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 DEVELOPMENT OF AN AGENT-BASED SOCIAL SIMULATION VISUALIZATION TOOL FOR INDOOR CROWD ANALYTICS BASED ON THE LIBRARY THREE.JS

Pablo Aznar Delgado

2017

TRABAJO FIN DE GRADO

Título:	Diseño y desarrollo de una herramienta de visualización de simulación social basada en agentes para el análisis de multitudes en interiores de edificios basado en la librería Three.js
Título (inglés):	Design and Development of an Agent-based Social Simulation Visualization Tool for Indoor Crowd Analytics based on the library Three.js
Autor:	Pablo Aznar Delgado
Tutor:	Carlos A. Iglesias Fernández
Departamento:	Ingeniería de Sistemas Telemáticos

MIEMBROS DEL TRIBUNAL CALIFICADOR

Presidente:

Vocal:

Secretario:

Suplente:

FECHA DE LECTURA:

CALIFICACIÓN:

UNIVERSIDAD POLITÉCNICA DE MADRID

ESCUELA TÉCNICA SUPERIOR DE
INGENIEROS DE TELECOMUNICACIÓN

Departamento de Ingeniería de Sistemas Telemáticos
Grupo de Sistemas Inteligentes

TRABAJO FIN DE GRADO

DESIGN AND DEVELOPMENT OF AN AGENT-BASED SOCIAL SIMULATION VISUALIZATION TOOL FOR INDOOR CROWD ANALYTICS BASED ON THE LIBRARY THREE.JS

Pablo Aznar Delgado

Junio de 2017

Resumen

La simulación social consiste en modelar el comportamiento de las personas con métodos computacionales. Estos modelos permiten a los investigadores explicar fenómenos sociales y analizar cómo las personas interactúan entre ellas en determinadas situaciones.

Además, la simulación social proporciona información que puede tener diferentes interpretaciones y utilidades. Por ejemplo, permite ver cómo las personas se comportan y se mueven en la evacuación de un edificio o cómo varía su consumo energético. Todos los datos obtenidos de la simulación pueden ser interpretados, pudiendo saber cuál es la forma óptima de diseñar un edificio en función de las necesidades de evacuación o hacer una estimación de la energía consumida, así como de la variación de temperatura.

Existen diferentes herramientas para analizar, de forma numérica, el comportamiento de las personas en interiores de edificios. Sin embargo, existe la creciente necesidad de visualizar estos comportamientos en un entorno 3D. Esto proporcionará la oportunidad de analizar los resultados de la simulación de una forma más sencilla e intuitiva, ya que será más simple y rápido encontrar soluciones a los posibles problemas. Así, poder visualizar en un entorno 3D el comportamiento humano permitirá obtener una información valiosa.

Por consiguiente, el objetivo de este proyecto es diseñar y desarrollar una herramienta que permita visualizar un entorno 3D en el que haya personas interactuando entre ellas. Esto proporcionará una forma muy útil de analizar diferentes situaciones. Para ello se utilizará Three.js, la cual es una librería para visualizar y animar entornos 3D.

Esta herramienta creará un modelo 3D del interior de un edificio con diferentes objetos, donde será posible observar personas haciendo diferentes actividades. Todo esto será muy útil porque dará la oportunidad de ver como diferentes aspectos varían dependiendo del comportamiento humano, como la variación de la temperatura de una habitación cuando no hay nadie en ella o cuando está llena de gente. Además, otro aspecto que podría analizarse sería el consumo de energía, ya que podría representarse cómo varía el consumo en función del estado de los dispositivos, encendido o apagado, cuando no están siendo utilizados.

Palabras clave: Simulación Social, Agente, Three.js, JavaScript, Visualización, Interiores, 3D.

Abstract

Social simulation consists of modeling social behavior with computational methods. These models allow researchers to explain social phenomenons and to analyze how people interact with each other in specific situations.

In addition, social simulation provides information that can be interpreted in a lot of ways and have different utilities. For example, it allows to see how people behave and move in the evacuation of a building or to see how its energy consumption varies. All the data obtained from the simulation can be interpreted in order to know which is the best way to design a building according to evacuation needs or to make an estimation of the energy consumption as well as the temperature variation.

There are different tools that analyze people behaviors inside buildings in a numerical way. However, there is an increasingly need of visualizing these behaviors in a visual form, that means, in a 3D environment. This will give the opportunity to analyze the results of the simulation in an easier and more intuitive way because it will be simpler and faster to find solutions for possible problems. Thus, being able to visualize in a 3D environment the human behavior will allow, in a very easy and simple way, to obtain very valuable data.

Consequently, the objective of this project is to design and to develop a tool that allows to visualize a 3D environment in which there are people interacting with each other. This will provide a very useful and helpful way to analyze different situations. For this purpose, Three.js will be used, which is a JavaScript library to visualize and to animate 3D environments.

This tool will create a 3D model of a specific building interior with its different objects and in which it will be possible to observe people doing several activities. All this could be very useful because it will give the opportunity to see how different aspects vary depending on human behaviors, as the temperature variation of a room when nobody is in it or when it is plenty of people. Furthermore, other aspect that could be analyzed is the energy consumption, as it could be represented how energy consumption varies according to the state, on or off, of the devices when people are not using them.

Keywords: Social Simulation, Agent, Three.js, JavaScript, Visualization, Indoor, 3D.

Agradecimientos

Primero, agradecer a mis padres y a mi hermano por apoyarme durante todos estos años y por sus ánimos y confianza para que acabase esta carrera.

También agradecer a Mar todo su apoyo, sin el cual no habría sido capaz de llegar hasta aquí.

Gracias a mis amigos por todos los momentos compartidos a lo largo de esta carrera y por hacer que todo sea más fácil.

Asímismo, me gustaría agradecer a mis compañeros del GSI por amenizarme las horas de trabajo.

Por último, gracias a mi tutor Carlos Ángel Iglesias por darme la oportunidad de formar parte del Grupo de Sistemas Inteligentes y por su ayuda y apoyo recibido durante la realización de este proyecto.

Contents

Resumen **VII**

Abstract **IX**

Agradecimientos **XI**

Contents **XIII**

List of Figures **XVII**

1 Introduction **1**

1.1 Context 1

1.2 Project goals 2

1.3 Structure of this document 3

2 Enabling Technologies **5**

2.1 Introduction 5

2.2 SOBA 5

2.3 CESBA 6

2.4 Blueprint3D 6

2.5 Three.js 7

2.6 TypeScript 8

2.7 Grunt 9

2.8 Blender 9

3 Architecture **11**

3.1 Introduction . . . 11

3.2 Overview . . . 11

3.3 Batch Module . . . 12

3.4 Model generator . . . 17

3.4.1 Scene . . . 17

3.4.1.1 Audio . . . 17

3.4.1.2 Light . . . 17

3.4.1.3 Video . . . 18

3.4.1.4 Background color . . . 19

3.4.1.5 Fire . . . 19

3.4.2 Items . . . 20

3.4.3 Floor plan . . . 20

3.4.3.1 Performance . . . 21

3.4.4 Camera . . . 23

3.4.5 Agent . . . 23

3.4.5.1 Sentiment . . . 25

3.5 Visualization Module . . . 26

3.5.1 Camera Controls . . . 26

3.5.2 Simulation Controls . . . 26

3.6 Conclusions . . . 27

4 Case study **29**

4.1 Introduction . . . 29

4.2 SOBA . . . 29

4.3 Fire evacuation . . . 32

4.4 Smart office . . . 34

5 Conclusions and future work **39**

5.1 Introduction 39

5.2 Conclusions 39

5.3 Achieved goals 41

5.4 Problems faced 42

5.5 Future work 42

Bibliography **44**

List of Figures

2.1 Blueprint3D 7

3.1 Architecture 12
3.2 Lights 18
3.3 Video 18
3.4 Fire 19
3.5 Initial Performance 21
3.6 Initial Performance 22
3.7 Final Performance 22
3.8 Agent 23
3.9 Sentiments 25
3.10 Dead 26
3.11 Visualization 27

4.1 Lights 30
4.2 SOBA representation 31
4.3 Fire 33
4.4 CESBA representation 34
4.5 GSI Laboratory 35
4.6 Camera 36
4.7 Beacons 36
4.8 TV 36

4.9 GSI Laboratory 37

CHAPTER 1

Introduction

In this chapter, the context where this project takes place as well as its goals and the structure of this document are going to be presented.

