Dibris

Dipartimento di Informatica, Bioingegneria,
Robotica e Ingegneria dei Sistemi (DIBRIS)

Modelling, Simulating, and Reasoning on Crime Scenes with DigForSim and DigForReason

by

Alessandro Biagetti

DIBRIS, Università di Genova

Via Opera Pia, 13 16145 Genova, Italy http://www.dibris.unige.it/

University of Genoa

Laurea magistrale in Informatica

Modelling, Simulating, and Reasoning on Crime Scenes with DigForSim and DigForReason

Author:
Alessandro Biagetti

Supervisors:
Prof. Viviana Mascardi
Dr. Angelo Ferrando
(Univ. of Liverpool, UK)
Examiner:
Prof. Giorgio Delzanno

A thesis submitted in fulfilment of the requirements for the degree of Laurea Magistrale in Informatica in the

Department of Computer Science, Bioengineering, Robotics and System Engineering

December 2019

Declaration of Authorship

I, Alessandro Biagetti, declare that this thesis titled, 'Modelling, Simulating, and Reasoning on Crime Scenes with DigForSim and DigForReason' and the work presented in it are my own. I confirm that:

- This work was done wholly or mainly while in candidature for a research degree at this University.
- Where any part of this thesis has previously been submitted for a degree or any other qualification at this University or any other institution, this has been clearly stated.
- Where I have consulted the published work of others, this is always clearly attributed.
- Where I have quoted from the work of others, the source is always given. With the exception of such quotations, this thesis is entirely my own work.
- I have acknowledged all main sources of help.
- Where the thesis is based on work done by myself jointly with others, I have made clear exactly what was done by others and what I have contributed myself.

Signed: _____

Date: _____

UNIVERSITY OF GENOA

Abstract

Scuola di Scienze Matematiche, Fisiche e Naturali
Department of Computer Science, Bioengineering, Robotics and System Engineering

Laurea Magistrale in Informatica

Modelling, Simulating, and Reasoning on Crime Scenes with DigForSim and DigForReason

by Alessandro Biagetti

Evidence analysis is one of the Digital Forensics tasks and involves examining fragmented incomplete knowledge and reasoning on it, in order to reconstruct plausible crime scenarios. After one year of activities within the DigForASP COST Action aimed at exploiting Artificial Intelligence and Automated Reasoning for addressing evidence analysis, the lack of real data about real crimes emerged as a main obstacle for the experimentation and tuning of the DigForASP prototypes. In this Master Thesis we present DigForSim, an Agent-Based Simulation tool designed and implemented with Unity in order to produce realistic and controllable synthetic data, and a prototype for reasoning on them, DigForReason. DigForSim takes as input a user-defined 2D or 3D environment, and a well-structured file created by the user containing all the information about entities and their behavior; the aim of the application is to produce some files containing pieces of information about all the entities that were present on the crime scene as their positions and their names. These two files have to be passed, together with a list of locations, to DigForReason that provides an interface for querying the output files of the simulation.

Acknowledgements

Thanks to all the people that put up with me all these years and that helped me getting through all the difficulties. Thanks to my rapporteurs and to the supervisor of this thesis for being always available for clarifications and any kind of help with the work. A special thanks to my family and friends that are always with me.

Contents

ABMS **A**gent **B**ased **M**odeling (and) **S**imulation

ABM **A**gent **B**ased **M**odeling

ABSS **A**gent **B**ased **S**ocial **S**imulation

ACL **A**gent **C**ommunication **L**anguage

AI **A**rtificial **I**ntelligence

AOP **A**gent **O**riented **P**rogramming

APD **A**ntisocial **P**ersonality **D**isorder

BDI **B**elief **D**esire **I**ntention

CAS **C**omplex **A**daptive **S**ystems

DFIRLABS **D**igital **F**orensic (and) **I**ncident **R**esponse **LABS**

DFRWS **D**igital **F**orensic **R**esearch **W**ork**S**hop

DSL **D**omain **S**pecific **L**anguage

FIPA **F**oundation (for) **I**ntelligent **P**hysical **A**gents

GAML **G**eneralized **A**nalytical **M**arkup **L**anguage

GIS **G**eographic **I**nformation **S**ystem

GUI **G**raphical **U**ser **I**nterface

HPC **H**igh **P**erformance **C**omputing

IED **I**mpulse **E**xplosive **D**isorder

IoT **I**nternet **o**f **T**hings

JADE Java **A**gent **DE**velopment

JESS **J**ava **E**xpert **S**ystem **S**hell

KML **K**eyhole **M**arkup **L**anguage

KQML **K**nowledge **Q**uery **M**anipulation **L**anguage

KMZ **K**eyhole **M**arkup language **Z**ipped

MADIK **M**ulti **A**gent **D**igital **I**nvestigation Tool**K**it

MAS **M**ulti **A**gent **S**ystem

NIJ **N**ational **I**nstitute (of) **J**ustice

OOP **O**bject **O**riented **Programming**

RAT **R**outine **A**ctivity **T**heory

REPAST **RE**cursive **P**orous **A**gent **S**imulation **Toolkit**

RFID **R**adio **F**requency **ID**entification

SACI **S**imple **A**gent **C**ommunication **I**nfrastructure

SIFT SANS **I**nvestigative **F**orensic **T**oolkit

STIX **S**tructured **T**hreat **I**nformation **X**pression

SWGDE **S**cientific **W**orking **G**roup (on) **D**igital **E**vidence

Dedicated to my family

Introduction

In 2001, the attendees of the First Digital Forensic Research Workshop published a report entitled "A Road Map for Digital Forensic Research" where Digital Forensics (DF) was defined as "*the use of scientifically derived and proven methods toward the preservation, collection, validation, identification, analysis, interpretation, documentation, and presentation of digital evidence derived from digital sources for the purpose of facilitation or furthering the reconstruction of events found to be criminal, or helping to anticipate unauthorized actions shown to be disruptive to planned operations*" [1]. Evidence analysis involves examining fragmented incomplete knowledge and reconstructing and aggregating complex scenarios involving time, uncertainty, causality, and alternative possibilities. In order to overcome the limitations due to the lack of established methodologies for digital evidence analysis, the "Digital Forensics: Evidence Analysis via Intelligent Systems and Practices" (DigForASP[1]) Action was funded by the European Cooperation in Science and Technology (COST) and its activities started on 10/09/2018. The challenge addressed by DigForASP is to reason on evidences on crimes and their perpetrators collected from various electronic devices in order to reconstruct possible events, event sequences and scenarios related to the crime. The evidences elaborated in DigForASP are not raw ones (possibly damaged chips, mobile phones, etc) but aggregations of data extracted from these evidences, already transformed in some machine and human readable form, like logs or tables. In order to produce plausible scenarios and hypotheses by reasoning on these data, DigForASP adopts techniques rooted in Artificial Intelligence and Automated Reasoning. These techniques are reliable, verifiable, and explainable: these features are mandatory to make the results of the DigForASP reasoning tools available to, and useful for, law enforcement, investigators, public prosecutors, lawyers and judges. After one

[1]CA17124, https://www.cost.eu/actions/CA17124/, active from 10/09/2018 to 09/09/2022.

year of activities, and despite the DigForASP networking results which went beyond the initial expectations[2], it became clear that there is one main obstacle to the large scale experimentation of the protopypical tools designed and implemented within the COST Action: the *lack of real(istic) data about real(istic) crimes*. While digital forensics datasets exist, DigForASP needs more complex, precise and realistic data, carrying on as much contextual information as possible. Magistrates, digital forensics experts, and investigators access and manage these kinds of data in their daily work, but it is needless to say that such extremely sensible data are protected by government laws that prevent their usage – at least "as they are" – outside the legal process, even under Non Disclosure Agreements. Unfortunately, anonymizing these data and still retaining all the associated information needed for reasoning on them, is harder than expected. Let us suppose that the log of the movements of one suspect includes the information the he was at $(44°24'38.0''N, 8°55'44.2''E)$ at 10AM of 19/11/2019. To actually anonymize the data we should obfuscate the geographical information too. We might achieve this goal by translating all the coordinates of $(15°, 15°)$, for example, to keep the geometry of the crime scene and hiding the real places, but we would lose precious information: at $(44°24'38.0''N, 8°55'44.2''E)$ there is a bank that the suspect might want to rob, whereas at $(59°24'38.0''N, 23°55'44.2''E)$ there is sea. Adding the sole information that there is a bank nearby would not help: indeed, a few meters from $(44°24'38.0''N, 8°55'44.2''E)$ there are restaurants, shops, apartments, and one postal office: the suspect's goal might be one of them, but adding information on all of them to the translated crime scene is out of reach. The purpose of this thesis is the design and implementation of tools allowing a user friendly creation of virtual crime scenes. The latter are used to study and investigate the behaviour of people present on the scene at the crucial moment (when the crime happened). One of the best way to describe these scenarios is through simulations. Simulation-based approaches have been applied in many different scenarios, and are a widely studied technology, starting from old simulations based only on text, to new simulations enriched with virtual reality (or even Augmented Reality). To better understand their value, we may just think about flight simulators, largely used to train pilots, or warfare simulators, used to train military officers and law enforcement. These last simulations consist in different interactive scenarios projected on a screen (also on multiple screen) putting trainee in the condition of

[2]At the time of writing DigForASP involves 33 EU countries, one Near Neighbor Country, and almost 200 participants including law enforcement, lawyers, experts in Digital Forensics and experts in AI, both from the academia and the industry.

making some decisions, teaching them how to behave in certain situations [2]. We can find applications of virtual reality simulation also in medicine (for surgery and other sensitive intervention), and even though in its early stage, in higher education (to enrich the learning process). Second Life is a well-known "game" which has been used in such context, where the users can interact with a virtual world and perform activities that usually reflect their interests in real life [3]. Second life provides tremendous creative potential in that users can create content within the virtual world, including buildings, environments, and objects. An example of usage is given by Psychology instructors that can use SL as a space to meet with students and to create labs, buildings, and objects that can be used to learn psychology content and skills. Students' engagement with course content and sense of community with the class can be enhanced using interactions within SL [4]. Simulations are already used in many forensic areas, such as finding evidences, where suspect disks are given to trainees in order to be analysed, or size computer systems and finally delivering evidences, where a simulated court is recreated and the evidences can be delivered as videos of crime scenes' simulations [5]. Nonetheless, a simulation alone would not be enough, since in the forensic scenario the investigation and analysis of the crime scene are of greater importance. Because of this, using a combination of a specific set of tools, we created DigForSim, an Agent Based Modeling and Simulation (ABMS) tool which does not only provide a good visual representation but produce realistic and controllable synthetic data. DigForSim allows the user to decide how many entities of interest and passers-by to insert in the simulation, to choose how often to print their position and to explore the environment thanks to a camera that can move around the scene and to some other cameras placed on the head of entities of interest. At the end of the simulation, DigForSim returns a list of entities' positions at different timestamps and the list of entities of interest. This application allows representing custom made crime scenes only requiring some additional work by the user in designing the environment. In order to make the simulator output files usable, investigators should already have a general idea of the behaviour of the entities of interest to be put in the crime scene, naturally, without these information, a random scenario could still be created, but it would not be equally informative. In the case of an outdoor scene, there is the possibility to define the crowd inserting passers-by entities, and also for these entities is possible to define, to a limited extent, the way they behave in the scene. The reconstruction of the scene during the design phase should take into account all the information taken on the real scene of the crime

and one of the most important things to reconstruct a reliable scene of the crime is to analyze good quality photos and videos. These output files become the input of DigForReason, the other tool developed in this project, which analyses them and answers to questions regarding people movements on the crime scene; this second application provides a GUI (Graphical User Interface) that simplifies the user questioning phase. These two applications allow the user to create, visualize and evaluate the crime scene. More specifically, the evaluation allows the user to exclude impossible – or unwanted – scenarios, and focus only on the ones of interest. Extensive experiments have been carried out with DigForSim, which is the core of our task in DigForASP and in this work. DigForReason is an early-stage prototype which seeks to show some of the features that the DigForASP reasoning tools should support, and its aim is to mainly validate the DigForSim output. The Master Thesis work is organized in the following way, in Section 1, we introduce all the background notions required to fully understand the novel content of the thesis, in Section 2, we present all the requirements and design choices related to the development of the two tools, in Section 3, we present motivations that leaded us to the choice of certain tools instead of other, in Section 4 we present the tool used for DigForSim in the detail and all the implementation choices done for the application, in Section 5 we present and explain all the implementation choices done for DigForReason, in Section 6 we present all the experiments done for both the application developed and in Section 7 we briefly talk about how the two developed applications seems to work and about some possible upgrades that could be useful in the future.

