerability level).

Several aspects have been considered for modeling the radicalism growth at the meso level. First, the model considers the impact of *havens* [39] and *training areas* [40]. Havens, also known as sanctuaries, provide radical groups the possibility to obtain long term funding and serve the purposed of solidifying group cohesion. Terrorist training camps aim at providing indoctrination and teaching for terrorism and are distributed around the world. They foster group identity formation and group cohesion, and require geographical isolation and easy access to weapons.

The modeling of the radicalism spread involves population and places as it was discussed above. People can play two roles: (1) population as the people that can be radicalized and (2) terrorist that spread their message to locals and try to recruit civilians to join the terrorist network.

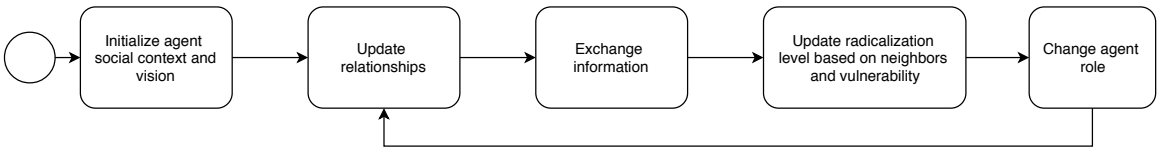


Figure 4.2: General workflow of the simulation

Based on a previous model proposed by Cummings [32], terrorists have little opportunities for effective training, planning, and other logistic necessities. Those areas are modeled by (1) training environments, which increase the influence to the nodes that are attached to them, and (2) heavens where people are safe. The nodes that are joined to havens get less influenced if the heaven is not radicalized, but it could get radicalized and its behavior will change.

For implementing the environment described, we will use four different agents that interact with each other.

- Spread model in charge of the information flow which determine the state of population. Each node contains a threshold where once reached, the node is marked as informed and it will pass from a civilian state to a radical state.
- Network model in charge of controlling spacial and social relations between population.
- Heavens model which will modify nodes vulnerability depending on heaven state as it is going to be explained below.
- Training areas model which will decrease neighboring nodes vulnerability.

The network consists on N nodes that have two coordinates, as since Geometric Graph Generators [31] are used, that position each node in a map. The edge between two nodes, indicates direct bidirectional communication between both of them.

4.4 Spread Model

All agents are assumed to have similar parameters but are heterogeneous in their representation. Within the spread model, each node develops its own belief about whether the information is valid by calculating weighted mean belief B_i from it neighbors, and combining that with its initial belief B_0, which is normalized between 0 and 1 [34]. In addition, in every step two agents will exchange information given a probability of interaction.

The mean belief is calculated given its own vulnerability and the neighbors influence as well as the information spread intensity (α) which is also normalized and consider how much information is exchanged in every step of the simulation.

$$B_e = \sum_{i=0}^{n} \frac{B_i \ D_i}{\sum_{j=0}^{n} D_j} \tag{4.3}$$

The node influence D_i parameter has been included in Eq. 4.3 – where n is the number of neighbors of the node – as the change in behavior that one person causes in another as a result of an interaction [41] measured as degree centrality which returns values that are normalized by dividing by the maximum possible degree in a simple graph $N-1$ where N is the number of nodes in G.

$$B_n = B_e \ \alpha + B_0 \ (1-\alpha) \quad ; \quad 0 \leq \alpha \leq 1 \tag{4.4}$$

As it was explained above, in Eq. 4.4 the parameter to indicate the information spread intensity is included. When its value is 0%, no information is exchanged and when it increases, the knowledge diffusion grows.

$$B_i = B_n \ N_v + B_0 \ (1-N_v) \quad ; \quad 0 \leq N_v \leq 1 \tag{4.5}$$

The node vulnerability (N_v) parameter is included in Eq. 4.5 as the extent to which individuals conform or adopt variable attributes such as opinions from their attached nodes. In other words, if $N_v = 1$, the node will be fully influenced by their connected nodes, where a value of $N_v = 0$, would mean it would not be influenced by connected nodes, so no change in

the network is expected. Thus, individuals who are merely sympathetic may be influenced to more extreme opinions by their friends after they join the terrorist network.

Once the mean belief developed by the agent reach the threshold, it is marked as informed and it will change its state from civilian to radical. Every agent in radical state will be only influenced by radical agents since the radical experience no restraining influence from non-radicals [42]. Furthermore, once an agent is in the radical state, the information spread intensity will begin to value 100%, as once you are radical the most information you get from another radical agents.

With the purpose of determining the most important nodes within the terrorist network, they are marked as *leaders* based on betweenness centrality [34].

As node vulnerability (N_v) was explained above, training areas and havens will modify this attribute depending on their status. Training areas will decrease the parameter from its neighbors, where a value of 1 for training area influence will make all its neighbors fully vulnerable. However, a value of 1 for haven influence will make invulnerable all its neighbors when the state of the haven is not radical. Nevertheless, once the haven is marked as radical, its behavior will be similar to training areas.

4.5 Network Model

The network model is in charge of controlling spacial and social relations takes into account that agents have the opportunity to interact with other agents. They select an agent to interact with according to a probability of interaction – different from the one mentioned above – based on two parameters: (1) social distance and (2) spatial proximity.

On one side, social distance (SD) take into account the fact that if two agents must cross many social links, then the probability should be low and vice versa. It computes it by finding the shortest path between the agents and then dividing one by the number of links in that path.

$$SD_{i,j} = \frac{1}{|A\ A_{i,j}|} \tag{4.6}$$

where $|A\ A_{i,j}|$ is the shortest path from i to j. When computing the social distance, each agent can only reach all those nodes that are withing its sphere of influence parameter. An agent can recognize and distinguish the closeness of other agents withing the sphere of influence, but it can't differentiate the closeness when the interacting agent is outside the perimeter.

On the other side, spatial proximity (SP) takes into account that two agents at the same location are more likely to talk than being in different locations. Some might argue that SP is not significant on the Internet age. However, in the terrorism domain, attending the same training area or the same location is a critical interaction indicator [30].

As Geometric Graph Generators returns coordinates normalized between 0 and 1, the probability of being at the same location will be computed as the inverse of the distance between two agents.

$$SP_{i,j} = (1 - |d_{i,j}|) \tag{4.7}$$

where $|d_{i,j}|$ is the distance between the nodes. Like in SD the probability is bounded by a sphere of influence parameter, in SP the probability will be bounded by a vision range parameter. All agents outside this perimeter will be unreachable by the current agent.

Once defined both parameters, we can compute the probability of interaction that it will be calculated as following.

$$P_{i,j}^{Interaction} = \omega_1 \; SD_{i,j} + \omega_2 \; SP_{i,j} \tag{4.8}$$

where ω_1 and ω_2 are the weights of SD and SP respectively with the purpose of customizing the environment.

None of the parameters will limit the probability of interaction. Thus, the candidate agents will be the sum of all the agents that are inside the perimeter of the sphere of influence or the vision range.

Thereby, an agent can establish a new way of communication with its candidate agents, so the probability of interaction is calculated between each agent and its candidate agents.