1.1 Context

Social simulation consists of modeling social behavior with computational methods. It is focused in the processes, mechanisms and behaviors that build the reality. There are different types of social simulation, being agent-based simulation one of them.

Agent-based social simulation consists of modeling how the agents interact with each other. These models allow researchers to explain social phenomenons and to analyze how people interact with each other in specific situations [6].

In the GSI DIT research group, a software called UbikSim [4] for visualizing social simulations in a 3D environment has been developed. The problem of this software is that it is obsolete because it is based on SweetHome3D [19], which in turn is based on Java3D [18].

Sweet Home 3D is a free interior design application that helps people draw the plan of

their houses, arrange furniture on it and visit the results in 3D [19]. It is based on Java3D, which is an application programming interface used for writing three-dimensional graphics applications and applets [18].

Java3D is outdated and abandoned. Furthermore, UbikSim has only a desktop application, so it is no available online. Thus, there is the need to search for a solution of these problems.

This project is born from the need to solve these problems. It will allow to visualize the results of a social simulation in a 3D environment. Furthermore, this will give the opportunity to analyze the results of the simulation in an easier and more intuitive way because it will be simpler and faster to find solutions for possible problems. For this purpose, Three.js will be used, which is a JavaScript library to visualize and to animate 3D environments.

In addition, a software called SOBA [13] has been developed in the GSI, which is a Simulator of Occupancy based on Agents. Nevertheless, it only enables to visualize the results of the simulation in 2D, which means that it will be very useful to visualize in a 3D environment the results of it.

1.2 Project goals

The main goal of this project is the development of a tool that allows users to visualize a 3D environment in which there are people interacting with each other. Therefore, it will allow to other simulation tools to visualize the result of their simulation.

This main goal can be divided in different tasks:

- Generating the building in which the simulation takes place.
- Representing the agents.
- Visualizing the movement of the agents.
- Combining the social simulator with the representation tool.
- Representing agent's sentiments.
- Representing lights on and off.
- Setting or modifying simulation's parameters.

1.3 Structure of this document

In this section, a brief overview of the chapters included in this document is provided. The structure is the following one:

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

Chapter 2 provides a description of the main technologies that are used in this project and justifies the use of them.

Chapter 3 explains the architecture of this project. A global vision about the architecture and all its components in detail will be presented.

Chapter 4 presents different study cases that can be applied to this project.

Chapter 5 describes the conclusions, the achieved goals, the problems encountered during the development of this project and the future work.

CHAPTER 2

Enabling Technologies

2.1 Introduction

In this chapter, the technologies used in this project are going to be introduced. Firstly, it will be explained SOBA [13] and CESBA [9], the tools that will provide the parameters of the simulation. Secondly, it will be introduced Blueprint3D [1], which is the technology used to represent a 3D environment. Finally, the tools used by Blueprint3D will be described, such as TypeScript [2], Grunt [5], Three.js [8] which is a JavaScript library for represent and animate 3D environments and Blender which is a 3D creation suite.

2.2 SOBA

SOBA is a Simulator of Occupancy Based on Agents useful for developing studies and researches based on social simulation, mainly in buildings. In order to configure the simulation, the behaviors of all types of occupants, a physical space and agents that interconnect between each other and with the occupants have to be declared. The simulation and results can be evaluated both in real time and post-simulation [13].

SOBA, which is based on Mesa [12], will be used as the source of the parameters that

are going to be represented. It will run the simulation and generate a log with the agents behavior. This log will be interpreted and will define all the parameters of the representation.

2.3 CESBA

CESBA is a Crowd Evacuation Simulator Based on Agents [9] useful for developing studies and researches of building evacuations and it is based on Mesa [12].

This software allows users to know, through agent-based social simulation, how people react in case of fire and how they search the nearest emergency exit. It models how people act in their work place and how a fire is originated. Furthermore, it studies affiliation models in order to understand how people behave when a relative is also in the evacuation.

CESBA, as SOBA, will be used as the source of the parameters that are going to be represented. It will allow to visualize a simulation of a fire evacuation in a building. Being able to visualize the simulation in a 3D environment will allow to obtain more valuable data and analyze the situation in an easier way.

2.4 Blueprint3D

Blueprint3D is a customizable application that lets users to design an interior space such as a home or an apartment [1]. Moreover, it enables to modify the distribution of the rooms and the location of the items.

In this project, it is used to represent the 3D environment where the simulation takes place and it is modified to represent as well the agents and their behavior.

This tool is written in TypeScript, built on Three.js and it runs in a web browser making use of a local server such as Python's built in webserver.

It contains an example directory in which there is an application built using the core Blueprint3D JavaScript building blocks. Furthermore, it adds html, css, models, textures and more JavaScript to tie everything together.

The core of Blueprint3D is made up of the following components:

- ***Core:*** basic utilities such as generic functions.
- ***Floorplanner:*** 2D tool for editing the floor plan.

- ***Items:*** various types of items that can be located in different rooms.
- ***Model:*** data model that links the 2D floor plans with the items in it.
- ***Three:*** mainly the 3D view controller. It manages the camera and item placements controls.

Figure 2.1: Blueprint3D

2.5 Three.js

Three.js is a JavaScript library for creating and displaying animated computer 3D graphics in a web browser. It was first released to GitHub in 2010 by Ricardo Cabello (mrdoob) [14] and nowadays it has around 800 contributors.

To display the animated computer 3D graphics is needed a browser where you can open an html file in which the Three.js script it is executed. In that script, it has to be declared a scene, a camera and a renderer so we can render the scene with the camera [7].

- ***Scene:*** all the things that are going to be represented are added here.
- ***Camera:*** there are a few different cameras in Three.js and they display the part of the scene that is wanted to be represented.
- ***Renderer:*** to actually render the scene is needed what is called a render loop. It will draw the scene 60 times per second, using instead of JavaScript's function *setInter-*

val(). The advantage of this is that if the users navigate to another browser tab, it pauses not wasting computing resources.

The following code is an example that shows how to add a cube to the scene [14].

Listing 2.1: Three.js example code

```
var geometry = new THREE.BoxGeometry( 1, 1, 1 );
var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );
var cube = new THREE.Mesh( geometry, material );
scene.add( cube );
```

As it can be seen, first of all the geometry which contains the vertices and faces has to be defined. There is a considerable number of different geometries available in Three.js library. Secondly, the material which color the geometry has to be determined. It is an object of properties that is applied to the geometry, but in this example is only defined the color attribute. Thirdly, the cube is created and it is the type of mesh. A mesh applies a material to a geometry. Finally, the cube is added to the scene and it will be represented.

In addition, Three.js also allows the animation of 3D models. The animation could be the cube rotating as well as a man walking.

2.6 TypeScript

TypeScript is a language for application-scale JavaScript. Moreover, it adds optional types, classes, and modules to JavaScript. In addition, it supports tools for large-scale JavaScript applications for any browser, for any host, on any OS. TypeScript compiles to readable, standards-based JavaScript [2].

TypeScript starts from the same syntax and semantics that millions of JavaScript developers know today. It uses existing JavaScript code, incorporate popular JavaScript libraries, and call TypeScript code from JavaScript.