Chapter 1

Background & Related Works

1.1 Digital Forensics

Jason Jordaan, principal forensic scientist at DFIRLABS (Digital Forensics and Incident Response Labs) [6], defines digital forensics as "*the identification, preservation, examination, and analysis of digital evidence, using scientifically accepted and validated process, and the ultimate presentation of that evidence in a court of law to answer some legal question*". To give the main idea, it is the science that takes care of processing digital data to detect useful evidence for investigations. Before the 1970 those who were defined as computer crimes were treated as common crimes using existing laws, but during the 1970s Computer forensics was born; this is a branch of forensic science (application of science to criminal and civil laws) and has then evolved in the one that we call digital forensics. Old computer forensics has been extended to networks (Network Forensics) and all digital devices, encompassing the analysis of materials obtained from them, like photos, videos, messages often related to computer crimes. Computer Forensics evolution has lead to digital forensics [7]. The exact origin of cybercrime is impossible to know; what is possible is to define the first major attack on a digital network and then use it as a reference point of event in the evolution of cyber-based crimes. In 1978 the State of Florida passed the "Florida Computer Crimes Act" [8]. This law, which included legislation against the unauthorized modification or deletion of data on a computer system and against damage to computer hardware including networks, might be the earliest American statute specifically against computer crimes. The maximum penalty for a single offense classified as a

Felony of the Third Degree was: "Up to 5 years of imprisonment and a fine of up to 5,000$ or any higher amount equal to double the pecuniary gain derived from the offense by the offender or double the pecuniary loss suffered by the victim". In 1981 Ian Murphy was the first person convicted of cybercrime. He hacked into the AT&T network and changed the internal clock to charge off-hours rates at peak times. He received 1,000 hours of community service and 2.5 years of probation, a mere slap on the wrist compared to today's penalties; this fact was the inspiration for the movie "Sneakers". As well as identifying direct evidence of a crime, digital forensics can be used to attribute evidence to specific suspects, confirm alibis or statements, determine intent, identify sources (for example, in copyright cases), or authenticate documents. Digital forensics investigations has a variety of applications; the most common is to support or refute a hypothesis before criminal or civil courts [9]. Depending on the device that has to be analyzed the Digital Forensics can be split in different sub-branches: computer forensics, network forensics, forensic data analysis, and mobile devices forensics.
Another important activity related to the digital forensic is Text mining; this can be useful to digital investigators for finding correspondences in a faster way; this because potential digital pieces of evidence has grown rapidly in this last years, in fact people nowadays are always more connected to the digital world, not just because almost everyone has a mobile phone, a laptop or a tablet, but because also cars, television and other objects of common use can be connected to the internet. From the analysis of all these devices there is the possibility to collect pieces of information related to relations between the victim and the suspect thanks to messages and phone calls if any, it is also possible to find complicit of the crime or finding pieces of information related to the reason of the crime. Best practices and methodology for forensic evidence collection, analysis, and reporting have been defined by Scientific Working Group on Digital Evidence (SWGDE) [10]. Additionally, the National Institute of Justice (NIJ) has a sub-practice devoted to Digital Forensics Standards and Capability Building [11]. The collection of digital evidence, as for physical ones, should always be performed to ensure that it will withstand legal proceedings. These are Key criteria for handling such evidence:

- The proper protocol should be followed for the acquisition of the evidence irrespective of whether it is physical or digital. Gentle handling should be exercised for those situations where the device may be damaged.

- Special handling may be required for some situations. For example, shut immediately down the device if it is actively destroying data through disk formatting. In other situations, it would not be appropriate to shut down the device.
- All artifacts, physical and/or digital should be collected, retained and transferred using a preserved chain of custody.
- All materials should be dated and time stamped, identifying who collected the evidence and the location it is being transported to after initial collection.
- Proper logs should be maintained when transferring possession.
- When storing evidence, suitable access controls should be implemented and tracked to certify the evidence has only been accessed by authorized individuals.

As far as concern analysis of file, it is better to work on an image of file data retrieved from devices, in order to preserve the original media. Some problems can arise if the file has been modified, deleted or encrypted but in all the three cases there is the possibility, working carefully, to obtain the right pieces of information [12].

1.2 Tools used for Digital Forensics

In this section we present some tools used for extracting and analyzing evidence from digital devices. Sometimes the analysis of evidence needs different tools than the extracting phase, as file viewers, hash generators and text editors:

- **Autopsy and the sleuth kit[13] (First documented version 2003):** Autopsy is a digital forensic platform that provides an interface for the sleuth kit. It is used by law enforcement, military, and corporate examiners to investigate digital devices, analyzing, for example, computer hard drives or smartphones. It can even be used to recover photos from the memory card of the user's camera or to analyze the file system image in an easier way. This is a quite simple technology to be used and is also extensible with third-parties modules and other already available with the initial kit:

- *Timeline Analysis:* Advanced graphical event viewing interface.
- *Keyword Search:* Indexed keyword search to find files that mention relevant terms.
- *Web Artifacts:* Extract history, bookmarks and cookies from different browsers.
- *Email Analysis:* Parses MailBOX format messages, such as Thunderbird.
- *File Type Sorting:* Group files by their type to find all images or documents.
- *Media Playback:* View videos and images in the application and does not require an external viewer.
- *Indicators of Compromise:* Scan a computer using STIX [1]
- *Multimedia:* Extract EXIF (Exchangeable image file format) from pictures and watch videos.
- *Data Carving:* Recover deleted files from not allocated space using PhotoRec[2]
- *Hash Filtering:* Flag known bad files and ignore the good ones.

This tool can generate HTML and excel artifacts, a tab delimited file report and a KML report for google earth. Autopsy runs background tasks in parallel using multiple cores and provides results to the user as soon as they are found. It may take hours to fully search the drive, but the user will know in minutes if his/her keywords were found in the home folder [14]. This tool is widely used because provides a large number of features as: *Multi-User Case, Registry Analysis using* ***RegRipper***[3]*, LNK File analysis, Email Analysis, Thumbnail viewer and more others.*

The sleuth kit can be used in two ways: as a collection of command line tools or as a C library that allows to analyze disk images and recover files from them. It is used behind the scenes in Autopsy as an integrated library or from command lines, but it is also used by other open source and commercial forensics tools [14].

[1] A structured language for cyber threat intelligence.

[2] PhotoRec is file data recovery software designed to recover lost files of any sort from digital camera memory. PhotoRec ignores the file system and goes after the underlying data, so keep working even if the media's file system has been severely damaged or reformatted.

[3] RegRipper is an open source tool, for extracting/parsing information as keys, values and data from the Registry and presenting it for analysis.

- **SIFT (SANS Investigative Forensic Toolkit) 2008 [15]:** This is an Ubuntu based toolkit that provides functionalities for an in-depth forensic or incident response investigation and supports the analysis of different evidence formats. It includes tools such as log2timeline for generating a timeline from system logs, Scalpel for data file carving, Rifiuti for examining the recycle bin. SIFT can match any modern-day incident-response and forensic tool suite which is also featured in SANS Advanced Incident Response course. It supports any format ranging from AFF (Advanced Forensic Format) to RAW (dd) evidence formats and even more.

- **Oxygen Forensic Suite (2008) [16]**: This is a forensic software for mobile phone, smartphone and tablet analysis developed by oxygen software. The suite can extract SMS, calendar events, event log and other pieces of information

- **Cellebrite (1999) [17]:** Cellebrite enables investigators to capture insights in today's complex, digital world. This digital intelligence platform provides a complete and objective picture of evidence, empowering agencies and investigators to solve and close cases. Nowadays, thanks to the Universal Forensic Extraction Device (UFED 2007) [18], this tool is used mostly for mobile phones, tablets, and smartphones, as Oxygen Forensic Suite. It has an integrated SIM reader, with wireless connection options also being integrated, such as IR and Bluetooth.

- **Forensic Toolkit (FTK first stable release in 2017)[13]:** Is a computer forensics software, made by AccessData, that, as Autopsy, can scan hard drive looking for various information; for example it can locate and eliminate e-mails or scan the disk to find text strings.

Other tools that can be mentioned are Crowdstrike [19], Volatility [20] and FTK imager (a standalone program of the FTK) [21], besides others that are presented in the web page: "Top 20 Free Digital Forensic Investigation Tools for SysAdmins"[4]. As we can see there is a large number of tools working for DF but some of them are better than the others. One of the best is the SIFT that can examine raw disks (i.e. the data in byte level secured directly from the hard disk drive or any other storage devices), multiple file systems and evidence formats. It is a Live CD including the tools one needs to conduct an in-depth forensic investigation or

[4]https://techtalk.gfi.com/top-20-free-digital-forensic-investigation-tools-for-sysadmins/.

response investigation. One of its main pros is that it is Open Source. Another good tool is the Sleuth Kit (+ Autopsy) that allows users/clients to give line commands to the machine. It allows the user to examine the disk images or the victim device and recover the damaged files. The last good tool that we present is volatility that has not been described before but, to make it short, is a framework that directly relates to the Advance Memory Analysis and Forensics. Advance Memory Analysis and Forensics are basically about analyzing the volatile memory in the victim system. With volatile memory/volatile data we refer to those data that change frequently and can be lost when the system restarts. This framework introduced the world to the power of monitoring runtime processes and the state of any system using data found in RAM (Volatile memory). Volatility also provides a unique platform that enables forensic research towards better efficiency which can be immediately taken up by digital investigators. This tool is used by the law enforcement of the country, the defense forces or any commercial investigators all over the world. Presentation of all these tools provides an up-to-date state of the art in the tools for supporting Digital Forensics investigation, but they have a different purpose from DigForSim and DigForReason they are used for extracting data from digital devices found on a crime scene. We can say that those tools want to answer to the questions: "Who?" and "Why?" while DigForSim and DigForReason wants to answer to the question "How?"; DigForSim allows us to create a simulation providing a log file that shows people movements in the scene, while DigForReason allows us to analyze this log file in order to obtain information on possible interactions between people. These two tools introduce new features in the DF field, entrusting the task of discarding impossible scenarios to an application.

1.3 Multi Agent System

In computer science, an agent is defined as a computer program, coupled with its environment, that acts on behalf of the user or of the program that executed it. Such "action on behalf of" implies the authority to decide, if any, which one is an appropriate action [22] [23]. For the agent definition is important to remember that: all the agents are programs but not all the programs are agents. Those computer programs are usually known as bots and can be embodied in other applications or software as chatbot executing on different devices (e.g Siri). Computer

agents may possess some human-like qualities, that leads to link them with the AI field, as personality, ability to reason or to have a sensible conversation; those are defined as intelligent agents and used as shopping bots for retrieving information about goods and services, in computer games, to find information on specific topics or for monitoring some activities. It is possible to distinguish two main usage for them: the most general is to mean an autonomous, reactive and proactive computer system that is able to communicate with other agents via some ACL (Agent Communication Language) while a more specific usage is to mean a computer system that is either conceptualized or implemented in term of concepts more usually applied to humans, the so called BDI architecture [24]. We can also distinguish agents in three types: Passive Agents that have no objectives[25], Active Agents[25] that have simple objectives and Cognitive Agents (those can do complex calculations for reaching complex objectives); each one of them has at least one thread of control but it could have more [26]. Agents properties are:

- **Autonomy:** It should be able to work in a certain environments without help from other entities in the system
- **Reactivity:** It should be able to react to events in the environment, changing the information that it keeps in memory and its needs
- **Proactivity:** It should be able to generate a plan for achieving its goal

Agents are the main component of the AOP (Agent Oriented Programming). Considering how agents communicate between each other and cooperate AOP can be seen as a specialization of the object-oriented programming (OOP) paradigm. Both frameworks view a computational system as made up of objects with states, that pass messages to one another and have individual methods for handling incoming messages. AOP fixes the form of the agents' state (their mental state), fixes the form of messages, distinguishing, in the spirit of speech-act theory [27][28], between informing, requesting, offering, and so on, and it places constraints on methods for responding to incoming messages. There are two main formalisms for programming with agents: Agent0 and BDI. Agent0 is one of the languages for programming with agents that also provides a particular agent architecture and built-in inter-agent communication model[29]. A BDI agent is defined by its own beliefs desires and intentions:

- **Beliefs:** What an agent think about the environment, its knowledge

- **Desires:** What an agent want to achieve
- **Intentions:** How an agent want to achieve his objectives

The notation MAS (Multi Agent System) refers to multiple agents that communicate, cooperate and revise their knowledge to solve a complex task. Revision of knowledge depends on what happens around the agents and on what other agents tell through communication. Each agent, taken individually has just a limited amount of information on what happens around him and that is why it can communicate and increment its knowledge, but how do they communicate? Studies on agent communication have started around the 1970 and led to three widely used method: (1) Speech act with which an agent can act as a speaker (S) producing utterance to change beliefs of the hearer[28] (H), (2) message passing with which agents directly message each other with point-to-point or broadcast communication and (3) blackboards whit which agents can collaboratively share data with each other using a central repository called Blackboard.
Multi agent systems can manifest self-organization as well as self-direction and other complex behaviors even if single agents' strategies are simple. When agents can share knowledge using any agreed language, within the constraints of the system's communication protocol, the approach may lead to a common improvement. Examples of languages are KQML (Knowledge Query Manipulation Language) or ACL. For the simulation that we want to create a MAS system seems to be a good option, since the fact that lots of entities should have to live and work together. Salient features of MAS, including efficiency, low cost, flexibility, and reliability make them an effective solution for solving complex tasks. The efficiency of these kinds of systems stems from the division of complex tasks assigned to them into multiple smaller tasks, each of which is assigned to a distinct agent [30]. Naturally, the associated overheads, e.g., processing and energy consumption, are amortized thanks to the multiplicity of agents, which often results in cheap solution as compared to an approach where the entire complex problem has to be solved by one single powerful entity. Each agent can solve the allocated task with any level of predefined knowledge, which introduces high flexibility [31]. The distributed nature of problem-solving adopted in MAS also imparts high reliability. In the event of agent failure, the task can be readily reassigned to other agents. Here we show some MAS applications:

- *Computer Networks: Agents are widely used to overcome the increasing complexity of those kind of networks*
- *Robotics: Using agents for robotics has been studied for over two decades with the first article published in 1996 outlining the the pros and cons of agents in robotics [32].*
- *Modeling of complex systems: Modeling complex dynamic systems is expansive and incurs significant overhead because it needs powerful modeling platforms and high complexity. The flexibility, autonomy, and scalability afforded by agents make Agent Based Modeling (ABM) a cheap and low resource solution to this task.*
- *Modeling Cities and built environments: In recent years researchers have focused their attention on the usage of agents in managing cities and buildings. In a city, an unorganized distribution of freight increases the cost, pollution, and congestion. This is why an agent-based method to address this challenge using six agents namely: RFIDG agents that receive data from RFID (radio-frequency identification) reader, retailer, supplier, carrier, network, and city agents [33].*
- *Smart grids: Agents are used to address multiple challenges of smart grids.*

1.4 Agent-Based Modeling and Simulation

ABMS (Agent Based Modeling and Simulation) [34] has its origin in the ABM (Agent Based Modeling) [34], a computational model used for simulating actions and interactions of autonomous agents with the aiming to represent their effects on the system as a whole. ABM is related to, but distinct from, the concept of MAS or multi-agent simulation; in that, the goal of ABM, is to search for explanatory insight into the collective behavior of agents obeying simple rules, typically in natural systems, rather than in designing agents or solving specific practical or engineering problems. ABMS consists of a set of agents and a framework for simulating their decisions and interactions. It is related to a variety of other simulation techniques, including discrete event simulation and distributed MAS. Although many traits are shared, the difference of ABMS is its focus on finding the set of basic decision rules and behavioral interactions that can produce the

complex results experienced in the real world. An agent is thus a software representation of a limited decision-making unit that follows few rules, depending only on imperfect local information, to decide what the agents will do next. Scientists conduct ABMS using agents not only to make predictions on events but to explain/guide data collection, illuminate core dynamics, limit outcomes to plausible ranges, offer crisis options and educate the general public. ABMS can capture much of the complexity and many of the characteristic features of CAS (Complex Adaptive Systems) [35] (non-linearity, unpredictability, butterfly effects and so on) in a manner previously considered impractical and the underlying notion that "systems are built from the ground-up" differently from the top-down systems of system dynamics [34]. Specified agents' rules do not need to be known in detail, in fact simulated systems are used for observing, analyzing and explaining systems' outcomes that could be not intuitive. Starting from the beginning of the 2000s there has been a growing interest in the area of ABSS (Agent Based Social Simulation), which integrates approaches from agent-based computing, computer simulation, and the social sciences; here ABM and ABMS may be useful. ABSS has been used for a better understanding of social phenomena using the agent-based modeling paradigm. The usage of ABSS is very effective since the fact that combines the advantages of the agent paradigm with those of social simulations, as the possibility to perform social "experiments" without much effort, and what makes agents very useful is the possibility to make them intelligent, for example using the BDI architecture, allowing the agents experts to make complex simulation that could give better results. ABM can be used across a variety of application domains: from modeling agent behavior in the stock market, supply chains, and consumer markets, to predicting the spread of epidemics, the threat of bio-warfare, and the factors responsible for the fall of ancient civilizations.

1.5 Tools used for MAS and ABMS

Examples of applications using MAS could be some videogames, where the more the AI is challenging and accurate the better the game is, in academic research, in industry, where they can offer distributed intelligent management and control functions with communication, cooperation, and synchronization capabilities, and also provide for the behavior specifications of the smart components of the system [36]. MASs have also been used in films [37], for example in "*The Lord of the*

rings" for creating rows of soldiers during battle scenes. Other applications [38] include transportation [39], logistics [40], graphics, manufacturing, smart grids and GIS. There are many tools for working with MAS, creating application for different fields and here we present an overview of them:

- **Jade (early 2000'):** Java Agent Development Framework is a software Framework fully implemented in Java. It simplifies the implementation of MAS through a middle-ware that complies with the FIPA (Foundation for Intelligent Physical Agents)[41][5] specifications and through a set of graphical tools that support the debugging and deployment phases. There is also BDI4JADE that is a platform that implements the BDI architecture, as a BDI layer implemented on top of JADE. Models based on jade can be distributed across machines and configuration can be controlled thanks to a remote GUI (Graphical User Interface). Jade is not a suitable tool for creating simulations [42]

- **AnyLogic (2000) [43]:** This is a tool created for simulations, it supports agents, without reasoning component, and provides a graphical modeling language; it is named AnyLogic, because it supports three well-known modeling approaches: system dynamics[44], discrete event simulation[45] and Agent-based modeling[46]. It is cross-platform since the fact that it works on Windows, MacOS and Linux[47]. It gives the possibility to extend simulations models with java code and a large library full of example and behaviors already created as:

  - *The Pedestrian Library*: This is Dedicated to simulating pedestrian flows in a physical environment. It allows to create models of pedestrian-intensive buildings (like subway stations, security checks etc.) or crowded streets.
  - *The Fluid Library*: This allows the user to model storage and transfer of fluids, bulk goods, or large amounts of discrete items, which are not desirable to model as separate objects.
  - *The Road Traffic Library*: This allows users to simulate vehicle traffic on roads. The library supports detailed, physical level modeling of vehicle movement. Each vehicle represents an agent that can have its own behavioral patterns inside.

[5] FIPA is an IEEE Computer Society standards organization that promotes agent-based technology and the interoperability of its standards with other technologies (http://www.fipa.org/).

  - *The Rail Library*: This supports modeling, simulating, and visualizing operations of a rail yard of any complexity and scale.

- **REPAST (2000) [48]:** The Recursive Porous Agent Simulation Toolkit is a widely used free, open-source, cross-platform, agent-based modeling and simulation toolkit that uses ReLogo as DSL (Domain Specific Language) for developing agent-based models. REPAST has a java based model called Repast Symphony, and a C++ one called Repast HPC (High Performance Computing) and both have built-in adaptive features such as genetic algorithms and regression. REPAST has an open design that can be used to develop a wide range of agent architectures, also the BDI one. ReLogo is a solid language that combines the sophisticated and powerful ABM infrastructure and capabilities in the Repast Suite with the ease of use of the Logo programming language and its associated programming idioms. This language combines also a large number of concepts as object-oriented programming, integration of existing libraries, statically and dynamically typed languages, some DSL, and the use of integrated development environments; it can be used to create simple and complex ABMs.

- **Zeus toolkit (2000)[49]:** It is a toolkit for constructing collaborative multi-agent applications. ZEUS represents a synthesis of established agent technologies to provide an integrated environment for the rapid development of MAS and its agents encapsulate the BDI model. Beliefs are translated to facts, desires to goals, and intentions to commitments[50]. Zeus gives also the possibility to use a visual environment for capturing user specification of agents.

- **MASON (2003) [51]:** This is developed by Computation Laboratory in conjunction with the Center for Social Complexity of George Mason University's Evolutionary. It is a fast discrete-event multi agent simulation library cored in Java, designed to be the foundation for large custom-purpose simulations, and also to provide more than enough functionalities for many lightweight simulation needs; MASON continues to be maintained and kept up to date. Its name, as well as referring to the parent institution, derives from the acronym Multi-Agent Simulator Of Neighborhoods (or Networks). It contains both a model library and an optional suite of visualization tools in 2D and 3D.

- **Netlogo (1999) [52]:** This is an agent-based programmable modeling environment with which is possible to create multi agent systems. This tool does not provide a true parallelism that is just a simulated property of the system; anyway, as other logo programming language based applications, it provides an integrated modeling environment. Thanks to its environment and its GUI it is a suitable option for creating simulations, and it is also easy to use through the browser. This tool also provides functions for importing maps from external sources (as google maps), creating real scenarios and giving the possibilities to obtain results more suitable for a realistic output.

- **GAMA (GIS Agent-based Modeling Architecture 2007) [53]:** This is a modeling and simulation development environment for building spatially explicit agent-based simulations; it uses GAML (Generalized Analytical Markup Language), a high level and intuitive language that is coded in Java and gives the possibility to create agent based environments. GAMA provides some additional libraries useful for certain domains and allows to easily load GIS (Geographic Information System) file; in addition with this tool is possible to reproduce more views for a system (both in 3D or 2D), in order to highlight different information. Thanks to an external Plug in it supports BDI concepts [54]

- **JASON (First stable Release in 2007) [55]:** This is an open-source platform for the development of multi agent systems that follows the BDI architecture. An extension of the AgentSpeak (an agent-oriented programming language) is used for programming the behavior of individual agents. JASON is developed in Java and allows the customization of most aspects of an agent or a multi-agent system. It comes as a plugin for either jEdit or Eclipse, and different infrastructures for the deployment of multi agent systems, for example using JADE or SACI (Simple Agent Communication Infrastructure) as an agent-based distributed system middleware. As Jade this does not suit well with simulations.

- **Active Component - Jadex [56]:** This is a framework for creating goal-oriented agents that follow the BDI model. It provides a rational agent layer that sits on top of a middleware agent infrastructure and allows for intelligent agent construction. It uses object-oriented concepts and technologies such as Java and XML for the creation of multi agent systems. Its reasoning

engine allows the realization of goal deliberation mechanisms and additionally facilitates application development by making results from goal-oriented analysis.

Since the fact that we are talking of simulations and considering that DigForSim should provide also a good visualization we want to give an overview of some possible 3D modeling tools that could be useful for recreating the scene of the crime[57] starting from measurements done on-site and from some digital photos of the scene:

- **3Dmax:** A 3D modeling software that allow to model objects and to create animations; it provides a large quantity of tool for simulating dynamic behavior of bodies.[58]
- **Poser Pro:** This is not a tool for modeling the crime scene but it is just for modeling human bodies, it provides a large number of prefabricated models that could be personalize to make them similar to the real ones.[59]
- **Rhino:** This is one of the more advanced 3D modeling software, it is based on NURBS (Non-Uniform Rational Basis Spline) and it provides a good management of curve lines using mathematical models and manipulating nodes.[60]
- **Blender:** Is an open source cross-platform tool for 3D modeling.[61]

1.6 Evaluating Tools for Reasoning on Digital Forensics Cases: existing Benchmarks and Competitions

Given the increasing prevalence of technology in our daily life, there is a corresponding increase in the likelihood of digital devices being pertinent to a criminal investigation or civil litigation. Here is why there is always a greater need for DF expertise in investigations. The variety of new digital evidence sources poses new and challenging problems for the digital investigator from an identification, acquisition, storage and analysis perspective[62]. In 2013 Sriram Raghavan outlined five major challenge[63] areas for digital forensics gathered from a survey [62]:

- **The complexity Problem:** Derived by the acquisition of data at the lowest (i.e. binary) format with increasing volume and heterogeneity, which calls for sophisticated data reduction techniques prior to analysis.
- **The diversity problem:** Derived by the increasing volume and type of data and by the lack of standard techniques to examine them
- **The volume problem:** Derived by the increasing storage capacities, the increasing number of devices that store information and by a lack of sufficient automation for analysis.
- **The consistency and correlation problem:** Derived by the fact that existing tools are designed to find fragments of evidence, but not to otherwise assist in investigations.
- **The unified time lining problem:** Derived by the fact that multiple sources present different time zone references, timestamp interpretations, clock skew/drift issues, and the syntax aspects involved in generating a unified timeline.

Other researches have led to other challenges in the following years and nowadays these aim, as this project does, to the creation of applications that can help in understanding what effectively happened on a crime scene and not anymore to the upgrade of technologies for DF investigations. An example can be done by the two challenges proposed by the DFRWS (Digital Forensic Research WorkShop) in years that goes from 2017 to 2019 that dealt with DF and IoT (Internet of Things) [6][64][65]. Consumer-grade "Smart" devices are increasing in popularity and scope. These devices and pieces of information collected by them are potentially interesting for digital investigations but also come with several new investigation challenges. Both 2017 and 2018/2019 challenges seek to advance the state-of-the-art in IoT forensics by focusing the community's attention on this emerging domain. The scenario of the DFRWS 2017 challenge[7] was: *a woman (Betty) has been murdered. The murder was called in by Betty's husband (Simon), who claims to have been at home at the time.* On the crime scene, there are different devices that could provide some interesting pieces of information for investigations, as Smart TV Rasberry Pi, Betty's Samsung Note 2, Simon's Samsung Note 2, Google OnHub

[6] IoT refers to the interconnection via the Internet of computing devices embedded in everyday objects, enabling them to send and receive data.

[7] 2017 challenge (https://github.com/dfrws/dfrws2017-challenge/).

Diagnostic report, Amazon Echo Cloud Data and MDS (Acme, Inc.) Smarthome Network Dump. The scenario of the DFRWS 2018/2019 challenge[8] was: *On 17 may 2018 at 10:40, the police were alerted that an illegal drug lab was invaded and unsuccessfully set on fire. The police respond promptly, and a forensic team is on scene at 10:45, including a digital forensic specialist. The owner of the illegal drug lab, Jessie Pinkman, is nowhere to be found. Police interrogate two of Jessie Pinkman's known associates: D. Pandana and S. Varga. Pandana and Verga admit having access to the drug lab's WiFi network but deny any involvement in the raid. They also say that Jessie Pinkman's had the IoT security systems installed because he feared attacks from a rival gang and that Jessie kept the alarm engaged in "Home" mode whenever he was inside the drug lab. Within the drug lab (** see diagram) the digital forensic specialist observes some IoT devices, including an alarm system (iSmartAlarm), three cameras (QBee Camera, Nest Camera and Arlo Pro) as well as a smoke detector (Nest Protect). An Amazon Echo and a WinkHub are also present. The digital forensic specialist preserves the diagnostic logs from the iSmartAlarm base station, and acquires a copy of the file system of the WinkHub. He also collects the iSmartAlarm and Arlo base stations to perform an in-depth analysis at the forensic laboratory. The digital forensic specialist also notices that the QBee Camera seems to be disabled, so he collects a sample of the network traffic. Back at the forensic laboratory, the digital forensic specialist uses the bootloader to collect a memory image of the two base stations as well as an archive of some folder of interest of the Arlo base station. Jessie Pinkman's Samsung Galaxy Edge S6 is found at the scene, likely dropped during the raid. The digital forensic specialist acquires a physical image of this Samsung device.* It was also provided a map of the drug laboratory. Also here there are different devices from which investigators have to try to extract some useful information, as: Jessie Pinkman's Samsung phone, iSmartAlarm – Diagnostic logs, Amazon Echo and others. Both these challenges aimed to motivate new approaches to forensic analysis and had four levels of participation:

1. Evaluating and Expressing Conclusions: Assigning the probability of the results given two competing propositions (e.g. The husband killed the wife, some unknown person did).