As it was explained, the aim of the model is trying to approach to the fact of the radicalism spread withing a specified geographic area. For that reason, in Table 4.1 all parameters of the simulation are detailed for representing a scenario as real as possible. Aside from all the parameters explained, the network can be modeled using one of the random network generation methods from NetworkX. It is also possible to control the ratio of each type of agent.

4.6 Configuration and visualization

All the variables previously presented, which are summarized in Table 4.1, are the input parameters of the model in its whole. These parameters should be included in the configuration file of the simulation next to other parameters such as the distribution of the agents at the beginning of the simulation, the network topology and its size, the duration of the simulation and the visualization parameters.

These parameters will allow the user to customize the environment of the simulation with the aim of approaching reality as much as possible, as it was discussed throughout the entire chapter.

The visualization is made using the tool described in Chapter 3 giving the option of launching several simulations with the same input parameters, which it could be interesting to compare simulations.

The output of the simulation is a graph $G = (N, E)$ where the nodes have defined its own attributes at each step, what makes possible to visualize the simulation among its development as well as analyze the final result and in each step.

The Spread Model explained in Sect. 4.4 is completely independent of the other ones. However, the Network Model described in Sect. 4.5 as it inherit the Spread Model, it needs its existence in the simulation.

The same occurs with both Training Area Model and Heaven Model. Those do not inherit the Spread Model one, but they need it to operate as they influence on population. Furthermore, both uses input parameters that are specified in the Spread Model inputs, but are not necessarily to be specified twice.

Keeping this in mind, Training Area Model and Heaven Model are independent between each other, and they could be optional when simulating.

Model	Name	Implication
Spread Model	information_spread_intensity	The amount of information exchanged in every step of the simulation.
	terrorist_additional_influence	Additional influence added to agents whom status is radical.
	min_vulnerability	The minimum vulnerability that an agent could have (*default 0*).
	max_vulnerability	The maximum vulnerability that an agent could have. The allocation of this parameter follows a continuous uniform distribution. The maximum value that this parameter can take is the unit.
	prob_interaction	The probability that two agents exchange information in one step.
Training Area	training_influence	The influence that a training area applies to its neighbors.
Haven	haven_influence	The influence that a haven applies to its neighbors.
Network Model	sphere_influence	The maximum number of social links that an agent can cross for a new interaction.
	vision_range	The range on the spatial-route network specifying the maximum distance an agent can move for a new interaction.
	weight_social_distance	The weight of social distance (SD) to calculate the interaction probability.
	weight_link_distance	The weight of spatial proximity (SP) to calculate the interaction probability.

Table 4.1: Simulation input parameters

CHAPTER 5

Experimental results

In this chapter, the results obtained by the simulation are presented, contrasted and evaluated. The implementation of the agent-based model of radicalization is introduced in Sect. 5.1. In Sect. 5.2 the evaluation of the model is discussed.

5.1 Implementation

The model described in Chapter 4 has been implemented using the Soil simulator. As it was explained in Sect. 2.1, Soil provides the option of modeling the agents using a Finite State Machine (FSM).

The first step is defining the states of the FSM of the spread model. Those will be the three states of a population agent which are: (1) civilian, (2) terrorist and (3) leader. Then, the transitions from one state to another are defined depending on a boolean condition as it is shown in Fig. 5.1.

When an agent is marked as *informed*, it will change its state to terrorist, while if the agent is in the terrorist state, it will change to the leader state when the betweenness centrality computed to that agent is the highest inside its sphere of influence. Each state

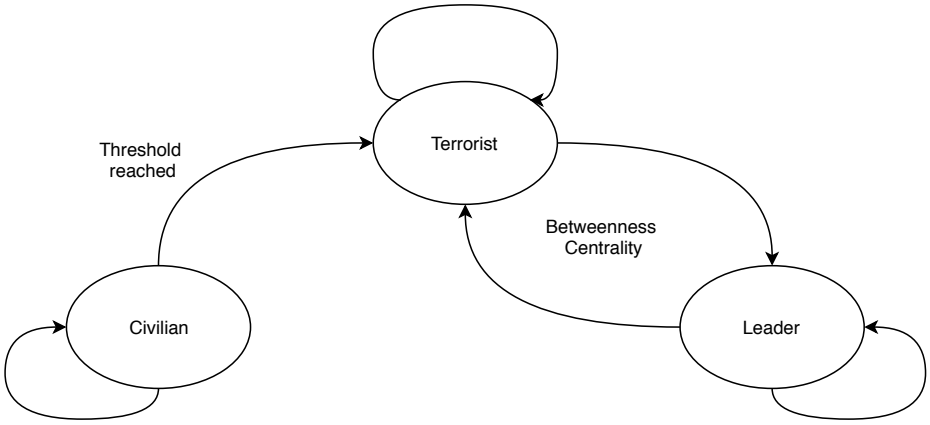
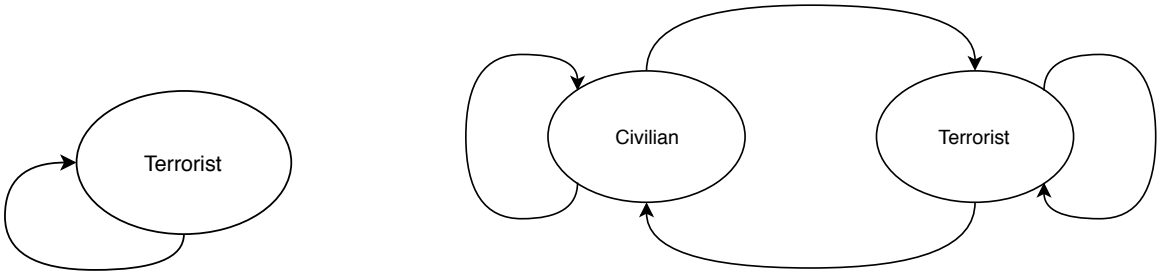


Figure 5.1: Spread model FSM

execute its own function that defines the behavior of the agent in that state.

With the purpose of modeling every agent in the same way, the network model redefine the states adding the fact of the spatial and social relations. The training areas model and the heaven model are also model with a FSM, but in these cases there are fewer states as it is shown in Fig. 5.2, with only one in the training area model since it is always in the terrorist state.



(a) Training area model

(b) Heaven model

Figure 5.2: Heaven and training area model FSMs

The simulation model has been implemented following the structure shown in the UML class diagram in Fig. 5.3 and consists on five main elements. The `soil.agents.FSM` class is the one provided by Soil and it is in charge of controlling the transitions between the different states. The other four remaining are the four agents. Each agent has its own class with its input parameters as attributes.

As it can be noticed in the class diagram, the training area and heaven models are joined to the spread model because both interact with population modifying internal attributes depending on their status.

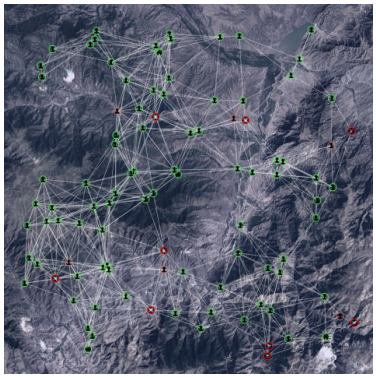
Figure 5.3: UML Class diagram

Every agent exchange information several times during the simulation and every portion of time is known as *step*. In order to present a simulation flow, in Fig. 5.4 an example is presented using the input values shown in Table 5.1 for a total of 100 nodes and simulation time of 150 steps.