There are two ways of using TypeScript tools:

- Installing in Visual Studio TypeScript's plugins.
- Installing through npm.

2.7 Grunt

Grunt is a JavaScript task runner to automate repetitive tasks with a minimum of effort. This tasks could be minification, compilation, unit testing, linting, etc. All this tasks can be configured through a Gruntfile and executed using a command-line interface. It is written in Node.js and distributed via npm [5].

To prepare a new Grunt project is needed to add two files to your project:

- ***Package.json:*** This file stores metadata of the project and is used by npm. Grunt and Grunt plugins has to be listed as *devDependecies*.
- ***Gruntfile:*** This file configures or defines tasks and load Grunt plugins.

2.8 Blender

Blender is the free and open source 3D creation suite. It supports the entirety of the 3D pipeline—modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation [10]. Its initial release was in January 1995.

It is cross-platform and runs in Linux, Windows and Macintosh. In addition, there is a Three.js exporter add-on which allows the exportation of 3D models ready to add to Three.js.

In this project, Blender is used to create and modify some 3D models and then export them to Three.js.

CHAPTER 3

Architecture

3.1 Introduction

In this chapter, the architecture of this project is going to be explained. Firstly, it will be presented a global vision about the project architecture, identifying all the modules that compose it and illustrating an overview of how they are linked. Finally, it will be described all the modules in detail, explaining all their components and how they interact between each other.

3.2 Overview

In this section, the global architecture is going to be presented, describing the main modules that compose it. Fig. 3.1 shows an UML diagram that represents the architecture of this project.

The following modules can be identified:

- ***Batch module:*** this module is in charge of collecting the data proportionated by the user, such as the floor plan, the rooms assignation and the movement of the agents.

- ***Model generator:*** thanks to the data collected by the batch module, it is able to generate the scene in which there are the different elements that are going to be represented. The main elements are the floor plan, the agents, the items and the camera. All of them make use of the renderer which proportionates the data to represent the visualization.

- ***Visualization module:*** it manages the final result of the visualization on the browser.

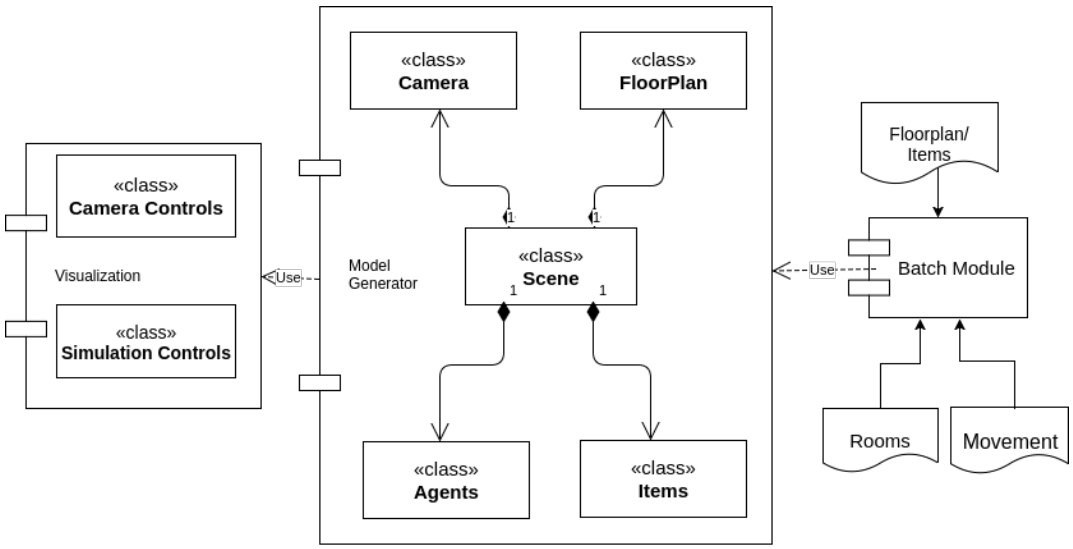

Figure 3.1: Architecture

In the following sections, the different modules that compose this project are going to be explained.

3.3 Batch Module

The purpose of this module is to transform a log generated in the simulated system and to convert it into the objects of the visualization module.

The files that the user has to proportionate are the following ones:

- ***Floorplan/Items:*** JSON file in which is defined all the floor plan and the items in it. In addition, the floor plan is compounded by all the corners of the map and the walls between them. It has to be defined as the following example code, which represents a wall defined by its corners and the wall itself.

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

In the following example, is declared an open door in the middle of the wall.

Listing 3.1: Example of floor plan

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

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

- ***Rooms:*** JSON file in which are defined the names of the rooms. It has to be declared the name and the coordinates of the center of the room. It is used to identify all the rooms by the name that the user has chosen.

Listing 3.2: Example of room

```
{"room":
  [{
    "name": "Hall.1",
    "x": 928.116,
    "y": 398.045
  }]
}
```

- ***Movement:*** JSON file in which are declared all the actions that occur during the simulation. It is divided in steps and in each one, all the actions that happen in that moment are defined. The steps are a way of measuring time and everyone has the same duration, which is defined by the simulator.

  The following code is an example that explains how to declare this file. Everything is inside an array called *steps* and every position of that array is an step.

  The different actions that can be executed are the following ones:

  - *Add new agent:* it is needed an agent id, its position and its sentiment.
  - *Turn on/off lights:* it is needed a parameter that indicates if the light has to be turn on or off and the room in which this action is wanted to be executed.
  - *Turn on/off TV:* it is needed a parameter that indicates if the TV has to be switched on or off and the room in which this action is wanted to be executed.
  - *Move an agent:* it is needed the id of the agent that is wanted to be moved, the room in which it is going to be moved and the step in which the agent is going

to arrive to that position. The speed of the movement is calculated from the duration of the movement and the distance.

- *Add fire:* the possibility of adding fire is implemented in order to represent fire evacuations. It is needed a parameter that indicates if there is fire and its position.
- *Change agent's sentiment:* it is needed the agent id and the sentiment. This provides the possibility of changing the sentiment of the agent in any step. It can also be changed when the movement is declared, so the agent's sentiment will change before the movement starts. To do so, it is necessary to add the attribute *sentiment* in the declaration.
- *Remove an agent:* in order to represent an agent leaving the building, the *outBuilding* attribute is used. It is needed the agent id and the parameter that indicates that it is out of the building.

Listing 3.3: Example of movement, type 0

```
{ "type" : 0,
  "steps": [
  [
    {
      "agent": 0,
      "position": "C.10",
      "sentiment": "anger"
    },
    {
      "agent": 1,
      "position": "C.1",
      "sentiment": "happiness"
  }],
  [
    {
      "light": false,
      "room": "Lab1"
    },
    {
      "video": true,
      "room": "Office3"
    },
    {
      "fire": true,
      "room": "Office1"
```

```
        },
      ],
      [
        {
          "agent": 0,
          "moveTo": "C.2",
          "toStep": 15
        },
        {
          "agent": 1,
          "sentiment": "sadness",
        }
      ]
    }
```

In this example, two different agents are situated in the first step. Then, in the next step, it is turned off the light of the *Lab1*, the TV of the *Office3* is switched on and in the *Office1* is lighten up a fire. In the following step, it is declared the movement of the agent *0* to *C.2* and the time of the arrival as well as the sentiment of the agent *1*. Therefore, the simulation will end in the step 15.

Furthermore, the position of adding a new agent or moving it, can also be proportionated by its coordinates. Therefore, the user can define an exact position of the agent in a room instead of the center of the room. Moreover, the user has to declare which of the two methods is going to use and for this purpose the type attribute is used. Thus, type 0 uses rooms and type 1 uses coordinates.

In the code above, the position is defined by the room whereas in the code below it is defined by coordinates.