2. Device Level Analysis: Developing methods and tools to forensically process digital traces generated by IoT devices, including on mobile devices.

[8] 2018/2019 challenge (https://github.com/dfrws/dfrws2018-challenge/).

3. Network and Cloud Level Analysis: Developing methods and tools to forensically process digital traces generated by IoT devices on networks and cloud systems.

4. Correlation and Analysis: Developing methods and supporting tools that combine information from various data sources and automatically compute, visualize, or otherwise expose patterns of potential interest.

With respect to all the competition seen, the application developed for this project, DigForSim, works differently, trying to recreate events happened on the crime scene, so it needs pieces of information about who was on the scene, what he did on the scene and it needs also some hypothesis on how he had moved on the scene; all these pieces of information are needed in order to create a simulation that could give an idea of what could effectively have happened. If we want to compare DigForSim with old challenges we can say that it does not have to analyze information saved on devices present on the scene, but, considering the 2018/2019 scenario of challenges, it has to simulate movements of the pyromaniac (following different hypothesis on where he was at the beginning of the events and at the end of them) in order to understand possible interactions between people on the scene or with the environment. However if we compare the last challenge whit the second application developed for this project, DigForReason, we can say that there is something similar between the two, in fact both could answer similar questions. DigForReason, in fact, can say where two people have been encountered, where a person was at a certain time, who has been in a certain place at a certain time and so on; all these questions are similar to the ones at which the last challenge had to answer: At what time was the illegal drug lab raided?, How was the QBee camera disabled?, Could any of the two friends of Jessie Pinkman have been involved in the raid? and if yes: Which friend?, What is the confidence in such a hypothesis?. As we can see questions are very similar.

1.7 Related Works

ABMS turns out to be extremely appropriate to analyze phenomena within the criminology and digital forensics domain. The state of the art in agent-based modeling of urban crime has been presented in an article published in 2019[66], whose

authors observe that Agent Based Modeling (ABM) "*has the potential to be a powerful tool for exploring criminological theory and testing the plausibility of crime prevention interventions when data are unavailable, when they would be unethical to collect, or when policy-makers need an answer quickly.* An example of ABMS usage in this field is MADIK (Multi Agent Digital Investigation toolKit), a tool based on a four-layer multi-agent architecture, as a metaphor to the organizational hierarchy levels, which is divided into strategic, tactical, operational and specialist levels. This architecture was developed with a blackboard approach, implemented over the JADE framework, using JESS(Java Expert System Shell). MADIK aims to help experts during the forensic examination process. The specialized intelligent software agents already implemented include:

- *HashSetAgent:* It calculates the MD5 hash from a file and compares it with its knowledge base, which contains sets of files known to be ignorable(e.g system files) or important (e.g contraband files).
- *FilePathAgent:* It keeps on its knowledge base a collection of folders which are commonly used by several application that may be of interest to the investigation like P2P (peer-to-peer), VoIP and instant messaging applications.
- *FileSignatureAgent:* It examines the file headers (the first 8 bytes of the file), to determine if they match the file extension.
- *TimelineAgent:* It examines dates of creation, access and modification to determine events like system and software installation, backups, web browser usage and other activities, some which can be relevant to the investigation.

Managers of different layers use different calculations to distribute work, according to organizations' priorities, resources' availability and capability, file size and case evidence related to different areas. Calculated resources are passed to the tactical managers that have to calculate each evidence they have to examine, in order to define how many of the available computers will be assigned to each one.

All the examples of MAS used in digital forensics are not related to the physical simulation of the crime, as DigForSim wants to do, but are related to the analysis of digital devices as already said, and MADIK is a case in point. Similar works to DigForSim have been done for general social simulations, for example, to represent

how certain behavior are carried out, as a BDI agent representing a criminal with IED (Impulse Explosive Disorder) that is a disorder of impulse control that leads to aggressive behavior for 10/20 minutes [67]. The BDI architecture has been used also for other simulations based on psychological states of criminals, comparing three personality types: the already said criminals affected by IED, others affected by APD (Antisocial Personality Disorder) and the violent psychopath [68]; this experiment showed a modeling approach at the cognitive level for different types of violent criminal behavior. Dynamical models were incorporated, addressing psychological factors relating to desires, such as levels of anxiety and excitement arousal, empathy or theory of mind, impulsiveness and aggressiveness in order to show which kind of people (criminal in this case) are more usual to certain behavior. In the last decade, a great effort has been done also for studying the spatio-temporal dynamics of crime, also because they are one of the main research interests within Environmental Criminology. Relevant questions within this area are related to movements and interactions between three types of agents: passers-by, criminals, and guardians, where the first ones are the potential victims. These studies are not strictly linked with the representation of a single crime scene, but they want to understand how crimes are diffused in certain zones; there are two approaches used for doing this:

- **simulating crime dynamics in existing cities** that could be beneficial because people can apply it to new real world situations.
- **Abstracting from empirical information** that is hopefully able to predict patterns of spatio-temporal dynamics of crimes.

These studies are based on the RAT (Routine Activity Theory) [69] theory that states: *a crime will occur when a motivated offender meets a possible target with no guardians that can intervene* [70]. Simulations have been used also in other fields related to criminology as for studying perceptual deterrence, for studying psychological processes that trigger violent behavior[71][72] and moreover for studying social learning of delinquent behavior in adolescents[73]. All these works are focused on studying general aspects of crimes, giving an idea of "how people act" and "why they act like that", studying the psychological aspects of criminals, while DigForSim wants to show what has happened on a specific scene of a crime,

showing and marking movements of people in order to analyze them in a second moment. Pieces of information obtained could be used for excluding some impossible scenarios.

Chapter 2

Requirements and Design

2.1 DigForSim Requirements

The user of this application should be the investigator that is analyzing the behaviour of people that were present on the scene of the crime, with the aim of getting information on how things have carried out and on possible interactions between people. Considering the need of simulating something it might be necessary to create an application providing a good 3D or 2D visualization. One of the main functionalities that DigForSim should provide is the possibility to create an environment or to import it from external tools, increasing application flexibility. Going more in the detail it should be a good idea to use a tool that allows the creation of well structured 3D environments; this would be useful because the scene could need to represent an environment divided on more plans (as a two floor apartment). The advantage of a 3D model is the representation of the environment as a whole, avoiding the need to create different floors as plains placed side by side, in addition in the 2D case both planes have to pass information between each other when an entity moves from one to the other. Another important thing that the model should provide is an identifier for each zone, for example if the 3D model represents a house, a part of it should be named kitchen, another one should be named living room and so on; this results useful for the analysis of the simulation results, in fact it should be possible to define where an entity has moved during the simulation. Considering the usage of a 3D model it could be good to have a camera allowing the user to navigate in the scene, in order to

analyze the simulation during the execution. This camera could be structured in two ways:

- A camera able to move around the environment, to zoom in and zoom out.
- A playable camera that can move inside the environment (this have to be disembodied).

The second option is not typical of a simulation, this could result in something more similar to a videogame; it could anyway result in a good feature to insert a second camera above each entity, allowing the user to see the environment through the entities' eyes. The simulation will need also to know who was present on the scene and their behavior in order to show something that could be useful, so it has to take pieces of information from some files. Since the fact that on a crime scene, apart from criminals, police man and the victim/victims, it is possible to find also some passers-by, it could be nice to divide entities in two main categories: those in which investigators are interested and all the others. First ones are those that have to be well defined; each one with different properties defined by the user, while the others are copies of the same entity that should move randomly in the environment. Considering that it is not cheap to define movements point by point, it might be necessary to use a structure allowing entities to define by themselves a path for reaching their objectives avoiding major obstacles, as a wall could be; in addition, since the fact that it is possible to have an environment with a high population the application should be able to manage a great number of entities without problems, and tools used for MAS seems to be the best option. Considering that DigForSim should be a simulation for which is not explicitly requested to make differentiated human like entities it would be nice to be able to distinguish them thanks to their color; that is why it could be a good feature to make the user able to choose a different color for different entities, obviously this choice shall be done when setting up the simulation. Once that the simulation has been executed it would be necessary to output all the movements done by entities in order to make investigators able to analyze them, getting some useful information; the best way to do this would be to create a file with positions of each entity at different instants of time. Considering that the best time interval is not the same for all the possible simulations also this parameter should be part of the set up phase. In addition to the file with entities' positions the application could return a file with the population of the environment.

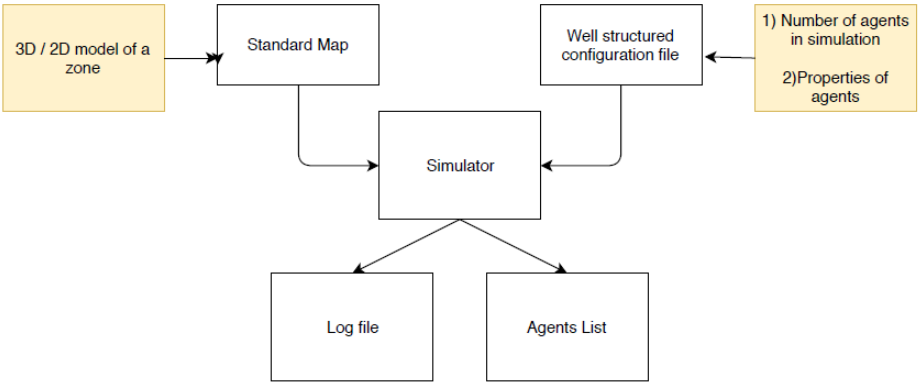

FIGURE 2.1: DigForSim diagram.

2.2 DigForSim Design

Considering that this application will have to read a configuration file in order to set the simulation, it will be necessary to give to the user the possibility to choose which configuration file to use; for this purpose the simulation part of the application should be preceded by a menu where enabling the user to choose the configuration file to use between the ones contained in a certain directory. Talking of the simulation itself, we want entities able to move randomly and others able to move following a defined list of objectives. Realism can be helped by using a 3D model, allowing people to move between different levels of the environment without being teleported from a level to another considering also the time taken by an entity to move up or down some stairs (or using a lift). Since the fact that the environment map has to be created in 3D, there are some pieces of advice that could be useful, whether that is created directly in the DigForSim application, whether that is imported by an external source: using planes for the ground (and in case for stairs) and cubes for walls. If the designer of the map wants to create visually better objects he could create some 3D models by itself. The next step will be to insert objects that could be important inside the scene, as a mobile phone or a computer, and also for these ones there is the possibility to find some texture on line or simply to create them (this should include some more work for a map designer). All these objects have to be inserted manually, because it would impossible to define, in a good way, the true position via configuration file. The main problem is given by the scaling of the environment, in fact it depends directly from the designer, that is why it might better to do all the design phase manually. There should be to define some zones in which entities can be spawned, giving to

the user the possibility to choose them through the configuration file, saving the designer from modifying manually each time the entities starting position; these spawn locations might be objects already present in the model, or objects inserted just as spawn locations. For what concerns the work done on texture research or on texture creation in order to give a better visual effect, this is not necessary for the aim of this Master Thesis; so also simple 3D geometric objects can be used (cubes, cylinders, planes, capsules and so on) to represent any object in the scene. Since the fact that results of the simulation should also show names of locations in which people are moving, each plane representing a part of the ground has a string value defining its name, also in order to help DigForReason in giving more detailed answers. As we already said in Section 2.1 for representing a great number of entities moving, each one, by itself a MAS could be a good idea, so we can start to define entities as agents. Agents should be of two types: those in which investigators are interested, that from now on will be called *special agents* and the others that represent passers-by, that will be called *wandering agents*. Both this type of agents have some properties which values are set from the configuration file:

- **Wandering Agents:**
  - **Wandering radius:** This defines the radius of the area where agents move.
  - **Time:** This defines the time these agents spend in the area defined by the radius previously mentioned. After the defined time agents recompute the area, changing the zone where they are going to move.
  - **Speed:** This defines the speed at which each agent moves. Wandering agents are divided in three types: fast, medium speed and slow.
- **Special Agents:**
  - **Name:** A parameter that defines their name making them recognizable.
  - **Speed:** A parameter that defines their speed.
  - **Color:** A parameter that defines their color.
  - **Starting Coordinates:** A parameter that defines their starting position.

– **Movements:** A list of parameters that defines their movements in the environment.

Talking about the cameras cited in the previous section: the main one allows the user to move around the scene and the others provide a way to see the world through "the eyes" of Special Agents. The best way to design a camera that has to move around an environment is creating a virtual trackball[1]. The number of cameras of the second type will be equal to the number of special agents in the simulation, and each one will be mounted on the top of each special agent, providing a first person view for the simulation. This second type of camera is not truly necessary for the purpose of our simulation, but it could become useful for future updates.