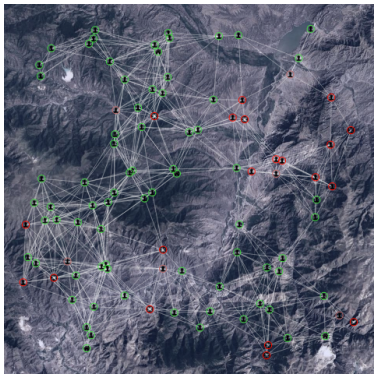
The network topology used is a random clustered network where nodes are connected when two of them are within 0.2 of distance. The distribution of agents has been made considering 80% of civilian agents, 10% of terrorist agents and 10% of heavens and training areas distributed equally.

For running the simulation, the web application described in Chapter 3 has been used. Once the configuration is uploaded to the application, it shows up a set of sliders from where the user can change the environment variables as it shown in Fig. 5.5.

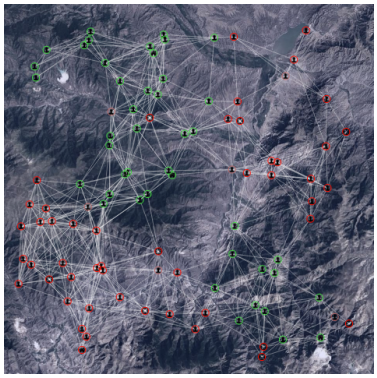
The user does not need to do any additional configuration because the web application is the one in charge of selecting the type of input depending on its value. Thus, the *sphere of influence* is not the same as the others because it is an int variable instead of a float one.



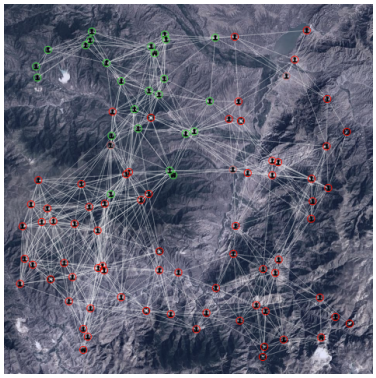
(a) $t = 0$



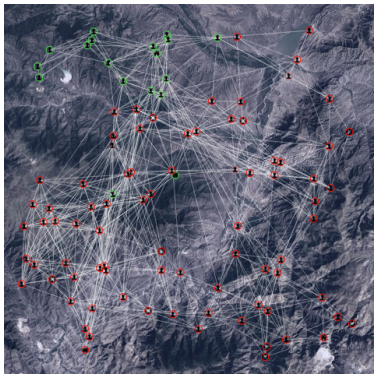
(b) $t = 25$



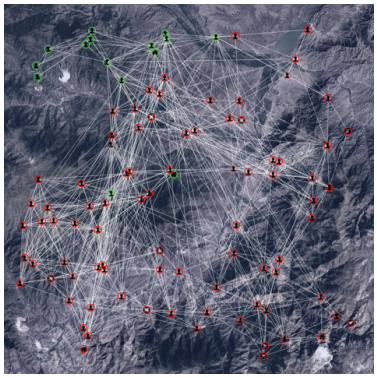
(c) $t = 50$



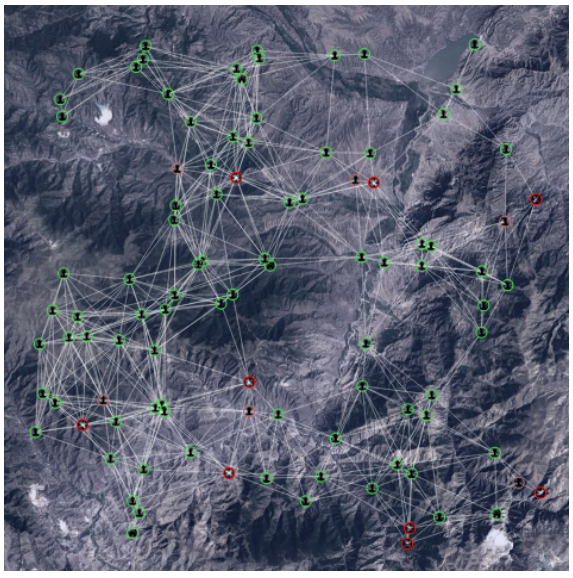
(d) $t = 75$



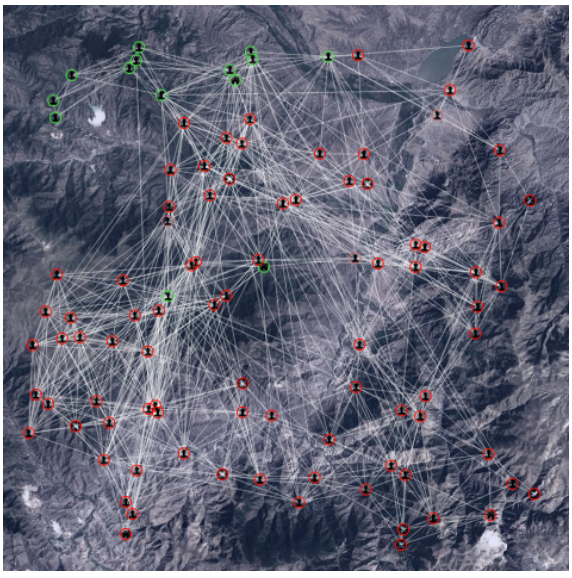
(e) $t = 100$



(f) $t = 125$



(g) Initial state



(h) Final state

Figure 5.4: Simulation flow

Parameter	Initial value	Bounds
Information spread intensity	0.70	0.0 – 1.0
Terrorist additional influence	0.035	0.0 – 1.0
Maximum vulnerability	0.70	0.0 – 1.0
Probability of interaction	0.50	0.0 – 1.0
Training area influence	0.20	0.0 – 1.0
Heaven influence	0.20	0.0 – 1.0
Vision range	0.30	0.0 – 1.0
Sphere of influence	2.0	$1 - N$
Weight of social distance	0.035	0.0 – 1.0
Weight of link distance	0.035	0.0 – 1.0