Listing 3.4: Example of movement, type 1

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

```
            "agent": 0,
            "moveTo": {"x": 2000, "y": 400},
            "toStep": 10
          }
        ]
        ]
      }
```

3.4 Model generator

This module is in charge of generating the scene with the data collected by the Batch Module. Blueprint3D proportionates the creation of the 3D environment but it had to be modified in order to improve its functionality. Furthermore, new features has been added and they are explained below.

3.4.1 Scene

In the scene, all the elements that are going to be represented are added. These elements are the floor plan and the items which are explained below.

The main elements of the scene that have been added or modified are the following ones:

3.4.1.1 Audio

Two different ways of adding audio have been implemented:

- **Background music:** it allows users to listen to music during the simulation.
- **Directional music:** it enables to simulate a sound system. If the camera is approached to the item, the music will sound louder.

3.4.1.2 Light

Blueprint3D does not proportionate a way to turn off lights. In this project, in order to represent a room with the lights off, the textures of the walls are changed. More specifically, the brightness and contrast of the textures have been modified. Fig. 3.2 shows an example of two rooms, one with the light on and the other with the light off.

Figure 3.2: Lights

3.4.1.3 Video

In order to make the simulation more accurate to real life, the possibility to switch on a TV has been implemented.

Three.js enables the implementation of video through a texture that changes every render. Furthermore, a function that search a TV in a room and place the texture on the TV has been created.

Figure 3.3: Video

3.4.1.4 Background color

The background color of the scene in Blueprint3D is white and it is difficult to differentiate that color from the walls. Furthermore, Blueprint3D use what is called a skybox which is a giant sphere in which is inside the scene, achieving that the background color was the same as the color of the sphere. When loading a big floor plan, the skybox causes some problems because it is not big enough. As a consequence, to solve those problems, the skybox has been deleted and the background color was changed to grey.

Listing 3.5: Background color

```
renderer.setClearColor( 0x808080, 1);
```

3.4.1.5 Fire

In order to represent a fire evacuation, the possibility of adding fire has been implemented.

As it is shown in Fig. 3.4, the fire is represented as a sphere with its texture illustrated through a PNG image similar to the flames. The sphere has been animated in order to make it more real, so it is deformated during the simulation. Furthermore, some shaders have been applied, avoiding it to appear static.

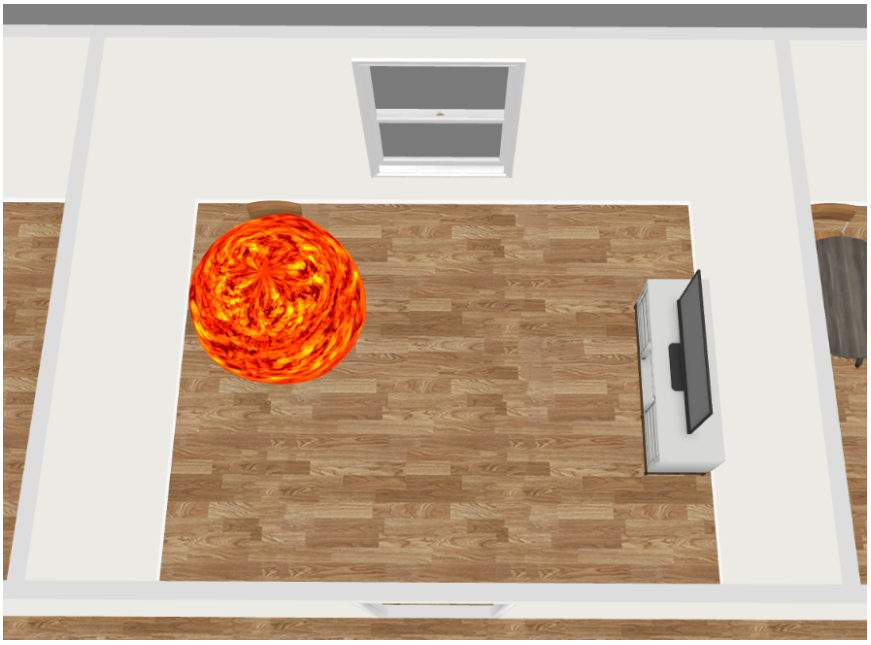

Figure 3.4: Fire

3.4.2 Items

Blueprint3D has a lot of items to decorate the building. However, some of them had the textures missing, so they have been deleted because they were not useful.

In order to make the process of adding objects to the scene easier, Blueprint3D defines different types of objects:

- **Type 1, *FloorItem*:** it covers all the objects that are on the floor such as chairs, tables, bookshelfs...
- **Type 2, *WallItem*:** it covers all the objects that are hanged in the wall such as posters.
- **Type 3, *InWallItem*:** this type of item refers to windows.
- **Type 7, *InWallFloorItem*:** this type of item refers to doors.
- **Type 8, *WallItem*:** this type of item refers to carpets.

On the other hand, some objects has been added in order to satisfy the needs of this project, for example the blackboard, because it was needed to represent a class in a faithful way.

The items are JSON files that can be generated using Blender ThreeJS exporter. These files are loaded using ThreeJS JSON loader, but Blueprint3D has a problem loading them. This problem is that if there are, for example, two doors, instead of loading one time the JSON 3D model, it loads them one time for each model. As a consequence, this slowed down the whole process and it was non-viable. So, to speed up the process, it has been modified and now the items are cached. This means that, despite how many same items are in the scene, the 3D model is only loaded one time.

3.4.3 Floor plan

Blueprint3D creates the floor plan from the JSON file explained before in the Batch Module. It takes all the corners and walls and creates the rooms that form the floor plan.

However, when loading the textures, Blueprint3D has the same problem as loading objects. Despite of all the textures are the same, it loads the same texture one time for each wall, so this slowed down the process of loading a floor plan. As well as the items, textures have been cached and now they are only loaded once and then placed in every wall.

3.4.3.1 Performance

At the beginning of the development of this project, everything was tested in a floor plan with fifteen rooms and the problems mentioned before when loading items and textures were found. Therefore, Blueprint3D took about thirty seconds to load all the floor plan. Once cached items and textures, this problem was solved because the loading time was reduced to less than a second.

When the development of this project was advanced, the floor plan was changed and everything was tested in the floor plan of the B Building of the ETSIT. However, it took about twenty seconds to load it, in spite of items and textures were cached.