2.3 DigForReason Requirements

This application should be used to analyze the output of DigForSim in order to obtain answers to some predefined questions, giving useful information for the investigations. Some possible questions could be: "Is possible that *entity1* and *entity2* have encountered between 3.00 and 3.10?", "Where was *entity1* between 2.00 and 2.20?", "Did *entity1* went through *location*?" and so on. DigForReason needs different input files for reasoning: the list of locations of the 3D environment, the main output of the simulation, that gives all the possible pieces of information about agents' movements, and the secondary output of the simulation, that is a text file containing names of all the entities in the scene. DigForReason differently from DigForSim, will require more interaction from the user, that will have to click buttons, fill text field, choose files and so on; that is why DigForReason will need a good GUI (Graphical User Interface) to be easy to use.

The bigger text box in Figure 2.3 is the communication box and shows the answer to the chosen question. A good idea could be to insert also the possibility to print all the content of the communication box on a text file to keep a record of what have been "discovered".

[1] A trackball is a sphere built around an object (that is in its center). This sphere allows the user to rotate the object, exploring it. In our case the camera moves on the surface of this sphere in order to rotate around the object. It provides also the possibility to zoom in and to zoom out [74].

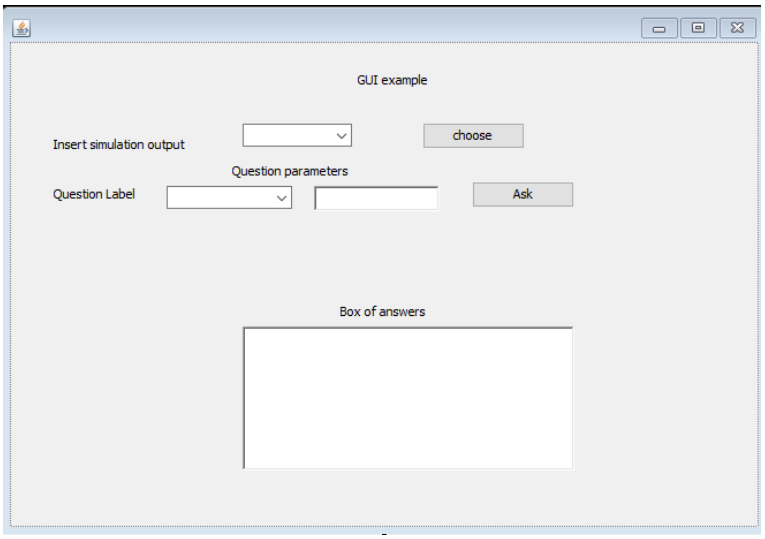

FIGURE 2.2: DigForReason GUI example.

2.4 DigForReason Design

The GUI of DigForReason should be a window with everything immediately visible. The main idea is to put in the top part of this window some dropdown menus allowing to choose between different files in different directories to make DigForReason know on which files it will have to work. There will be the possibility to change the files on which the user is working during DigForReason execution, without the need of restarting it. All these files should be previously inserted in the correct directories, in order to make the application able to find them. Chosen files for the analysis need to be related the one with the other, in fact if the analyzer takes as input the output file of a *scene A* and the population of a *scene B* results might not have sense, and in the worst case this mismatch will result in a wrong answer because population of a certain scene, in the most cases, is different from the one of another scene; in case of some entities with the same name the behaviour would be surely different from the one that we are expecting (also because the environment could be different). Immediately after the file selection area there will be a list of all the possible questions, some of them have been already seen in Section 2.3; since the fact that the user will have to choose some parameters for asking questions, each one is followed by some dropdown menus and text boxes equal to the number of parameters that the question needs. On the right part of the window there will be the communication box, that will show the answers of the analyzer at runtime. In order not to loose the output shown in the communication box, each time an answer is given, it will be copied in a file and removed

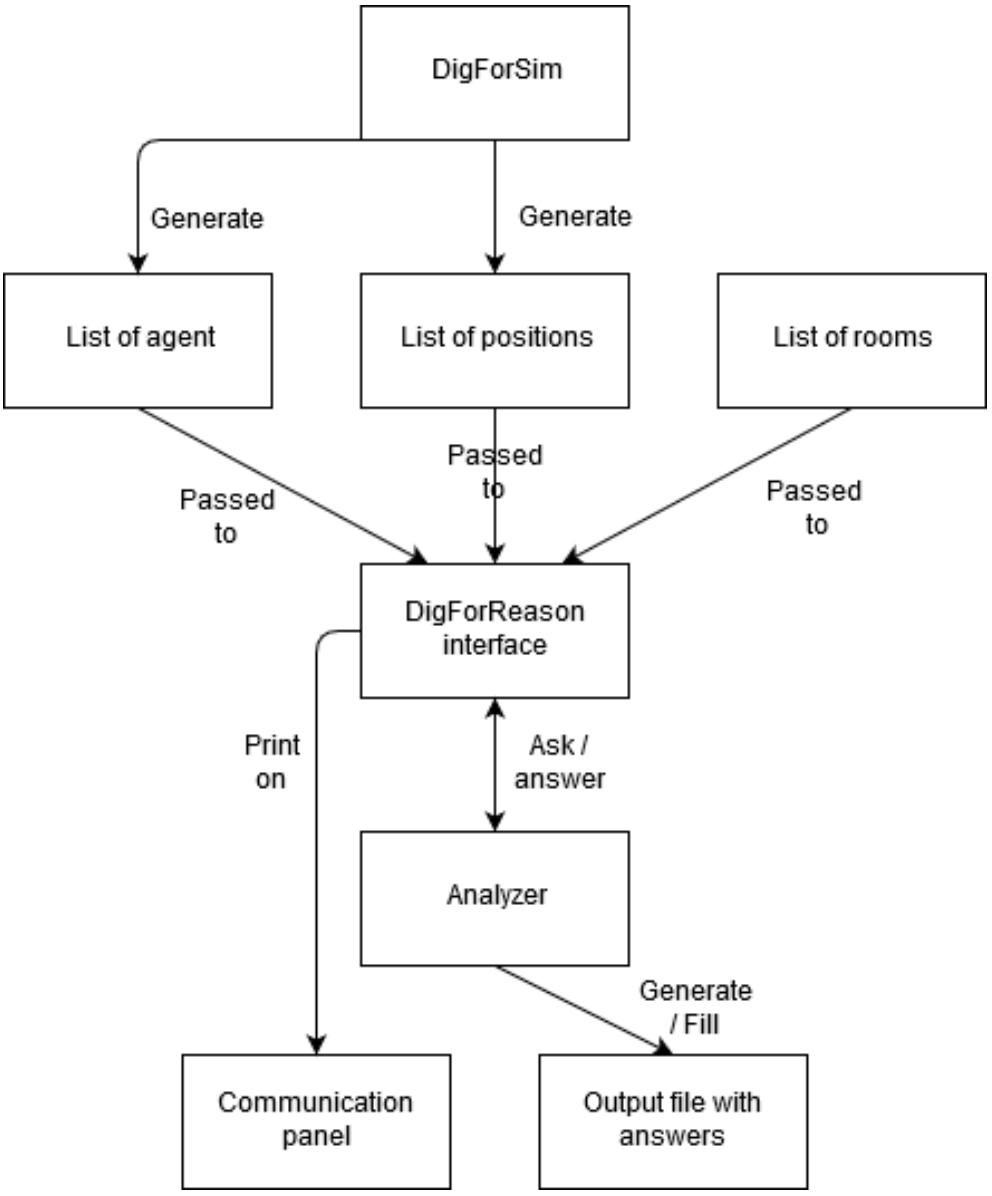

FIGURE 2.3: DigForReason diagram.

from the box when another question will be asked. The output file will be created after the choice of the input files for the current execution, and its name will be the concatenation of all input files names (*Scene1Population1Rooms1.txt* for example). Once the output file is created, if in a future execution of the application, the same input files will be chosen, the text shown in the communication box will be added to the old file, without creating it again; this allows the user to have an overview of all the questions done about certain scenarios. Also the output files will be saved in a specific directory. All the useful directory have to be present for the correct execution of the program, so they have not to be deleted. For what concern the aesthetic part, the interface will be very simple, and as more intuitive

as possible.

Chapter 3

Selecting the Tool for DigForSim Development

3.1 Tools comparison

3.1.1 Netlogo Vs Relogo

This project aims to represent movements of people (represented by agents) on crime scenes, creating data related to locations of people and possible interactions. Considering that agents, in this simulation, do not need to reason it is possible to avoid the usage of a BDI architecture so we can exclude from the pool of tools all those that are based on this type of architecture, as Jadex, Jade and Jason. We also want to use an up to date tool that is why we can surely exclude Zeus whose last version dates back to 2006 (Zeus toolkit 2.0) and also MASON that has a quite scarce documentation. Then there is AnyLogic which last version is recent (2018), but seems to require a quite powerful computer to work well. Tools that remain available are: Repast, Netlogo and Unity. Netlogo is based on the quite intuitive Logo language, very well documented and, also if its last version is not properly new it seems to be a good option because it is easy to use, also working through the browser application, and very suitable for our needs. Relogo is based on Java but, in addition to java features, allows to use also all the Logo language primitives. Unity is based on C sharp that has a good documentation and is similar to java. All these remaining tools seems to be good for our purpose but there is the needs to work just with one of them. Netlogo and Relogo seems

to be similar in their features, except for the language they use; since the fact that these two tools seems to be more or less the same thing based on different languages, both with their pro and cons, in the next section we will compare them more in detail and the best one will be compared with Unity that is surely the one that provides better features for the 3D visualization of the simulation, but it may not be the best one for managing agents.

Here is a comparison between Netlogo and Relogo, showing both pro and cons of each tool: [75]:

- **Netlogo**
  - **Pros:**
    - Intuitive also for novice programmer
    - Good documentation
    - Created for agent programming
    - Good GUI that includes the possibilities to directly manage variables
    - Easy to use since the fact that allow to work from the browser
    - Good execution time
  - **Cons:**
    - lack of a step by step debugger
    - It is harder to make agents do complex action because of the structure of the language
- **ReLogo**
  - **Pros:**
    - Based on java (exist also a python version), it takes advantages of Object oriented structure
    - Complex action are easier to implement
    - Good GUI based on the one of Netlogo
    - Primitives and structures for Agent programming based on the ones of Netlogo

    - Possibility to also use Repast Simphony's larger library and wider variety of modeling capabilities

- **Cons:**
    - It needs to be used with Eclipse IDE that is a powerful but complex tool to learn if used with additional libraries
    - Scarce documentation
    - It requires a pre-existent knowledge of Groovy Language and Java

The main advantage of Repast is that in addition to all the logo primitives, it supports java features, while the main problems are that documentation is scarce and is of difficult usage since the fact that requires to download and link lots of libraries to eclipse; these disadvantages led us to choose to discard Relogo.
Netlogo is widely used for creating multi-agent simulation (mainly in 2D), and provides the possibility to import shapefiles, created in order to represent some real environments, to be used for simulations. There are several ways to create a shapefile, and one of them is using ArcGis on maps exported from google earth; these have to be exported as KML (keyhole markup language) or KMZ (Keyhole markup language zipped) files and then converted in shapefiles. Another possibility is using QGIS, this one is also a free tool, so it could have been a valuable option but unfortunately it has a scarce documentation. Working on these kind of files it is possible to notice that google earth works as a model where we have to draw lines (representing streets) that becomes our shapefiles and use markers for defining buildings. As we can see this is not an optimal solution for creating some indoor environment. Once that the .kml file has been created it can be converted in a shapefile thanks to QGIS. There is the possibility to find also some kml files already created and there is also an online converter (https://mygeodata.cloud/converter/kml-to-shp) that automatically creates shapefiles from kml files. The shapefile represents a popular geospatial vector data format for GIS. It is developed and regulated by ESRI (Environmental Systems Research Institute) as an open specification for data interoperability among ESRI and other GIS software products [76]. The shapefile format can spatially describe vector features: points, lines, and polygons, representing, for example, water wells, rivers, and lakes. Each item usually has attributes that describe it, such as name or temperature. All this data type can be used in Netlogo thanks to the GIS extension. On the other side there is Unity, that has evolved from its first version and nowadays it provides also some internal and external libraries for creating multi-agent simulation. The environment in

Unity has to be created directly from the application or with 3D modeling tools; both these approach needs a designer working on it.

In conclusion, considering that environments can be imported also from google maps, without the need for a strong design phase, Netlogo seems to be the best option, also if it does not provide a good 3D visualization. Anyway, in case of a failure with Netlogo, we want to give Unity a chance because of the simple library that allows the usage of agents, giving to the user the possibility to create good applications without the need of a complex phase of coding, allowing the programmer to focus its attention on details and functionalities of the simulation.

3.1.2 Tools testing: Netlogo

We have studied how to use the GIS extension in the best way: there is to define a transformation between GIS data space and NetLogo space, then to load datasets and perform various operations on them. The easiest way to define this transformation is to take the union of the "envelopes" or bounding rectangles of all of datasets in GIS space and map directly to the bounds of the NetLogo world. A projection may optionally be defined for the GIS space, in which case datasets are reprojected to match that projection as they are loaded (Thanks to the *.prj* file that is always present in this kind of data and that describes the projection or geographic coordinate system of the data). If no associated *.prj* file is found, it will be used the default projection. Once the coordinate system is defined, we can load datasets of both types :

- **Vector Dataset -** consists of a collection of vector features, each one of which is a point, line, or polygon, along with a set of property values. A single vector dataset may contain only one of the three possible types of features. There are several things that can be done with a vector dataset: ask it for the names of the properties of its features, ask it for its "envelope" (bounding rectangle), ask for a list of all Vector Features in the dataset, ask it for its centroid (center of gravity), ask for a subset of a given agentset whose agents intersect the given vector feature, search for a single Vector Feature or list of Vector Features whose values for a particular property are less than or greater than a particular value, lie within a given range or

match a given string using wildcard matching ("*", which means any number of occurrences of any characters). If the vector features are polygons, It is also possible to apply the values of a particular property of the datasets' features to a given patch variable. For point data, each vertex list is a one-element list, for line data, each vertex list represents the vertices of a line that makes up that feature and for polygon data, each vertex list represents one "ring" of the polygon, and a single vertex plays the role of the first and last vertex of the list. Vertex lists are made up of values of type Vertex, and the centroid has type Vertex as well.