Table 5.1: Simulation input values

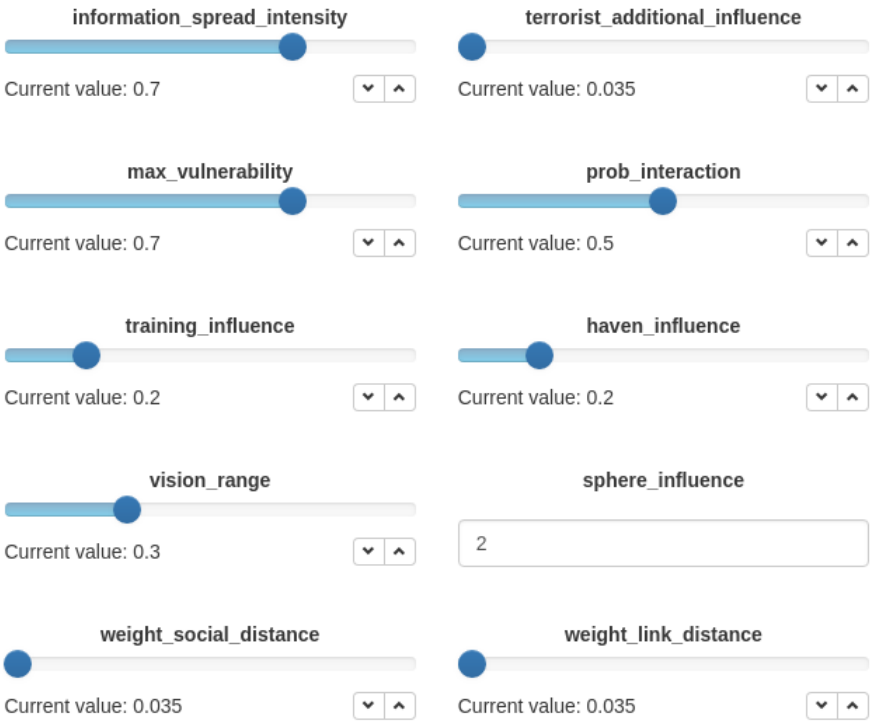


Figure 5.5: Set of sliders for configuring a simulation

5.2 Evaluation

Sensitivity Analysis (SA) is an important tool when developing numerical simulation models. It allows studying how the uncertainty in the output of a model can be apportioned to different sources of uncertainty in the model input. It may be used to determine the most contributing input variables to an output behavior as the non-influential inputs, or ascertain some interaction effects within the model [22].

The sensitivity of each input is often scored by a numeric value, called the *sensitivity index*, which come in several forms.

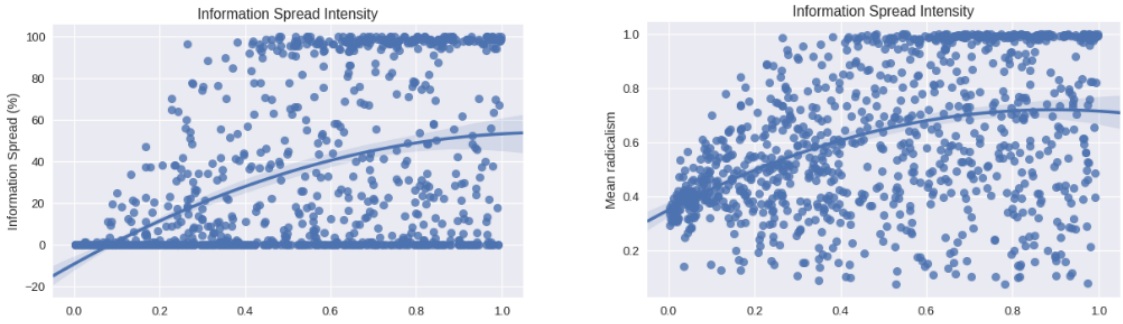
- First-order indices measures the contribution to the output variance by a single model input alone.
- Second-order indices measures the contribution to the output variance caused by the interaction of two model inputs.
- Total-order index measures the contribution to the output variance caused by a model input, including both its first-order effects (the input varying alone) and all higher-order interactions.

The evaluation have been divided in several parts: (1) the network model has been evaluated on his own, perturbing only its input parameters, (2) network topologies and agent distribution have been evaluated separately, and (3) training areas and havens the same. Besides, there are two primary model outputs of interest in the sensitivity analysis: (1) the **radicalism diffusion** computed as the percentage of agents that have been radicalized during the simulation from those who were not radical at the beginning and (2) the **mean radicalism** within the network computed as the average radicalism of all the agents.

The model has been evaluated using two different SA techniques. The first one is a local approach known as One-at-Time (OAT) design, where each input is varied while fixing the others [43]. To bring about this method, 1,000 simulations have been launched with small perturbations on the model input.

With the results of all the simulations, the output has been computed and scattered. Each of the scatter plots shows all the model outputs on the y-axis, but re-ordered by the relationship to the input variable on the x-axis.

Scatter plots can tell quite a lot about the relationship between the model inputs and outputs. As it can be noticed in Fig. 5.6, the input variable “information spread intensity” increase the output value while increasing its value for both outputs.



(a) Radicalism diffusion

(b) Mean radicalism

Figure 5.6: Scatter plots for information spread intensity

The other method applied is the Morris method [44] that is referred to as "global sensitivity analysis" that in contrast to local sensitivity analysis, it considers the whole variation range of the inputs [23]. This method is computed using the SALib [26] library for Python.

The Morris method allows classifying input variables into three types: (a) negligible, (b) linear and additive, or (c) non-linear or involved in interactions with other factors [45]. The experimental plan is composed of individually randomized OAT experiments. The input space is discretized in a d–dimensional grid with n levels by input. Eq. 5.1 define $E_j^{(i)}$ as the elementary effect of the j–th variable obtained at the i–th repetition [23].

$$E_j^{(i)} = \frac{f(X^{(i)} + \Delta e_j) - f(X^{(i)})}{\Delta} \tag{5.1}$$

where Δ is a predetermined multiple of $\frac{1}{(n-1)}$ and e_j is a vector of zeros, but with a unit as its j–th component.

The sensitivity measures proposed by Morris [44], μ^* and σ, are respectively the mean and the standard deviation of the elementary effects. Morris suggests sampling r trajectories of $(k+1)$ points in the input space, each providing k elementary effects, one per input factor. The total cost of the experiment is thus $r(k+1)$. These indices are obtained as it is defined in Eq. 5.2.

$$\mu_j^* = \frac{1}{r}\sum_{i=1}^{r} |E_j^{(i)}| \qquad \sigma_j = \sqrt{\frac{1}{r}\sum_{i=1}^{r}\left(E_j^{(i)} - \frac{1}{r}\sum_{i=1}^{r} E_j^{(i)}\right)^2} \tag{5.2}$$

The interpretation of the indices is the following [23].

- μ_j^* is a measure of influence of the j–th input on the output. The larger μ_j^* is, the more the j–th input contributes to the dispersion of the output.

- σ_j is a measure of non-linear and interaction effects of the j–th input. If σ_j is small, elementary effects have low variations on the support of the input. Thus, the effect of a perturbation is the same all along the support, suggesting a linear relationship between the studied input and the output. On the other hand, the larger σ_j is, the less likely the linearity hypothesis is. Thus, a variable with a large σ_j will be considered having non-linear effects, or being implied in an interaction with at least one other variable.