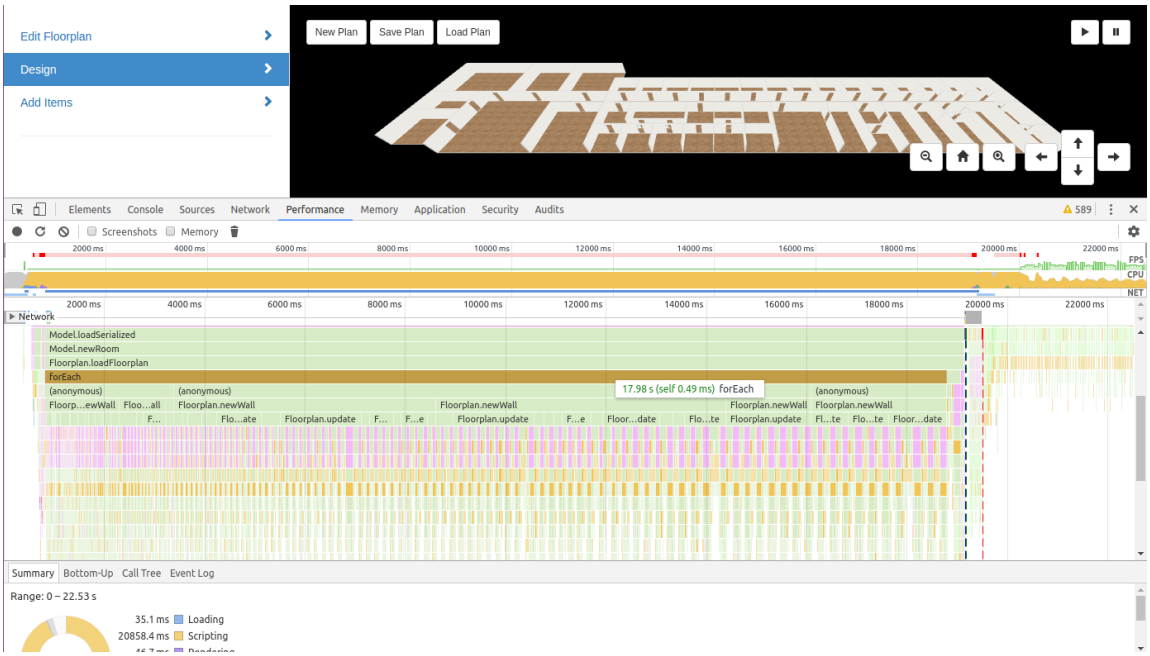

Figure 3.5: Initial Performance

As it can be seen in Fig. 3.5, there was a function called *forEach* that elapsed 17,98 seconds.

In Fig. 3.6, it can be seen that inside the *forEach* function there are a lot of *newWall* functions and each one execute an update and some callbacks. So the problem was that the same function was executed a lot of times.

Therefore, in order to solve the problem, the way of creating the walls has been modified. Now, every wall is defined and after all, the update is executed. Only one update is executed, instead of executing one update for each wall. In Fig. 3.7, it can be seen that now the loading time has been reduced to less than 2 seconds.

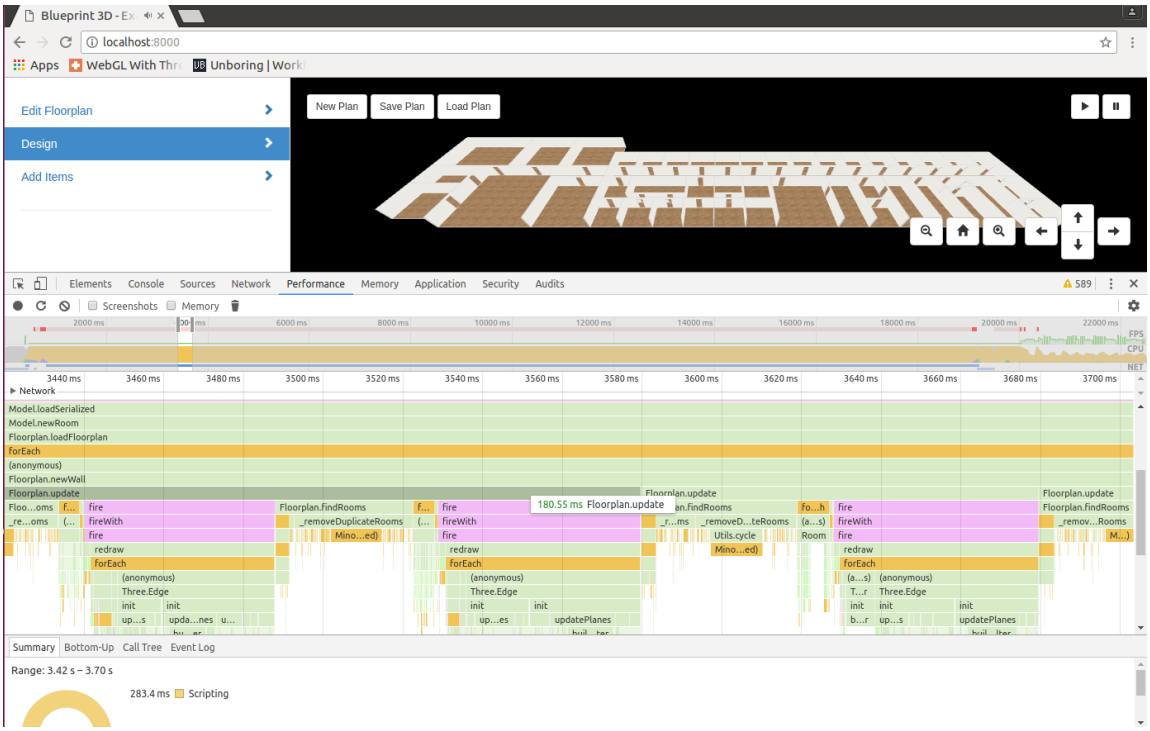

Figure 3.6: Initial Performance

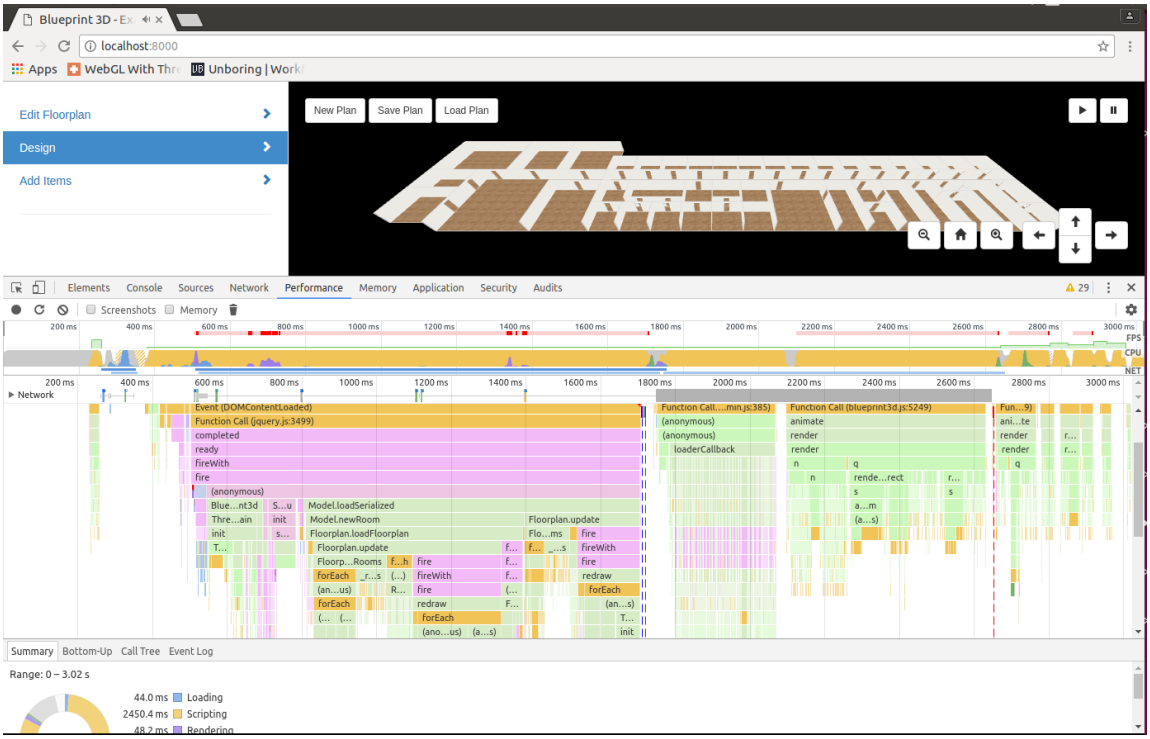

Figure 3.7: Final Performance

3.4.4 Camera

The camera was setted up by Blueprint3D and it uses the *Perspective Camera* of Three.js. Nevertheless, Blueprint was not designed to visualize big buildings, so the zoom limits and the render distance were too small. In order to visualize a big floor plan in this project, these parameters has been changed.