- **Raster dataset -** Represent the world as a surface divided into a regular grid of cells. Raster models are useful for storing data that varies continuously, as in an aerial photograph, or an elevation surface. Also with Raster Data is possible to do lots of the things already mentioned in Vector Datasets, but Raster Data functionalities mostly involve sampling the values in the dataset, or re-sampling a raster to a different resolution. It is also possible to apply a raster to a given patch variable, and convolve it using an arbitrary convolution matrix.

After having analyzed both dataset types the vector one seems to be the more suitable option for our purpose. First implementation aimed to try Netlogo and the GIS extension, so it is a sort of testing of their functionalities in order to make it easier to work with them on the simulation that we want to create. Initially we have imported a shapefile representing Italy as a grid; we downloaded three different files, each one with a better level of resolution for the grid (the higher the resolution the more accurate was the shape of Italy) and we have chosen to use the less accurate one that had bigger squares in order to better see what we were doing on the shapefile.

The second thing that we have tried was to move a single agent just inside the shapefile, avoiding the possibility of moving in the "empty space"; thanks to the function " **gis:intersects?** " we could take over the intersection of the agent with the shapefile, and modifying the turtle logic we managed to make it move just in the designed zone. The problem in this first experimentation is that before understanding not to be in the shapefile zone anymore the turtle has to get out. During tests, we found an example that made turtles move just on streets (lines drawing the shapefile) but this example uses another shapefile made of dots to

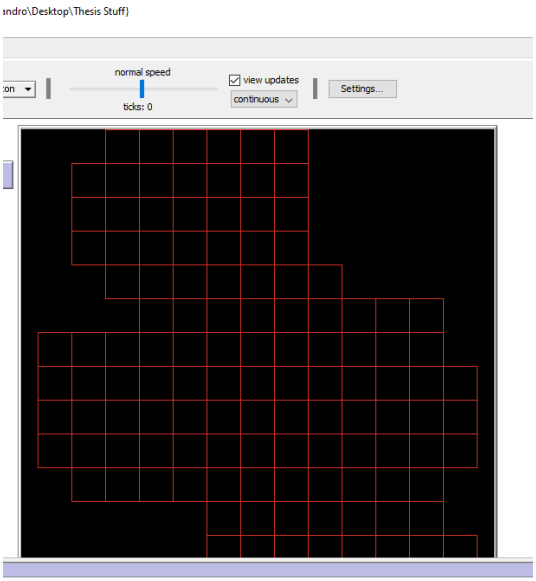

FIGURE 3.1: Very low resolution shape of Italy in Netlogo.

create turtles on the right positions (File that we haven't found). During trials with the Netlogo "GIS" extension we have tried to understand how to get the position of turtles on the shapefile and using the "**gis:contains"/"gis:intersect**" functions and some others for managing turtles we have found the right way to do it. For highlighting results agents (represented by Netlogo turtles) filled different grid's squares with their color using the **gis:fill** function.

We managed to move turtles just inside the shapefile controlling ahead patches of each turtle and allowing them to move just if those patches were in the shapefile using the **gis:intersect?** function, that check if a certain patch is part of the shapefile. We have also found a way to increase dimensions of the world (resize world function) that allow the user to better understand positions of turtles. Since the fact that initially, the idea was to create a simulation that could also fit an event happened in a quite big area of a city, we had the need to make turtles move not inside the shapefile, but on lines in order to simulate the movement on streets , and this is what we have tried next. After lots of trials, we managed to make agents move just on lines, and we've downloaded from the internet two different files representing waterways of a country and streets of another one. Surfing on the internet it is possible to find sites that provide downloadable shapefiles of all countries. Moving turtles just on streets has been the hardest task because it is difficult to manage the movement on boundaries of the shapefile, mainly the ones that have to move randomly. Firstly we have found a way to create agents at a certain longitude and latitude of the current shapefile's line (this took a long time

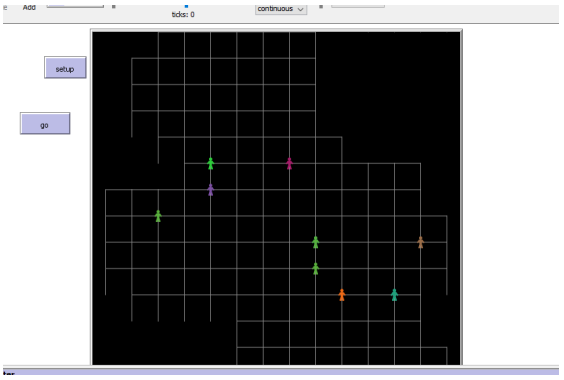

FIGURE 3.2: Creating agents on the map.

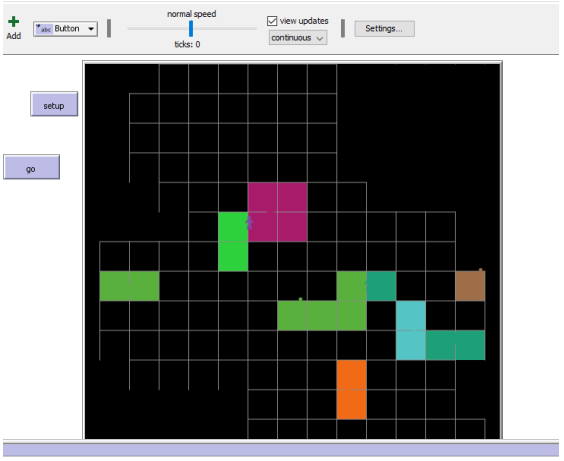

FIGURE 3.3: Moving agents on the map.

for initializing); then setting his direction to the next point of that line (this is possible since the fact that the shapefile is created by linking dots with a certain longitude and latitude); we made turtles able to move forward or backward until the end of their line. The main problem of this implementation is that each agent should keep in memory the list of all points of the line on which it is moving in order to be able to step on the next/previous point (depending on a random variable) of the line, and this is not optimal since the fact that agents should be able to move on all lines and to swap from one to another at crossroads. Starting the application we have noticed that when the test function was launched turtles seemed to move once at a time, so they were not moving concurrently and this is probably caused by the fact that in Netlogo concurrency is simulated. At the end of the first test phase, turtles were able to move back and forth on the shapefile lines where they have been created.

At this point running the test application at a low tick rate (or at an high tick rate

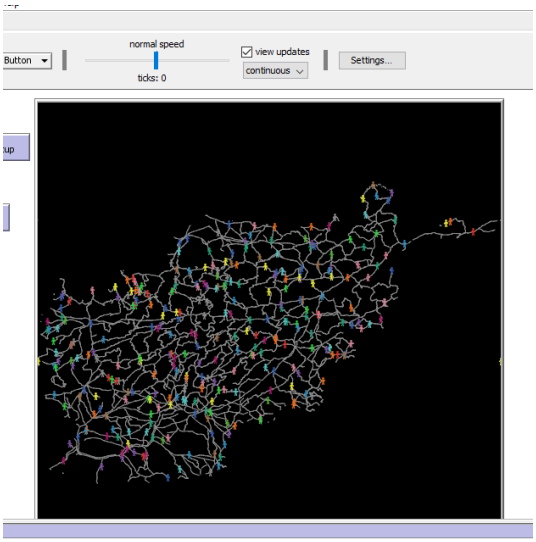

FIGURE 3.4: Spawning agents on lines of a big shapefile.

but with lots of agents) we have noticed that turtles do not move exactly straight on the shapefile's lines, but they moved on patches that cross the line's path, so it seemed like they were moving on diagonals intersecting lines. Probably this is because of the move function that allows turtles to change the patch where they are stationed. The way to make turtles move from one line's point to another is setting their coordinates at each step and this is possible since the fact that each agent has an in advance knowledge of the line where it moves. Next step is to enable turtles to move on every line of the shapefile, independently from where they have been created but here comes the real problem: after lots of trials working just with shapefiles we found a way to convert the shapefile in a more simple structure, Netlogo links (links between turtles), that can be used as the shapefile. The idea is simple: creating invisible turtles called dots for each point of the shapefile and then linking them with Netlogo links recreating the shapefile(that is not rendered anymore); now movements of turtles should be easier and less expansive since the fact that there is no need to save a path in turtles memory and that there are more functions for managing what is around a certain turtle. The main problem is that converting a shapefile we have noticed that there were no intersection points, and each line started and finished before intersecting the others; anyway we have found a partial solution to avoid this problem making turtles move on a point that is in a certain radius; this solution leads to some other problems, as the dimension of the radius: if the radius is too large turtles jumped on nearby lines and if the radius is too small the previous problem is not solved. Links gave anyway the possibility to create lines whit a different thickness and I could have managed them in order

to give to some turtles the possibility to go on some lines and to other turtles the possibilities to move on other lines(an example could be pedestrian and cars that moves on different zones). After all the trials done, moving turtles on lines as we wanted seemed not to be feasible, and in addition this method resulted not good for representing environments like houses and dispersive for representing outdoor environments; that is why we have decided to treat just smaller cases and to represent the inside of a building or a limited outdoor zone, using as starting point a sort of crowd simulation. At this point, unity could be a good tool, since the fact that nowadays it started to be used for simulating crowded environments and because it provides a good 3D visualization.

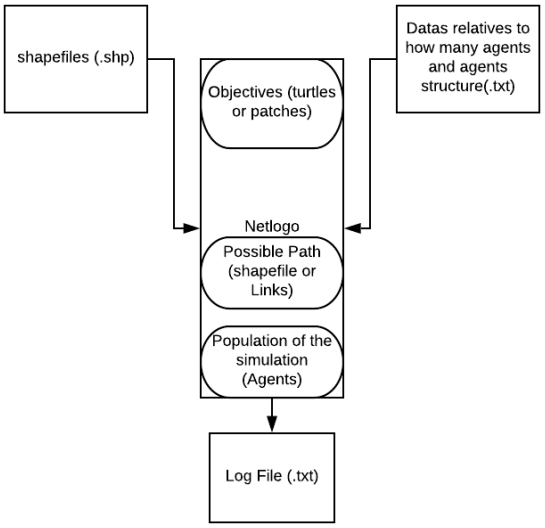

FIGURE 3.5: Diagram for a possible Netlogo version.

3.1.3 Tool testing: Unity

Unity is a resource-intensive tool so having a high number of entities on the scene could have the side effect of slowing down the execution (if the computer is not powerful enough); this is why is probably better to decrease the dimension of the scene and the number of entities for this specific application, in order to have a more fluid execution. Some tests helped us in understanding if Unity should effectively be useful for our purpose. Initially, we tried to create an agent moving

in the environment aiming for its objective, and this was quite simple using the **NavMesh** library and following these steps:

1. **Creation of the environment:** modeling the environment and populating it with some objects, then the navigation option has to be set to static, this could be done directly from Unity. Considering that Unity provides a limited amount of predefined objects (mainly geometric figures or solids), there could be the necessity to import some complex ones from downloadable assets or to create them with tools as Sketchup or Blender could be.

2. **Creation of agents:** we have used the capsule object that is of common usage to represent a person when there is a lack of 3D models, but this could be easily modified in a future moment.

3. **Creation of navigation meshes (zones where agents can move):** this is possible thanks to a library that provides components to attach to agents and scene objects; it gives also the possibility to define some agents' and objects' properties that can be used by Unity for creating the walkable zone for agents.

4. **Simulation start:** Once that everything is set up, agents move towards their targets following the shortest path.

Once that all these things are done we have a simple application where the agents' behavior is to move toward their objectives following the shortest path. We have tried to create also a zone with a little climb for reaching the target and it works perfectly. The next step has been to create some agents moving randomly in the environment and to make some of the previous agents able to change target after reaching the first one; this work with Unity has been easier with than with Netlogo and the code is composed of few lines. The main advantage is that Unity provides most of the things that we had to implement by ourselves in Netlogo. The second part of tests with Unity and agents has been to create some entities configuring them by text file, and since the fact that Unity relies on C sharp, this has not been difficult. Initially, the file allowed to define just two properties for managing the random movement of some agents then tests moved to the creation of a group of agents similar to each other and those agents had to move randomly in the environment as they were a crowd. Next trials wanted to understand how to create more type of agents, each one moving on a different zones, to understand

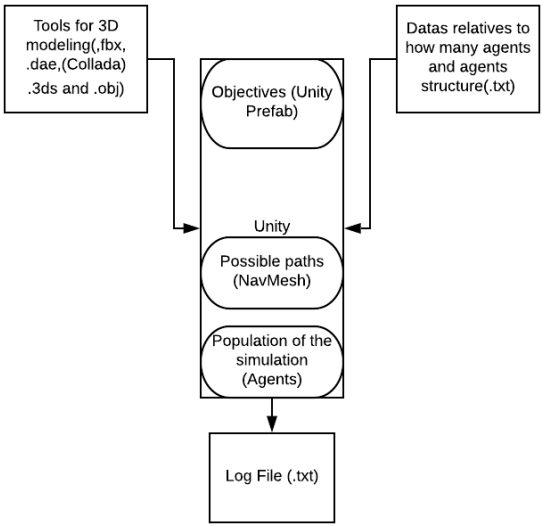

FIGURE 3.6: Diagram for a possible Unity version.

if it is possible to have a simulation that includes different types of entities (for example cars and humans if the simulation needs to represent a street); in these case it has been discovered that Unity does not support more than one agent type at once, but that there is an external library that allows this functionality. With this one, it is possible to create different empty objects that can be filled with some components of this library, allowing each one to manage a different zone where agents can move. Thanks to this library we managed to insert in the testing program also a street where just Agent of type vehicle can move. A nice feature of this library is that they can be modified at run time so an agent that opens a door or that moves an obstacle creates a new viable path that will be taken into account also from other agents when they will compute future paths for their movements. The only problem is that agents computes their path just once when their objective is modified, so if they are already moving towards an object they will take into account the new viable path just for their next objective. Since the fact that this application is developed in Unity, this section is more an overview with respect to the previous one. All the details of the Unity implementation will be described in the 4 chapter.