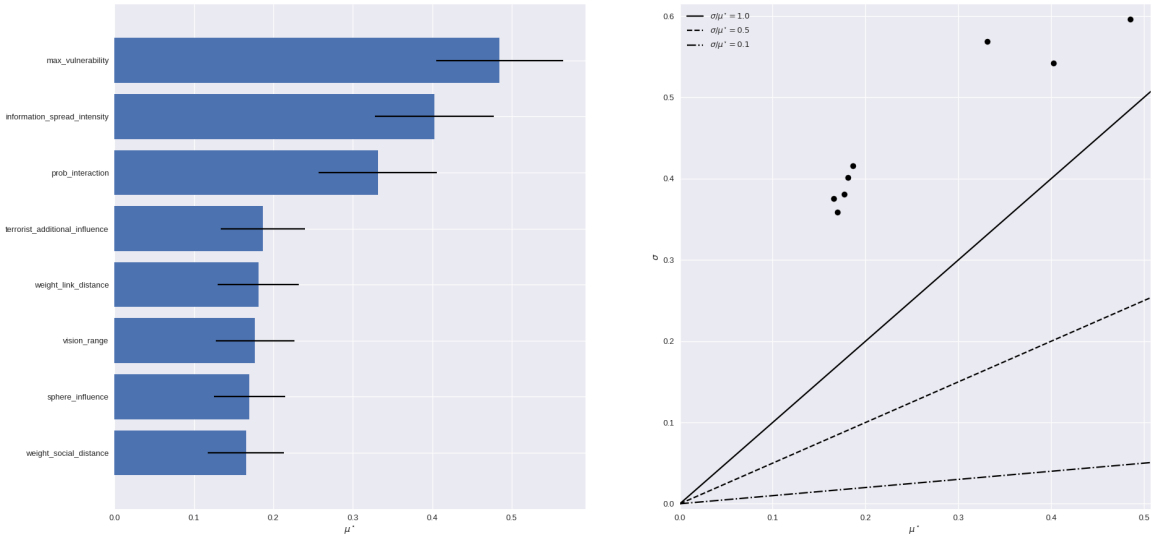
The network model has been evaluated on its own for 200 trajectories what makes a total of 1,800 simulations because of its 8 input variables. The evaluation has been also made considering the same topology, random clustered, and the same agent distribution: 90% of civilians and 10% of terrorists distributed throughout the network.

In Table 5.2 the Morris indices are detailed for the network model and both outputs. Fig. 5.7 plots the results on the graph (μ^*, σ) and identifies the maximum vulnerability and the probability of interaction as the strongest influence on the radicalism diffusion and mean radicalism respectively.

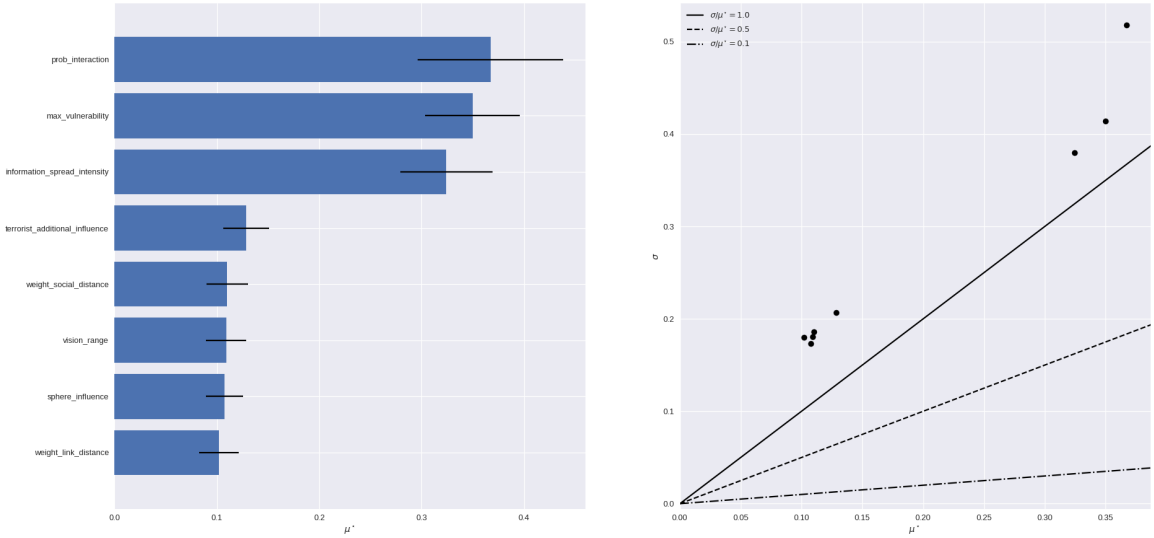
As it can be noticed, variables seems to have strong influence with non-linear effects. However, this may make sense if it is taken into account that the phenomena being modeled involve non-linear relationships.

Parameter	**Radicalism diffusion**			**Mean radicalism**		
	μ	μ^*	σ	μ	μ^*	σ
information_spread_intensity	0.392	0.402	0.541	0.252	0.324	0.379
terrorist_additional_influence	0.092	0.186	0.415	0.036	0.128	0.206
max_vulnerability	0.466	0.484	0.596	0.243	0.349	0.413
prob_interaction	0.268	0.331	0.568	0.320	0.367	0.517
vision_range	−0.001	0.176	0.380	0.019	0.109	0.180
sphere_influence	0.005	0.169	0.358	0.006	0.107	0.173
weight_social_distance	0.003	0.165	0.375	−0.004	0.110	0.186
weight_link_distance	−0.012	0.181	0.401	0.007	0.101	0.179

Table 5.2: Morris indices for network model



(a) Radicalism diffusion



(b) Mean radicalism

Figure 5.7: Morris method results representation for network model

In Table 5.3 Morris indices are detailed for Scale free and Small world topologies. The output taken into account is the radicalism diffusion within the network.

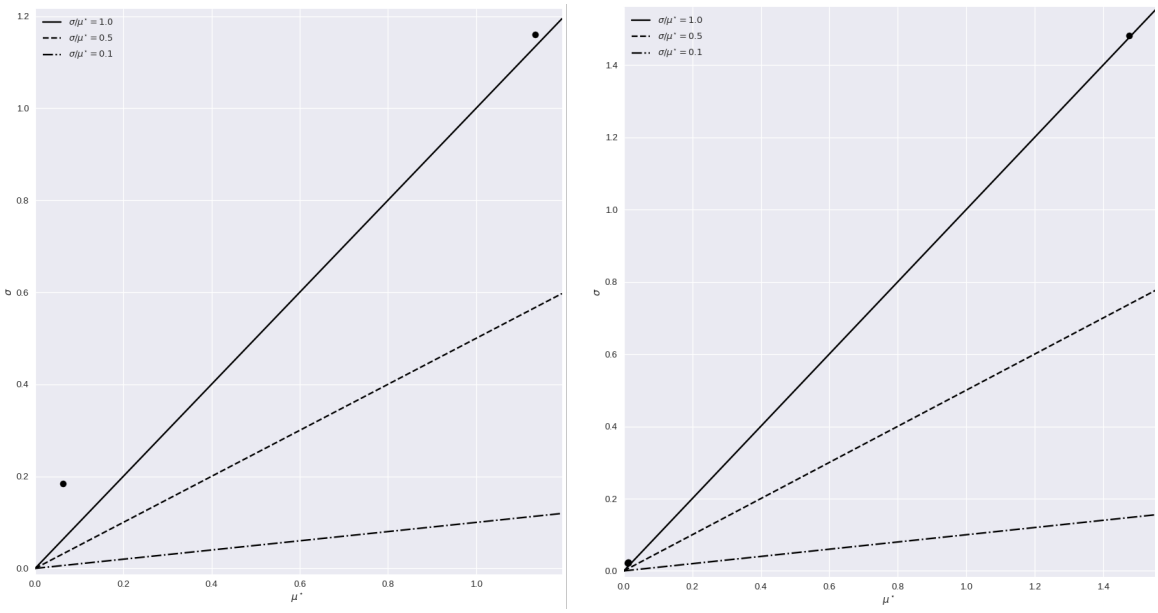
Morris indices for both topologies have similarities, as the weight of the radical agent distribution is the most influential parameter for both outputs as it can be noticed in Fig. 5.8 which plots the results.