Listing 3.6: Camera

```
camera = new THREE.PerspectiveCamera(45, 1, 1, 14500);
```

In the code above, it can be seen the declaration of the camera. The first attribute is the field of view and the second one is the aspect ratio. The third and fourth parameters are the render distance, anything nearer or further than this attributes will not be displayed [7].

3.4.5 Agent

The agents are the representation of humans beings in the simulation. The 3D model of the agents can be seen in Fig. 3.8.

Figure 3.8: Agent

This model is imported to Blender and exported to a JSON file using the ThreeJS exporter. In order to load it into this project, it is used the JSONLoader from ThreeJS.

Listing 3.7: JSON Loader

```
var jsonLoader = new THREE.JSONLoader();
jsonLoader.load( "/models/js/walkmorphcolor.json", addModelToScene);
```

The load function has a callback called *addModelToScene* function, which is in charge of adding the module in the right place to the scene.

Furthermore, the 3D model is cached in order to not impair performance, as well as the items and the textures. Thus, the 3D model is only loaded one time regardless of how many agents are in the scene.

The movement is proportionated by the user, as it has been explained before in the Batch Module. The walking action is composed of two movements:

- Translation: it is the movement that enables the agent to advance.
- Animation: legs and arms can be animated.

Listing 3.8: Move function

```
this.move = function (mesh, i, speed) {
  var time = Date.now();
  // Translation Movement
  var moveDistance = speed;
  scene.meshes[i].translateZ(moveDistance);
  // Animation
  for (var z=0; z<mixers.length; z++){
    mixers[z].update((time - prevTime)*0.0005);
  }
  prevTime = time;
}
```

In the code above, the function *move* is presented. It enables to translate the agent as well as play its animation.

In addition, the agent has to rotate an specific angle to reach the final position of its movement. Therefore, that angle is calculated and the agent rotates during its movement.

Besides, the agent stops when it hits a wall in order to avoid some possible errors of the simulator that proportionates the parameters of the movement.

The speed is calculated from the duration of the movement and the distance. However, the distance that is going to be covered has to be declared in every render. So, in order to set the speed of the agent, the following formula has been used:

$$speed = distance/(time * fps) \tag{3.1}$$

The frames per second are calculated according to the time of each render. In this way, the distance covered in every render can be calculated.

3.4.5.1 Sentiment

In order to be able to identify the sentiment of the agent, it has been chosen to change the agent's color. The sentiments chosen to visualize are the big six emotions defined by Paul Ekman: happiness, sadness, fear, surprise, anger and disgust [17]. The colors that represent each sentiment are: [3]

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

Figure 3.9: Sentiments

In addition, representing dead people could be very useful in a fire evacuation. For this purpose, black color has been selected as it is shown in the Fig. 3.10.

Figure 3.10: Dead

3.5 Visualization Module

The visualization module is in charge of representing the final result in the browser. It allows users to control the simulation as well as the camera.

A local server is needed to run the software, such as the *python SimpleHTTPServer*, which enables to visualize in a browser the html in which all the Three.js script is executed. Once running the local server, accessing to *localhost:8000* will allow to visualize the simulation.

3.5.1 Camera Controls

Blueprint3D allows users to move the camera. However, some parameters of the controls have been changed because, for example, the zoom-out was not enough to see a big floor plan.

The camera can be moved using the mouse and with buttons that are located down to the right, as it is shown in the Fig 3.11. The permitted movements are zoom-in, zoom-out, rotation and translation.

3.5.2 Simulation Controls

The visualization module enables to pause and play the simulation. This is very useful because being able to pause the simulation in any moment allows users to see everything carefully in a way that, if it was not stopped, would not be possible. Therefore, the possibility of analyzing the simulation in a better way is given.

Figure 3.11: Visualization

3.6 Conclusions

In this chapter, the three modules that compose this project are explained, as well as how they are linked.

In conclusion, this project counts with a Batch Module that collects all the data proportionated by the user in order to allow the Model generator to interpretate it. Therefore, the Model generator creates the scene in which there are the different elements that are going to be represented. Ultimately, the Visualization Module manages the final result of the visualization on the browser, enabling to control the camera and some parameters of the simulation.

CHAPTER 4

Case study

4.1 Introduction

In this chapter, three examples that could be represented with this software are presented. In the GSI DIT research group, different softwares that represent people's behavior in different situations has been developed. So, three study cases will be detailed. Firstly, the representation of SOBA will be explained. Secondly, the case study of an evacuation caused by fire will be described. Finally, the case study of a Smart Office will be presented.

4.2 SOBA

Social simulation gives the opportunity of studying people's behavior. Modeling how people act when they are working could be of great utility in order to know the time at which they arrive to their work place, whether they turn off lights and computers when they are not using them... Interpreting all this data allows users to study the energy consumption and how temperature varies in an office.

Furthermore, being able to make an estimation of energy consumption and temperature variation gives the opportunity of improving the infrastructures in order to save energy. It

enables to know where the air conditioner is necessary and which lights or computers can be switched off. Therefore, the impact on the environment will be reduced as well as the energy costs.

Consequentially, in the GSI DIT research group, a software called SOBA has been developed. As it has been explained in the Chapter 2 Enabling Technologies, SOBA is a Simulator of Occupancy Based on Agents. The simulations are configured by declaring a type of occupant, with specific and definable behavior, a physical space and agents that interact between each other [13].

This software models how many time people are working and, on that basis, it makes an estimation of how much energy has been consumed and how temperature has varied. Visualizing the results of the simulation will allow to reach better conclusions in an easier way. Therefore, the software developed in this project is a great complement of SOBA.

For this purpose, the exact position of the agents is not important, all that matters is the room where they are. For this reason, it is enough that the movement parameters proportionated, explained in Sect. 3.3 Batch Module, were of type 0. That is because type 1 defines the exact coordinates of the agents not needing that precision. Furthermore, it is not necessary because SOBA does not have that level of abstraction, so it does not have to be such specific.

Figure 4.1: Lights

In the Fig. 4.1, it is shown how lights are represented. As it was explained in Sect. 3.4 Model Generator, it is executed by changing the texture of the walls.

The representation of SOBA consists of visualizing how the agents move around the office. Furthermore, it is represented if they switch off the lights when they take a break or when they finish working.

In order to improve the software, a new feature that could be implemented, as it is explained in Sect. 5.5 Future Work, is adding the possibility of visualizing the energy consumption and the temperature variation. In this way, more results of the simulation could be represented, facilitating the process of making conclusions.