Chapter 4

DigForSim Implementation and Usage

4.1 DigForSim Implementation

4.1.1 Intro to Unity elements

Before talking about the implementation itself it is necessary to introduce briefly the Unity engine. As already said in Section 1.5 Unity is a game engine, in fact, it is mainly used for creating videogames. The editor provided by Unity gives the possibility to create 3D/2D environments inserting different type of basic elements as planes, cubes, cylinders, capsules, and other simple geometric objects, but it also gives the possibility to create some simple Graphical user interfaces inserting dropdown menus, text boxes, and other GUI elements. All objects inserted in a scene can be personalized through the inspector, where all the properties (size, position, rotation, material, text color, text size) can be modified. When managing a terrain, the inspector shows some of the general properties, but allows also to modify some others to allow the creation of different type of terrains (mountains, plains, hills and so on); this can become useful also for simulations enabling the user to create the desired environment; modifying terrain's height or size and enriching it with trees, rocks and other elements. There is also the possibility to download some assets that could be textures, scripts that do something useful (as script for the player, script for the camera and so on) or models. From the asset store is possible to download also packages that defines objects ready for

being used in a project (an example could be a car with texture, script and sounds already attached to it).

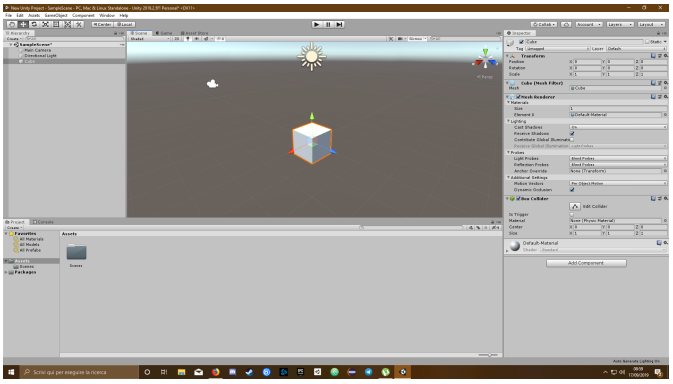

FIGURE 4.1: Object Inspector.

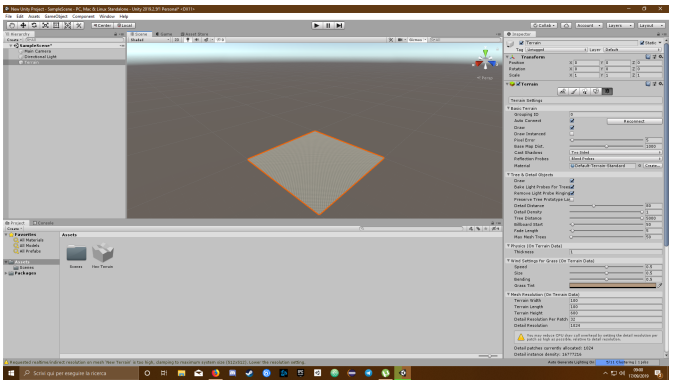

FIGURE 4.2: Terrain Inspector.

Unity provides the possibility to create prefabs (prefabricated objects), that are pre-built objects used for instantiating at runtime different entities with the same behavior. Prefabs can be useful in our case for instantiating different agents at runtime (an attached script defines the behavior of agents). For this application, we have created two agent **prefabs**: one for each wandering agent instance and one for each special agent instance. Prefabs could be used also for entities that have the same behavioral properties (given by the script) but different values set at runtime, exactly as happens for special agents (they have different speed and different targets but they behave in the same way). A prefab can be created by modeling the object in the scene and then dragging it in one of the project directories, usually in the *Prefab* one. Each object in the scene can have different child objects and each one of them can have a script attached that gives it a certain behavior.

As we can see in figure 4.3 a prefab can be an object, with some other child objects that put together create a 3D model that can be useful; the folder Prefab,

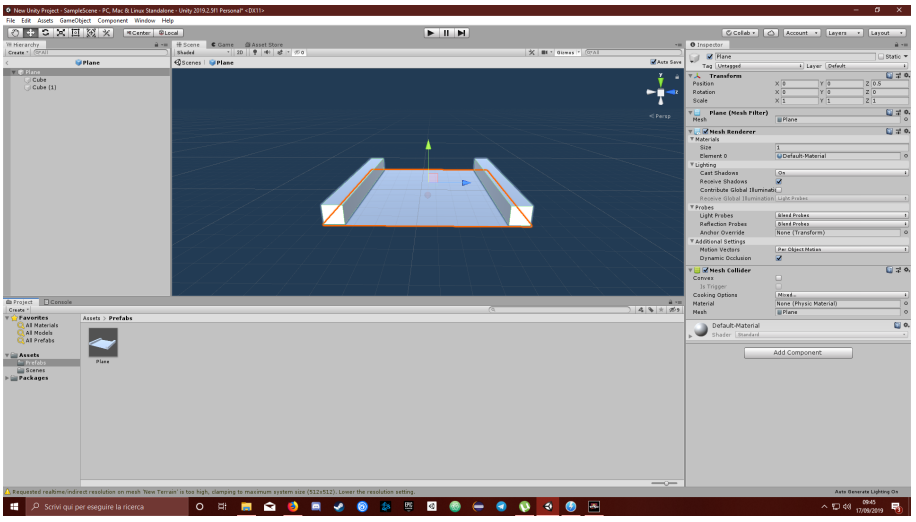

FIGURE 4.3: Prefab Example.

present in the image is not a standard folder, it needs to be created manually, and prefabs need to be dragged into it (this is just for having order in the project structure). In the inspector are shown all the components attached to the selected object, each one managing different properties; some of them are:

- **Transform:** This component determines the Position, Rotation, and Scale of each object in the scene. Every GameObject has a Transform. The number of coordinates depends on the number of dimensions used for the project that is being developed.

- **RigidBody:** This enables an object to act under the control of physics. The Rigidbody can receive forces and torque to make the objects move realistically. Any object must contain a Rigidbody to be influenced by gravity, act under added forces via scripting, or interact with other objects.

- **Collider:** This component manage collision with other objects and allow scripts to detect them thanks to two standard functions: *OnCollisionEnter() and OnTriggerEnter()*. The *OnCollisionEnter()* is used to manage collisions between objects so that when an object collides with another they get repelled by their forces. To use *OnCollisionEnter()* a collider must be attached to the object and the*IsTrigger* property should be unchecked and to receive a physical collision event both objects must have a collider. The *OnTriggerEnter()* is used to detect collisions and it needs the *IsTrigger* property of the collider checked. There is a specific collider type for cubes, spheres, capsules, and other objects but if this specific collider does not exist the object uses

the mesh collider; this last one is a bit more complex than the others, in fact, it has more properties and the user can define some shapes for it, as a plane, a cylinder and so on.

- **Mesh:** 3D Meshes are the main graphics primitive of Unity. Various components exist in Unity to render regular or skinned meshes, trails or 3D lines.

- **Script:** This component is the code attached to the object and public variables are shown on the object inspector, giving both the possibility to analyze them in a better way during the execution of the program and linking them some other objects. An example could be an agent that has to follow another agent; the followed agent is linked to the variable target of the first agent (this could be done also by script). Usually, all scripts contain two main functions: **Start() & Update()**, the first one is called before the first frame of the application, while the latter is called at each frame; both functions might be not defined. There is also another function used for the initialization, the **OnEnable()** that is called before the **Start()** function and is used when an object needs to be initialized before starting its execution.

- **Nav Mesh Agent:** This component defines the type of the selected agent, in fact, there is the possibility to create different agent's type by the **Navigation** tab next to the inspector one. If the **Navigation** tab is not visible it can be opened going to the *window* dropdown menu and then going in the *AI* section. Each type of agent has different properties: height, radius, speed, step height, and max slope. In this component are present some values that can be set by the user as *stoppingDistance*, *remainingDistance*, *obstacle avoidance*, all the properties present in the navigation tab and some others. Figure 4.4 shows the navigation tab

Unity allows the user to import external 3D models whether that they are downloaded from official assets or that are imported from external tools, as Blender could be. The programming languages that could be used for programming with Unity are C#, UnityScript (JavaScript's stricter version) and Boo Script (Almost Python); we use C#. Multiple scripts can be attached to an object and each one of them defines a behavior and can be activated or deactivated whether directly from the editor or by the code. What makes Unity a convenient tool for this project is also the possibility to use some public variables allowing the user to

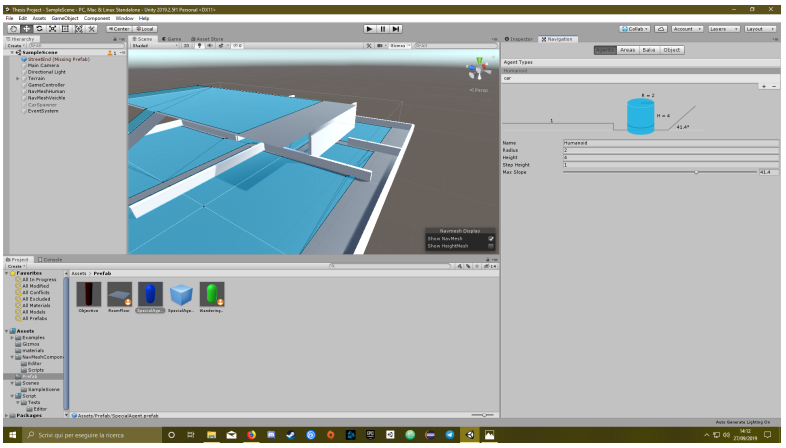

FIGURE 4.4: The navigation tab.

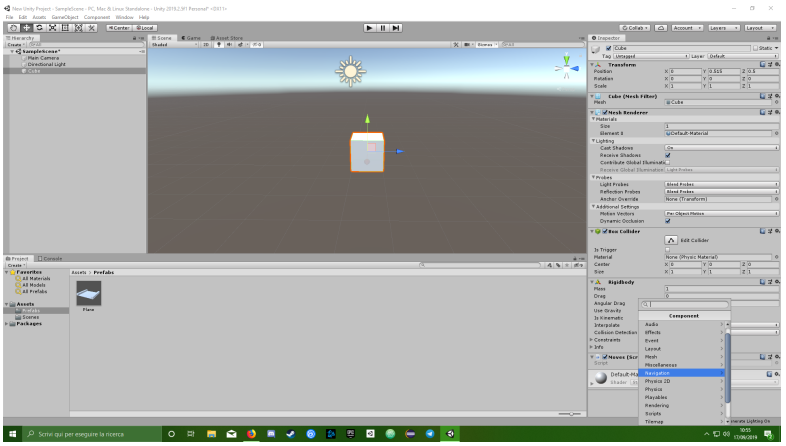

FIGURE 4.5: Some of the possible components of an object.

modify them at runtime directly from the editor (in this case they do not maintain changes), helping the debugging phase. There is also the possibility to activate and deactivate the object in the scene at runtime; hese two functionalities can help in some tests. At each object is assigned a layer, and for some operations, some layers can be discarded together with the objects they are assigned to. There is a certain number of default layers, but those can be increased by creating others and naming them as the user wants. For managing agents, the NavMesh library has been recently made available in Unity; this allows to attach the NavMesh agent component to objects; this component allows the user to set up some properties for the agent as speed, type of the agent, stopping distance and all the properties that an agent needs for knowing where and how to move. Obviously, the navigation mesh requires certain settings in order to allow a certain type of agent to move on it taking into account also agent properties (as the radius of that type of agent, max slope, step height, height and so on). Agents have a target, that

can be changed at runtime, and they compute the shortest path to reach it when the target is updated. Forcing this computation at each frame of the execution it is possible to set as agent's target a moving object (as another agent could be), simulating a pursuit. A nice feature of navigation meshes is that is possible to create more of them, giving each one a different cost. So an agent that has to compute the shortest path for reaching its target, considers the cost of each step on a certain navigation mesh choosing the cheapest path. Since the fact that the standard NavMesh library allows the user to create navigation meshes just for one type of agent, it has been necessary to use an external library called **NavMesh Components** that extend the standard one, allowing to create multiple NavMesh surfaces for different types of agents. Unity gives the possibility to create applications divided into scenes and we have one containing the selection menu and one containing the simulation. These scenes have to be loaded once at a time, in order not to overlap their content. There are some functions for loading/unloading scenes as the *LoadScene & UnloadScene()*. It is important to know that loading another scene destroys all the objects of the previous one, but if some of them are necessary for the new scene they can be maintained calling on them the function *DontDestroyOnLoad(ElementToKeep)*.