Parameter	**Scale free**			**Small world**		
	μ	μ^*	σ	μ	μ^*	σ
nodes	0.027	0.063	0.185	−0.001	0.011	0.022
civilian_weight	−0.067	1.134	1.161	0.053	1.475	1.482
joints	–	–	–	0.006	0.013	0.024
probability	–	–	–	−0.002	0.012	0.023

Table 5.3: Morris indices for scale free and small world

Nevertheless, the size of the network has no influence on the two model outputs. Besides, joints and probabilities, which are two necessary variables for computing the algorithm of small world, have no influence either in the output.

The furthest point is the weight of the radical agents and it has a linear influence on the output because σ_j and μ_j^* have the same order of magnitude.



(a) Scale Free Output

(b) Small World output

Figure 5.8: Morris method results representation for radicalism diffusion

CHAPTER 6

Conclusions and future work

In this chapter we will describe the conclusions extracted from this project in Sect. 6.1, achievements in Sect. 6.2 and finally in Sect. 6.3 the thoughts about future work.

6.1 Conclusions

In this thesis a web application has been developed, where Agent-based Social Simulations could be visualized. The application allows the user to analyze the simulation in real time as well as launching new simulations which are configured from the web.

The application uses D3.js for representing the results and Python for deploying the server and executing the simulations. For the development of the model, Python has also been used as Soil uses this language for developing new languages due to its recently increased popularity.

This software could be used for other projects whose purpose is ABSS because visualizing the results will give the opportunity to analyze them in an easier and more intuitive way because it will be simpler and faster to find solutions for possible problems or to reach conclusions.

With the purpose of proving the features of the software, a multi-agent social model which tries to simulate the growth of radicalism in a society has also been developed using SNA techniques. The motivation comes from the growth of radical ideologies and spread of terrorist organizations. Trying to understand the spread of information between members or even non-members of the organization could help with the purpose of destabilizing them.

Understanding radicalization roots is a first step for being able to define and apply suitable counter-terrorism measures. There are many challenges for analyzing terrorism networks, given the lack of public datasets and the sensibility of this information. Nonetheless, the application of agent based social simulation is an effective technique for modeling non-linear adaptive systems, and they enable analyzing and validating social theories of the radicalization process.

The proposed agent-based model is built around two main concepts, the spread model and the network model. While the first defines the specific behavior of every agent, the second is in charge of managing agent relationships. This approach has been applied for modeling terrorist growth. The proposed model is focused on analyzing the impact of the information exchange and environmental radicalization in the radicalization process. The evaluation and analysis of the simulation results provides insight regarding the importance of the simulation parameters, including the network characteristics.

It should be noted that this project has been accepted in the 6th International Workshop on Engineering Multi-Agent Systems (EMAS 2018) which is hold in Stockholm, Sweden:

> *Tasio Méndez, J. Fernando Sánchez-Rada, Carlos A. Iglesias and Paul Cummings (2018). A Model of Radicalization Growth using Agent-based Social Simulation. In Proceedings of EMAS 2018. Stockholm, Sweden [46].*

EMAS 2018 aims to gather researchers and practitioners working in the domains of agent-oriented software engineering, programming multi-agent systems, declarative agent languages and technologies, machine learning, and AI in general to present and discuss their research and emerging results in MAS engineering.

6.2 Achieved goals

In this section, the goals achieved during the development of the project are described.

Simulate the radicalism and the spread of information through a network. Soil which is an agent-based simulator provides the necessary classes for developing new

agents. Thus, the model consists on different agents that interact with each other and represents different roles in the society.

Visualize the results of the simulation. After a deep analysis of the current libraries for representing graphs, D3.js was the chosen one due to its extensive documentation as well as because it provides a scene graph that can be inspected for debugging. D3 is also extremely fast and supports large datasets and dynamic behaviors for interaction and animation.

Interact with the simulation in real time. JavaScript is a lightweight interpreted, object-oriented language and a powerful scripting language that support controlling the web page behavior. Thus, the dynamic graph which is implemented with D3 could have been modified depending on user actions.

Extract data from the results. Soil provides such a wide range of methods to access data from a simulation and D3 is focused on graphs. Grouping the features of both of them, the extraction of the data of the graph has not been such a difficult task as it was supposed at the beginning of this thesis.

Show data correctly for its research. Once the data has been extracted, it has been presented using different charts so a summary of the simulations is provided at a glance in one of the views of the application.

6.3 Future work

In this section, some ideas for new features or improvement of this software are presented. These ideas are for the web application and the radicalism growth model.

Adding new agents. A new agent could be added which purpose will be to break the information flow. The infiltrator will find nodes to attack in order to inhibit the growth of the terrorist group. The infiltrator has the option of looking for leaders, and removing them when they are found. Another option is that the infiltrator provides misinformation to disrupt the flow of critical information. Both options should be considered within this model in order to analyze which one is better. Besides, the algorithm to find the nodes should be also considered so as to find the optimal solution.

Adding new statistics. The current version shows some statistics about the simulation. However, statistics between trials could be added. When running a simulation, the

user can specify the number of trial to run and provide statistics between them in order to realize a deeper analysis could be interesting.

Adding SNA techniques from the web. Another feature that could be added to the application in the future is the option of computing different measures related to the graph in a specific instant of time (i.e. betweenness centrality, degree centrality, density, cohesion, community detection, etc.).

Increase visualization parameters. The current visualization options are changing background to position the nodes, and change its shape and colors. Some other configuration parameters could be added as changing the layout of the graph if nodes do not have coordinates or changing the size of the node depending on its attributes.

Develop new models from the browser. The current version needs the model to be stored in the server as it is used by the simulator module. However, it could be a good option to have an online text editor to develop new models directly from the browser and try them in the application.

Configure the model from the application. Following the idea of an online text editor, it could be interesting to provide the option of writing the configuration file directly in the browser instead of uploading it every time.

APPENDIX A

Impact of this project

This appendix reflects, quantitatively or qualitatively, on the possible impact (positive or negative, direct or indirect, current or future) and the responsibilities related to this thesis. It considers the social (Sect. A.1), economic (Sect. A.2) and environmental (Sect. A.3) impact as well as the ethical implications in Sect. A.4.

A.1 Social impact

Research works on political terrorism began in the early 1970s. These works were focused on collecting empirical data and analyzing it for public policy purpose. Many scholars, government analysts and politicians point out that since the mid 1990s terrorism has changed. This "new" for of terrorism is often motivated by religious beliefs and it is more fanatical, deadly, and pervasive. It also differs in terms of goals, methods and organization as it was discussed in Sect. 1.1.