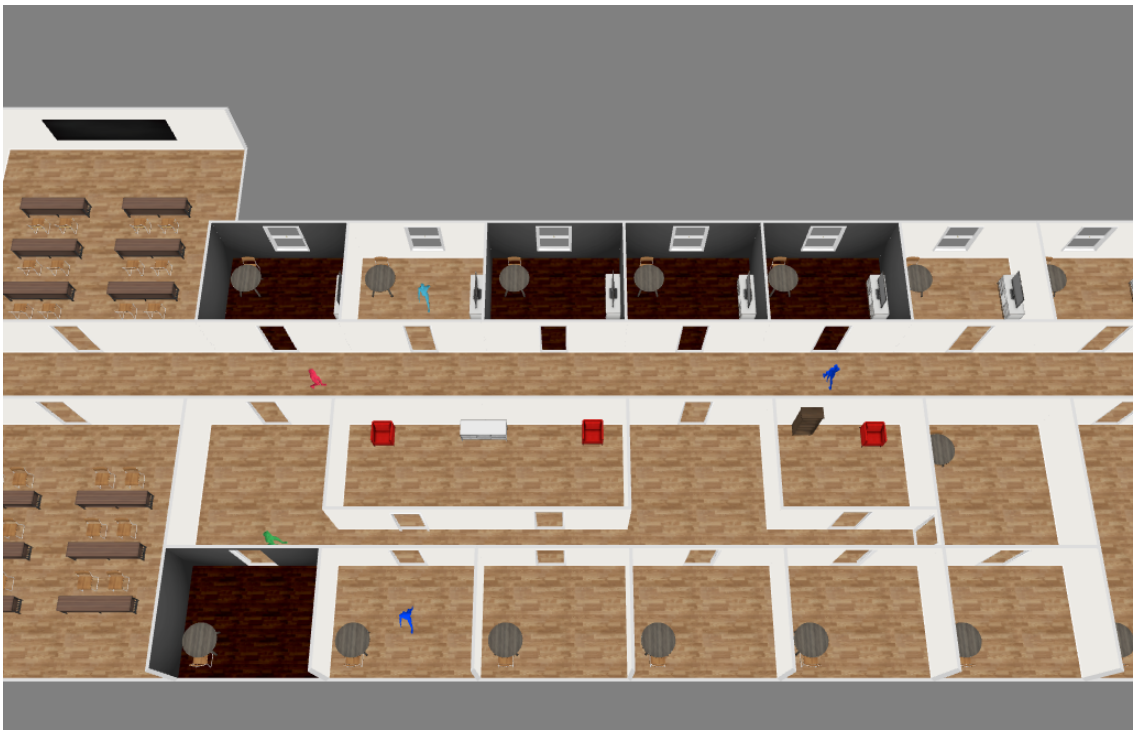

Figure 4.2: SOBA representation

In the Fig. 4.2, it is shown an example of representation of SOBA in which it is represented the second floor of the B building of the ETSIT with its different classes, offices and laboratories. Moreover, it can be seen various agents in their work place and how lights are represented.

4.3 Fire evacuation

In a fire evacuation, it is very useful to know the way in which people react and behave. The best way to find out these things is through a fire drill because it is able to proportionate very valuable and accurate data that could help to improve the infrastructures and the location of emergency exits.

However, making a fire drill is very expensive and complicated to organize. Thus, social simulation is capable of solving these problems because it provides an accurate way to analyze people behavior without arranging a fire drill.

As a consequence, in the GSI DIT research group, a software called CESBA [9] has been developed, which is a Crowd Evacuation Simulation Based on Agents. This software allows users to know, through social simulation, how people react in case of fire and how they search the nearest emergency exit.

CESBA models how people act in their work place and how a fire is originated. Until the fire alarm does not turn on, people will continue working because they are not aware of any problem. When the fire alarm is activated, it is shown how they start running and looking for the nearest exit. Furthermore, the spread of fire is also modeled, providing the possibility of studying where is the most dangerous place in a building where a fire could be originated.

The software developed in this project will represent the data obtained in the CESBA simulation. Being able to visualize the simulation in a 3D environment will allow to obtain more valuable data and analyze the situation in an easier way. In this representation, it will be shown how people run looking for the exit and how the fire spreads. In case that an agent dies, it will be represented as shown in the Fig. 3.10.

In addition, the exact position of the agents in an evacuation is very important. In this way, it could be represented how an agent cannot run faster because someone is blocking its path as well as bottlenecks originated in doors. Therefore, in order to proportionate the movement parameters explained in Sect. 3.3 Batch Module, the type 1 is going to be used. This is because the type 0 only defines the room in which the agent is located and, for this purpose, more precision is needed. Using the type 1 allows users to define the agent's position by its coordinates, so it meets the requirements.

In the Fig. 4.3, it is shown how fire is represented. CESBA will proportionate the positions of the fire, therefore, it will be able to represent its spreading.

Figure 4.3: Fire

CESBA distinguishes different types of agents, such as old people and children. As a consequence, in the simulation, not every agent run at the same speed.

In this project, different speeds could be represented, but all of them through the same 3D agent model. As it has been explained in Sect. 5.5 Future Work, in order to improve the representation, a possible new feature that could be implemented is adding new 3D models of agents, proportionating the possibility to represent women, old people and children.

In conclusion, the software developed in this project could be of great utility to CESBA because it will be a great tool to analyze and obtain all the data of the simulation. Besides, the obtained information could be very similar to the one obtained from fire drills. As a consequence, this software will save costs and difficulties originated by fire drills.

In the Fig. 4.4, it is shown an example of a simulation in which people are running due to the risk situation created by a fire.

Figure 4.4: CESBA representation

4.4 Smart office

In an office, it could be very useful to know workers' mood in order to improve productivity. Employees suffering from high stress levels are less productive and have higher absenteeism levels than those not working under excessive pressure [11].

Social simulation enables to know when people are beginning to feel stressed, therefore, some measures can be applied in order to reduce that stress and to improve productivity.

As a consequence, in the GSI DIT research group, a software that models people emotions while they are working has been developed [15]. It uses facial detection in order to recognize the sentiments of the workers. For this purpose, some cameras are needed and the images captured by them are processed to know the emotions.

Furthermore, beacons are used as presence detectors. They are passive gadgets that only emit data, what means that they do not receive any information. They detect human presence and behavior and trigger pre-programmed actions delivering contextual and personalized experiences [16]. For example, if a worker approaches the TV, it will be automatically switched on.

With the data collected from the cameras and the beacons, the office environment can

be adapted to people needs. For instance, if a worker is stressed, some relaxing music will be played.

The software developed in this project is very useful to represent a Smart Office because it allows users to visualize the agents and their emotions. For this purpose, it has been chosen to represent the GSI Laboratory, as it is shown in the Fig. 4.5.

Figure 4.5: GSI Laboratory

In order to represent the different emotions of the agents, as it has been explained in Sect. 3.4.5 Agent, the color of the agent is changed. The emotions chosen to be represented are the big six emotions defined by Paul Ekman: happiness, sadness, fear, surprise, anger and disgust [17].

As the 3D models of the cameras and the beacons were not available in Blueprint3D, they have been modeled using Blender. Once the model was completed in Blender, it was exported to a JSON file using the Three.js exporter [8].

In the Fig. 4.6, it is shown the 3D model that represents the cameras that captures workers emotions and in the Fig. 4.7, it is shown the representation of the beacons used to detect presence.

Figure 4.6: Camera

Figure 4.7: Beacons

As it has been explained in the Sect 5.5 Future Work, in order to make the representation more accurate to real life, a new feature that could be implemented is the addition of new animations that represent different actions of the agents. Some examples of these new animations are the action of sitting down, pressing the light switch...

Furthermore, as it is shown in the Fig. 4.8, playing a video in the TV is also possible, allowing to understand the simulation in a better way.

Figure 4.8: TV

In the example of the Fig. 4.9, it is shown a representation of people working with different moods, such as stressed workers or relaxed people taking a break while watching TV.

In conclusion, being able to visualize the representation of an Smart Office enables to know how the stress levels of the workers evolve. Therefore, it facilitates the process of deciding what measures can be taken in order to lower those levels. Furthermore, it is possible to visualize how stressed workers react when the office environment is changed in order to create a relaxed atmosphere. All this enables to improve the productivity of the office since the workers will be more satisfied.

Figure 4.9: GSI Laboratory

CHAPTER 5

Conclusions and future work

5.1 Introduction

In this chapter, the conclusions, the achieved goals, the problems encountered during the development of this project and the future work will be described.

5.2 Conclusions

In this project, a 3D environment has been created, where the results of a social simulation can be represented. For this purpose, Three.js has been used, which is a JavaScript library that allows users to visualize and to animate 3D environments.

This project is composed of three modules: Batch Module, Model Generator and Visualization Module. The Batch Module collects the data proportionated by the user in order to allow the Model Generator to interpret it. This data corresponds to the floor plan, the location of the items and the behaviour of the agents. The Model Generator, thanks to the data collected by the Batch Module, generates the scene with all the elements that are going to be represented, such as the floor plan, the agents, the items and the camera. Lastly, the Visualization Module manages the final result of the visualization on the browser, providing

the possibility of controlling the camera and some parameters of the simulation.