Since September 11, terrorism has become a global problem and a worldwide concern. In fact, terrorist activities are one of the most important worries for those societies that suffer from this problem.

Terrorism has a huge impact on society. It produces fear in places where an act of terrorism has occurred. It also has a negative impact on tourism demand [47] which is linked to the economic income of these areas.

Besides, in the last year a new concept has increased its popularity due to the radical acts in the middle east countries, which is *Islamophobia* referring to a dread or hatred of Islam and therefore a fear or dislike of Muslims. This have had a huge impact in the Muslim society who have been involved in acts of xenophobia in some areas.

As terrorism is the consequence of a complex process of radicalization, the aim of this project is to understand this process and anticipate radical conversion. This project can be used by security forces with the purpose of preventing attacks and being one step ahead from terrorist intentions and movements.

A.2 Economic impact

As it was discussed in the previous section, terrorist activities has a negative impact on tourism which is linked to the economic income of a specific area. It also affects the allocation decision of firms investing money in real foreign assets. Terrorists can quite easily attack and damage foreign owned firms, seriously disrupting their activities.

Another impact is the stock markets whose prices are a potentially informative measure of the economic damage to terrorism as well as the urban economy [48]. The terrorist attacks on September 11, 2001, destroyed 13 million square feet of real state. One estimate of the real and human capital costs ranges from 25 to 60 billion dollars as it is explained in [49]. However, the indirect costs may be substantial. They consist of the induced cost of doing business, for instance longer waiting at airports, higher friction and transactions costs in international trade, as well as the cost of the military and civilian resources to fight terrorism.

Regarding the stock markets, expected profits are lower if security measures increase the costs of production and doing business, and if consumers' fear reduces demand, like in the airline industry.

As it can be noticed, terrorism have a huge impact in the economy of a specific country. Trying to anticipate such activities by understanding the radicalization process and how terrorists spread their knowledge is a point to start.

A.3 Environmental impact

The environmental impact of this thesis comes from the environmental impact of terrorism. The aim of the project is to fight against terrorism providing a model for simulating the radicalism and diffusion and a web application for visualizing the results.

ABSS does not have any environmental impacts. However, the terrorism is evolving as the new technologies evolve. Technologies evolve in all areas: computing, medicine, assistance to people, improvement of our lives, but it also evolves taking into account negative applications such as weapons. There is an increasing threat that terrorism evolves to chemical and biological weapons that can derive in an environmental disaster.

A.4 Ethical implications

ABSS is based on modeling social behaviors. The ethical implication of this project is basically the ethical implication of collecting data for modeling those behaviors.

However, the model developed throughout this project has been made using data collected from researches as there is a lack of information about this field. Thus, the most important ethical implications about this project is the fight against terrorism which is a worldwide concern.

Another ethical implication is the research ethics, they are moral considerations that influence the decisions made during the research process. This is because one of the most important considerations that the researcher must have is that the participants are humans who must always maintain their dignity despite research and its results. There should be a set of ethical rules that should be followed in investigations with humans.

APPENDIX B

Economic budget

This appendix details an adequate budget to bring about the project proposed covering aspects such as material execution expenses (Sect. B.1), performers' fees (Sect. B.2), licenses (Sect. B.3) and taxes (Sect. B.4) involved. Most costs are derived from the salaries of the developers, although some costs have been provoked by hardware needs.

B.1 Physical resources

This project provides a web application that could be run in a server or in a personal computer as it is launched in the port `8080` by default but it could be customized. Thus, during the development of the system, the recommended requirements for a computer that would run it without any kind of issue, independently of being a personal computer or a server, are approximately the following:

- **RAM**: 8 to 16 GB
- **Hard disk**: 20 GB SSD space
- **CPU**: 4 to 8 CPU cores at 2.80+GHz

On average, a computer with such features costs around 900 €. In case of deploying a server, expenses for the domain should be considered which costs around 100 € a year.

B.2 Human resources

In this section, the project is quantified by taking into account the hours employed to develop this software as well as the average salary of an Engineer in order to find the cost of developing the project.

Considering 5 months of work and 22 working days in a month in a part-time schedule (4 hours per working day), the cost in hours of this project is estimated to be around 440 hours. The salary of an engineer developing this software is estimated to be the standard salary of an internship for such jobs: 450 € per month. Therefore, the total cost for developing the software is estimated to be around 2,250 €.

B.3 Licenses

All the software used in the development of this project is open source software what means that it does not have any expenses in licenses.

However, one of the libraries discussed in Sect. 2.4 is not open source. Thus, in case of upgrading the software and adding new libraries for graph visualization such as Linkurious which identify hidden insights on graphs, a license should be considered although the price is not published by the company that owns the rights. Nevertheless, for the features presented in this thesis any license is needed, so expenses to take into account are null.

B.4 Taxes

One of the possible actions we could take once we have implemented the software is selling it to another company. In that case, some taxes derived from the sale of the software should be considered.

According to [50], there is a tax of 15% over the final price of the product, as regulated by the Statue 4/2008 of the Spanish law. Besides, if we want to sell the software to a foreign company, more taxes should be considered but this is beyond the scope of this project.

Bibliography

[1] Martha Crenshaw. The psychology of terrorism: An agenda for the 21st century. *Political psychology*, 21(2):405–420, 2000.

[2] Alexander Spencer. Questioning the concept of 'new terrorism'. *Peace, Conflict and Development*, pages 1–33, 2006.

[3] Oliver Gruebner, Martin Sykora, Sarah R Lowe, Ketan Shankardass, Ludovic Trinquart, Tom Jackson, SV Subramanian, and Sandro Galea. Mental health surveillance after the terrorist attacks in paris. *The Lancet*, 387(10034):2195–2196, 2016.

[4] David Tucker. What is new about the new terrorism and how dangerous is it? *Terrorism and Political Violence*, 13(3):1–14, 2001.

[5] Jesús M. Sánchez, Carlos A. Iglesias, and J. Fernando Sánchez-Rada. Soil: An Agent-Based Social Simulator in Python for Modelling and Simulation of Social Networks. In Bajo J. Vale Z. Demazeau Y., Davidsson P., editor, *Advances in Practical Applications of Cyber-Physical Multi-Agent Systems: The PAAMS Collection*, volume 10349 of *LNAI*, pages 234–245. PAAMS 2017, Springer Verlag, June 2017.

[6] Kevin Ryczko, Adam Domurad, Nicholas Buhagiar, and Isaac Tamblyn. Hashkat: large-scale simulations of online social networks. *Social Network Analysis and Mining*, 7(1):4, 2017.

[7] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[8] Gabor Csardi and Tamas Nepusz. The igraph software package for complex network research. *InterJournal, Complex Systems*, 1695(5):1–9, 2006.

[9] Diego Blanco-Moreno, Rubén Fuentes-Fernández, and Juan Pavón. Simulation of online social networks with krowdix. In *Computational Aspects of Social Networks (CASoN), 2011 International Conference on*, pages 13–18. IEEE, 2011.