The software developed in this project is going to be used in other projects of the GSI research group such as SOBA, CESBA and Smart Office, making possible the visualization of the results of their simulations. Visualizing that 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.

This project is based on Blueprint3D which, as it is explained in Sect. 2.4, is a customizable application that lets users to design an interior space such as a home or an apartment.

Blueprint3D is outdated because there has not been any updates in the last year, which has caused some problems in the development of this project. For example, it does not use the last release of Three.js and that originated some difficulties explained in Sect. 5.4. Furthermore, the performance of Blueprint3D is not optimal which caused the necessity of changing some parts of the project. Moreover, it is a big project so, understanding how all the code works has not been easy because it is not documented. Despite all of this, Blueprint3D has provided a useful basis for this project and has made a lot of things simpler.

Furthermore, Blueprint3D is based on Three.js and the use of it has provided the acquirement of very useful knowledge. Additionally, the development of this project required the use of tools such as Blender, so it was necessary to learn how to use it. As it has been explained in Sect. 2.8, Blender is a 3D creation suite that allows users to create 3D models, animated films, video games...

Therefore, the development of this project has provided useful knowledge, particularly beneficial in video games development, thanks to the use of Three.js, as well as in 3D modeling, due to the use of Blender.

Moreover, this project has provided the opportunity to understand social simulation, specifically agent-based social simulation, which consists of modeling social behaviour with computational methods.

In the following sections, the achieved goals, the problems faced and some possible future work are explained.

5.3 Achieved goals

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

Generating the building in which the simulation takes place. The representation of the 3D environment in which the simulation takes place has been made through Blueprint3D. However, it has been necessary to change some features in order to improve the performance of the simulation. Moreover, a floor plan has been represented, allowing the camera to move in order to visualize what is wanted.

Representing the agents. For this purpose, a 3D model of a human being is added to the scene. The 3D model was loaded in Blender and exported into a JSON file using the Three.js Blender exporter. When loading this JSON file in the project, the agent is represented.

Visualizing the movement of the agents. The walking action is composed of two movements: animation and translation. The 3D model of the agent, explained in the previous point, incorporates the animation of walking. However, it has been necessary to implement the logic of the translation movement.

Combining the social simulator with the representation tool. This is the main goal of the project because the social simulator is the one which provides all the parameters that will be represented. These parameters are proportionated to the Batch Module and then interpreted in order to visualize them.

Representing agent's sentiments. For this goal, the agent's color is modified, having assigned different colors for each emotion. The emotions represented are happiness, sadness, fear, disgust, anger and surprise.

Representing lights on and off. In order to achieve this goal, the wall textures has been changed. Moreover, to represent a light off, the brightness and contrast of the texture of the wall has been modified to make it darker.

Setting or modifying simulation's parameters. The simulation can be paused in every moment, which allows users to analyze in a better way what is happening in a certain moment of the simulation.

5.4 Problems faced

During the development of this project, some problems has had to be faced. These problems are the following ones:

- Items and textures were not cached in Blueprint3D: every item and texture was loaded once for each one of them, instead of caching the ones that were repeated. This impairs the loading time, making it non-viable.
- Loading a big map lasted more than 20 seconds: once the items and textures were cached, loading a map with 15 rooms was not a problem. However, the load of a big map such as the B building of the ETSIT lasted more than 20 seconds. This was provoked by the way Blueprint generates a map, since it defines a wall and executes an update of all the floor plan. This means that the same task was executed for each wall instead of defining all the walls and then update the floor plan. As a consequence, this was changed, achieving that the load of a big map stopped impairing the performance.
- The Three.js release used by Blueprint3D was outdated: Blueprint3D uses the release 69 of Three.js. However, this provoked the impossibility of visualizing an animation. The animations in this project are very important because one of the main goals of it is to visualize an agent walking. In order to solve this problem, the release used of Three.js has been changed to the release 79. Some Blueprint3D features stopped working but they were not important for the development of this project.

5.5 Future work

In this section, the implementation of possible new features or improvements of the project will be presented.

Adding new animations At this moment, the only animation that can be visualized is the action of walking. For this reason, an improvement of the project could be the addition of more animations in order to visualize different actions of the agents, making the simulation more accurate to real life. Some examples of the new animations are the actions of sitting down, pressing the light switch, pressing the TV switch...

Adding new agents Currently, all the agents are represented through the same 3D model. Therefore, the incorporation of new 3D models such as women, children and old people may represent a good progress in the project.

Visualizing energy consumption At this stage, it is possible to visualize the state, on and off, of lights and TVs. Thus, it could be interesting to implement a way of representing the energy consumption of theses objects. Furthermore, this will allow SOBA to visualize more parameters of its simulation.

Visualizing temperature variation This improvement is related with the previous one. The main idea of this point is to provide the possibility of visualizing the temperature variation in order to understand the energy consumption in a better way.

Fixing the floor plan creation tool Blueprint3D has a tool that enables to draw the floor plan in a 3D model from a 2D model. However, this feature stopped working when the Three.js release was changed. So, it could be very useful to fix this tool in order to make easier the floor plan creation for future developments.

Bibliography

[1] Blueprint3d, February 2016. Available at GitHub https://github.com/furnishup/blueprint3d.

[2] Gavin Bierman, Martín Abadi, and Mads Torgersen. Understanding typescript. In *European Conference on Object-Oriented Programming*, pages 257–281. Springer, 2014.

[3] Damian Borth, Rongrong Ji, Tao Chen, Thomas Breuel, and Shih-Fu Chang. Large-scale visual sentiment ontology and detectors using adjective noun pairs. In *Proceedings of the 21st ACM international conference on Multimedia*, pages 223–232. ACM, 2013.

[4] Francisco Campuzano, Ioannis Doumanis, Serengul Smith, and Juan A Botia. Intelligent environments simulations, towards a smart campus. In *2nd International Workshop on Smart University*, 2014.

[5] James Cryer. *Pro Grunt. js*. Apress, 2015.

[6] Paul Davidsson. Agent based social simulation: A computer science view. *Journal of artificial societies and social simulation*, 5(1), 2002.

[7] Jos Dirksen. *Learning Three. js: the JavaScript 3D library for WebGL*. Packt Publishing Ltd, 2013.

[8] Jos Dirksen. *Three. js essentials*. Packt Publishing Ltd, 2014.

[9] Guillermo Fernández Escobar. Design and implementation of an agent-based crowd simulation model for evacuation of university buildings using python. Master's thesis, ETSI Telecomunicación, June 2017.

[10] Roland Hess. *The essential Blender: guide to 3D creation with the open source suite Blender*. No Starch Press, 2007.

[11] Karen Higginbottom. Workplace stress leads to less productive employees.

[12] David Masad and Jacqueline Kazil. Mesa: An agent-based modeling framework. In *14th PYTHON in Science Conference*, pages 53–60, 2015.

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

[14] Mr. Doob. Three.js code, May 2017. Available at GitHub https://github.com/mrdoob/three.js/.

[15] Sergio Muñoz López. Development of an emotion aware ambient intelligent system for smart offices application in a living lab and in a social simulated scenario. Master's thesis, ETSI Telecomunicación, June 2017.

[16] Nic Newman. Apple ibeacon technology briefing. *Journal of Direct, Data and Digital Marketing Practice*, 15(3):222–225, 2014.

[17] Jesse Prinz. Which emotions are basic. *Emotion, evolution, and rationality*, 69:88, 2004.

[18] Kevin Sowizral, Kevin Rushforth, and Henry Sowizral. *The Java 3D API Specification*. Addison-Wesley Longman Publishing Co., Inc., 1997.

[19] SweetHome3D. Sweet Home 3D - Draw floor plans and arrange furniture freely. Available at http://www.sweethome3d.com.