[10] Joshua O'Madadhain, Danyel Fisher, Padhraic Smyth, Scott White, and Yan-Biao Boey. Analysis and visualization of network data using jung. *Journal of Statistical Software*, 10(2):1–35, 2005.

[11] David Gilbert. The jfreechart class library. *Developer Guide. Object Refinery*, 7, 2002.

[12] Klaus Muller and Tony Vignaux. Simpy: Simulating systems in python. *ONLamp. com Python Devcenter*, 2003.

[13] Michael Dory, Allison Parrish, and Brendan Berg. *Introduction to Tornado: Modern Web Applications with Python*. O'Reilly Media, Inc., 2012.

[14] Victoria Pimentel and Bradford G Nickerson. Communicating and displaying real-time data with websocket. *IEEE Internet Computing*, 16(4):45–53, 2012.

[15] Andrew Wessels, Mike Purvis, Jahrain Jackson, and Syed Rahman. Remote data visualization through websockets. In *Information Technology: New Generations (ITNG), 2011 Eighth International Conference on*, pages 1050–1051. IEEE, 2011.

[16] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 data-driven documents. *IEEE transactions on visualization and computer graphics*, 17(12):2301–2309, 2011.

[17] Nick Qi Zhu. *Data visualization with D3. js cookbook*. Packt Publishing Ltd, 2013.

[18] Casey Reas and Ben Fry. *Getting Started with Processing: A Hands-On Introduction to Making Interactive Graphics*. Maker Media, Inc., 2015.

[19] Dmitry Baranovskiy. Raphael–javascript library, 2010.

[20] Graciously Kharumnuid Swarup Roy. Effectiveness of javascript graph visualization libraries in visualizing gene regulatory networks (grn), 04 2015.

[21] Kevin Sheppard. Introduction to python for econometrics, statistics and data analysis. *Self-published, University of Oxford, version*, 2, 2012.

[22] Andrea Saltelli, Marco Ratto, Terry Andres, Francesca Campolongo, Jessica Cariboni, Debora Gatelli, Michaela Saisana, and Stefano Tarantola. *Global sensitivity analysis: the primer*. John Wiley & Sons, 2008.

[23] Bertrand Iooss and Paul Lemaître. A review on global sensitivity analysis methods. In *Uncertainty management in simulation-optimization of complex systems*, pages 101–122. Springer, 2015.

[24] Andrea Saltelli, Karen Chan, E Marian Scott, et al. *Sensitivity analysis*, volume 1. Wiley New York, 2000.

[25] Will Usher. Using sensitivity analysis to interrogate models. Environmental Change Institute, University of Oxford. `http://mybinder.org/repo/salib/satut`. Accessed May 5, 2018.

[26] Jon Herman and Will Usher. Salib: an open-source python library for sensitivity analysis. *The Journal of Open Source Software*, 2(9), 2017.

[27] Seth Tisue and Uri Wilensky. Netlogo: A simple environment for modeling complexity. In *International conference on complex systems*, volume 21, pages 16–21. Boston, MA, 2004.

[28] Mathieu Bastian, Sebastien Heymann, Mathieu Jacomy, et al. Gephi: an open source software for exploring and manipulating networks. *Icwsm*, 8:361–362, 2009.

[29] J Sergio Zepeda and Sergio V Chapa. From desktop applications towards ajax web applications. In *Electrical and Electronics Engineering, 2007. ICEEE 2007. 4th International Conference on*, pages 193–196. IEEE, 2007.

[30] Il-Chul Moon and Kathleen M Carley. Modeling and simulating terrorist networks in social and geospatial dimensions. *IEEE Intelligent Systems*, 22(5), 2007.

[31] Mathew Penrose. *Random geometric graphs*. Oxford university press, 2003.

[32] Paul Cummings and Chalinda Weerasinghe. Modeling the characteristics of radical ideological growth using an agent based model methodology. In *MODSIM World*, 2017.

[33] Miller McPherson, Lynn Smith-Lovin, and James M Cook. Birds of a feather: Homophily in social networks. *Annual review of sociology*, 27(1):415–444, 2001.

[34] Paul Cummings. Modeling the characteristics of radical ideological growth using an agent based model methodology. Master Thesis, George Mason University, 2017.

[35] Stephen P Borgatti, Ajay Mehra, Daniel J Brass, and Giuseppe Labianca. Network analysis in the social sciences. *science*, 323(5916):892–895, 2009.

[36] Stephen P Borgatti. Centrality and network flow. *Social networks*, 27(1):55–71, 2005.

[37] Tore Opsahl, Filip Agneessens, and John Skvoretz. Node centrality in weighted networks: Generalizing degree and shortest paths. *Social networks*, 32(3):245–251, 2010.

[38] Rositsa Dzhekova, N Stoynova, A Kojouharov, M Mancheva, D Anagnostou, and E Tsenkov. Understanding radicalisation. review of literature. *Center for the Study of Democracy, Sofia*, 2016.

[39] Ari Jean-Baptiste. *Terrorist Safe Havens: Towards an Understanding of What They Accomplish for Terrorist Organizations*. PhD thesis, University of Kansas, 2010.

[40] James JF Forest. Terrorist training centers around the world: A brief review. *The Making of a Terrorist: Recruitment, Training and Root Causes*, 2, 2005.

[41] Lisa Rashotte. Social influence. *The Blackwell encyclopedia of sociology*, 2007.

[42] Michael Genkin and Alexander Gutfraind. How do terrorist cells self-assemble: Insights from an agent-based model of radicalization. Technical report, SSRN, July 2011.

[43] Andrea Saltelli and Paola Annoni. How to avoid a perfunctory sensitivity analysis. *Environmental Modelling & Software*, 25(12):1508–1517, 2010.

[44] Max D Morris. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33(2):161–174, 1991.

[45] Francesca Campolongo, Jessica Cariboni, and Andrea Saltelli. An effective screening design for sensitivity analysis of large models. *Environmental modelling & software*, 22(10):1509–1518, 2007.

[46] Tasio Méndez, J. Fernando Sánchez-Rada, Carlos A. Iglesias, and Paul Cummings. A Model of Radicalization Growth using Agent-based Social Simulation. In *Proceedings of EMAS 2018*, Stockholm, Sweden, July 2018.

[47] Abraham Pizam and Ginger Smith. Tourism and terrorism: A quantitative analysis of major terrorist acts and their impact on tourism destinations. *Tourism Economics*, 6(2):123–138, 2000.

[48] Bruno S Frey, Simon Luechinger, and Alois Stutzer. Calculating tragedy: Assessing the costs of terrorism. *Journal of Economic Surveys*, 21(1):1–24, 2007.

[49] Gary Becker and Kevin Murphy. Prosperity will rise out of the ashes. *Wall Street Journal*, 29:A22, 2001.

[50] Francisco de la Torre Díaz. La tributación del software en el IRNR. Algunos aspectos conflictivos. *Cuadernos de Formación. Colaboración*, 22(10), 2010.

[51] Nigel Gilbert. Agent-based social simulation: dealing with complexity. *The Complex Systems Network of Excellence*, 9(25):1–14, 2004.