4.1.2 Structure of configuration file

A configuration file, called *ConfigSimulation.txt*, defines simulation settings, and a guideline for writing it is the README.txt file in the project directory. This file presents also the pull of possible colors for the agents. Here is an example of configuration File:

As we can see from image 4.6 it is divided in four sections:

1. **Selecting the time interval (seconds) between each print of the agent's position:** this parameter should be a number that is casted to float to represent time.

2. **Definition of objects to insert in the scene and their location:** this is a list "object1 location1 object2 location2" where location is where the user wants to place the object. This section allows the user to insert in the scene objects that have not been inserted during the design phase.

3. **Definition of all the quantities and properties of wandering agents:** there is to define the total number of wandering agents and the percentage of each type of them. All the properties of Wandering agents are numbers: the percentage, the speed, the radius of the zone in which agents move randomly for the defined time (the wandering radius), and the time they spend in the zone with the previously mentioned radius before changing it (the wandering time)

4. **Definition of special agents and their properties:** For each agent there is to define its the name, its spawn location (used for taking coordinates), its color (passed as a string that used as the identifier for a colors dictionary),its speed and a list of targets and time to spend next to each target (target1 time1 target2 time2).

Let's consider this possible definition of a special agent: **Bob Corridor Green 1.5 Phone 1 Charlie 2 Kitchen 3**. Bob's path highlights the main features of DigForSim: agents can reach rooms (the kitchen), objects positioned in rooms (the phone), but also agents whose position, at the time of reaching them, depends on how simulation evolved. The agents' paths should be driven by what the user, who might be a detective or a digital forensics expert in charge for supporting

```
Log Time interval
1

Objective Creation:
KitchenTable Kitchen

Agents Quantity
AgentsInSimulation: 60
FastWanderingAgents: 15 30 1 20
MediumWanderingAgents: 40 20 1 10
SlowWanderingAgents: 45 15 1 5

Special Agent
Giulio Kitchen red 15 Sofa 10
Jack LivingRoom yellow 10 Fridge 3 Table 15 Bed 10
Rosa Bathroom green 8 Bed 4 Television 1 Sofa 10
Damiano Kitchen blue 15 Corridor 1 LivingRoom 1 Table 5 WC 10
Alex Bed grey 16 Corridor 1 Shelf 5 Table 10
Carl Fridge black 20 BathroomSink 7 Shelf 6 Bedroom 5
Kraig Corridor white 7 Shelf 2 Fridge 3 KitchenTable 10
Fabio Bed green 14 WC 2 Shelf 1 Bed 1
Gianni DiningRoomTable magenta 17 WC 1 Sofa 2 DiningRoomTable
3
Franco LivingRoom red 12 DiningRoom 1 Bedroom 1 Kitchen 1
Fridge 1
Talita DiningRoom green 7 Fridge 1 Bed 1
David WC blue 10 Kitchen 1 DiningRoomTable 2 Sofa 3
Laura BathroomSink yellow 2 Bed 1 Fridge 1
Marco Bed red 10 Kitchen 1 DiningRoom 1 LivingRoom 1 Sofa 1
Alexander Bedroom magenta 15 Bathroom 1 WC 1
```

FIGURE 4.6: An example of configuration file.

a magistrate in his investigation, knows to be true or plausible about the crime scene. The way agents move from one objective to another is up to DigForSim. They might need to cross other rooms, to avoid collisions with wandering and special agents, to avoid bumping into obstacles. To enforce an agent to follow a very precise and constrained path, many objects (also "fake" ones) should be positioned in the environment, and be enclosed in the agent's path. To model the agent changing pace, longer or shorter stops should be associated with the targets in its path. A criminal agent following its victim can just have the victim as its objective; a policeman can have the criminal as its objective. Finally, the time lapse of the event emission and logging can be decided by the user: the longer, the fewer information the online and offline reasoning tools will count on in order to answer the user queries. Despite its simplicity, the configuration file allows the user to have a good control of what agents should do, giving them more or less freedom, depending on the available knowledge and on the hypotheses to be explored.

4.1.3 Creation of the simulation environment

The environment in which simulating the scene of the crime, as already said, could be created whether with other applications, as blender or sketchup, or directly in Unity. In the first case is better to create files of these types : *.fbx, .dae(Collada), .rds or .obj*, as shown in Figure 3.6. To import files into a project there is to select the assets window on top of the editor, choosing the option import asset, selecting the right 3D model file that have to be saved as an asset (it could be moved in another directory if the user wants), and then drag it into the wanted location. For what concerns the creation of the model with other applications, the only suggestion is to pay attention to objects'names to know which data to pass to the configuration file. If the environment is created directly in Unity, a suggestion is to use cubes for walls and planes for grounds, naming them correctly and checking all the objects' colliders. In both cases, it is important to fill the environment in Unity, with the provided cylinder prefab and to create a terrain that underlies the created environment. It is necessary for the computation of the agents' initial position to have an underlying terrain object bigger than the environment underlying it. This terrain has to be linked to the **GameController** object that is described in subsection 4.1.7. Since the fact that environments for DigForSim are minimal, a good idea would be to create them in Unity, but it is not

strictly necessary. Since the fact that agents have to walk just on the planes that represent the ground, these have a different layer than the terrain object placed under them, so the terrain is used just for taking into account some information about the dimension of the environment, while to compute the zone where agents can walk are considered just the planes. Zones, where agents can walk, are defined by the NavMesh (Navigation Meshes) and to each NavMesh can be passed the list of layers to keep into account, and to the NavMesh used for agents are passed all the layer except the one of the terrain object.

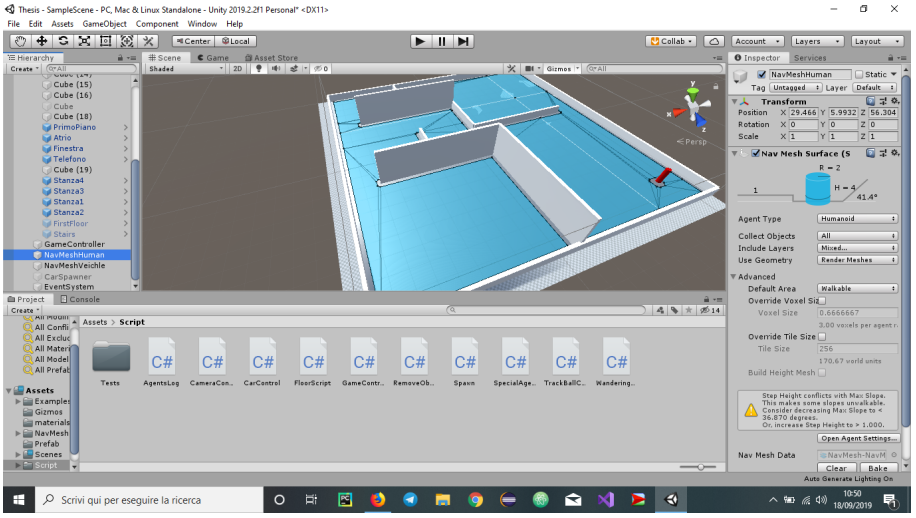

FIGURE 4.7: Example of environment and NavMesh.

Image 4.7 shows the map used for one of our tests. The light blue element, selected in the objects' list, is the Navigation Mesh for human-like agents. This NavMehs is computed, thinking to properties of that type of agent (height, size and so on). Looking at the image, it is possible to see some zones that are not included in the navigation mesh because unreachable for human agents. This computation is done automatically by Unity, there is just to define human agents' properties in the inspector (The light blue cylinder on the right shows all the properties of an agent type). Another important things for the creation of the environment is to ensure that each plane that is part of the terrain, and that can be thought as a room, or a zone where agents can walk, must have a mesh collider attached to it enabling collisions with agents; this collisions will provide information on the rooms/zones in which agents are walking, giving them the possibility to print rooms/zones names in the output file. This is why also creating the environment with external tools, each zone should be made with different planes, to make them recognizable.

4.1.4 Simulation's menu

This is a separate scene that is loaded before launching the simulation itself. This scene shows three simple elements, a label asking to choose the configuration files the user wants to pass to the GameController (see Section 4.1.7) so that it can set up the simulation; this file can be selected thanks to a dropdown menu element that contains all the files in the *ConfigurationFiles* directory and thanks to a button that loads the scene containing the simulation (populated following the chosen configuration file). For letting the button work we had to link to it the function that loads the next scene through the Unity inspector, where a little table gives the possibility to add all the functions to be called when the button is pressed, both if used from the editor and the executable file, or just in one of the two cases. This table is shown in the bottom-right corner of Figure 4.8 while Figure 4.9 shows the menu itself.

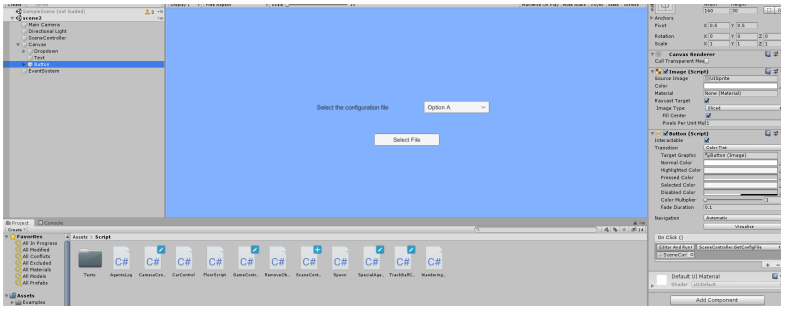

FIGURE 4.8: OnClick() element in button inspector.

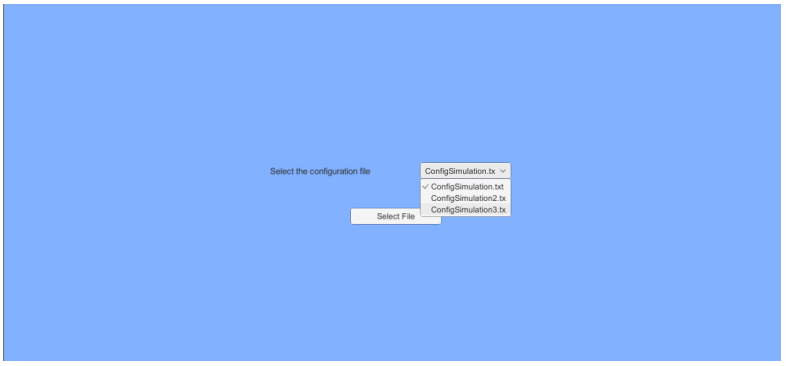

FIGURE 4.9: Here is the selection menu.

4.1.5 Simulation's prefab

Considering that the environment has to be created by the user of the application, or in case by a designer that works with the user, the aim of this application is

not to provide the single simulation, but to provide scripted prefabs that can be used in each environment created following the few suggestions given in Section 4.1.3 and an automated way to create navigation mesh on each used environment. Here is a list of all the provided prefabs:

- Wandering Agent: this is the entity that moves randomly in the environment. It is a green capsule with a *capsule Collider* and a *NavMesh Agent* component that makes the agent able to move on the Navigation Mesh. Agent's movements are defined by the *WanderingAgents.cs* script that is attached to each wandering agent, and consists in getting a point inside a circular zone of a certain radius moving toward it for a certain time and at a certain speed. These three wandering agent's attributes are defined in the configuration file described in subsection 4.1.2. The speed property allows to define three types of wandering agents: fast, medium speed and slow. Wandering agents are not important for the user, they are just a sort of noise in the simulation, so, to maintain the output file cleaner, they do not return pieces of information. We can define wandering agents as the passers-by. Since the fact that in this simulation physics is not necessary, and that the *Rigid body* component is not necessary for detecting a collision, both the agent prefabs do not have this component attached to avoid some computation to the computer.

- Special Agent: this is the object instantiated for each agent that has a particular behavior, and that moves following a path with different targets. Each special agent have the values of its properties defined manually in the configuration file. A special agent has its starting position, its color and its list of targets and time to spent near them, as properties defined by the configuration file. This agent has the same Unity components of a wandering agent but their *capsule collider* is positioned on a lower y coordinate, to hit the mesh collider attached to different planes, allowing the agent to get their names understanding in which room/zone it is moving. In addition scripts attached to special agents are two: *SpecialAgent.cs* and *AgentsLog.cs*. The difference between the wandering agent collider and the Special agent one is shown in Figure 4.10 and 4.11 where it is possible to see that the center of the special agent collider is not set at (0,0,0) but at (0,-0.2,0). The Wandering agent in the image is made black and not green to make the collider visible. The main problem for this agent is that it is not able to choose a

different path from the shortest one, so it needs to be forced on a longest one by giving it a list of targets to reach the real one; in this case targets can be rooms in which it has to pass for reaching its real objective. Each special agent is provided of an on top camera that allows seeing the scene from its perspective as shown in Figure 4.12. This camera can be activated by clicking on the special agent and deactivated in the same way. Before activating the camera of another agent the one in use has to be deactivated. This is not necessary for what the simulation wants to show, but could be useful for future updates that leads the simulation to show in the details what happened on the scene of the crime. For creating these agents the game controller check coordinates of their spawn location and.

- RoomFloor: This prefab is very simple but necessary. It has a mesh collider attached with the IsTrigger property checked, allowing the agents to understand in which room they are walking by triggering the collision with the mesh collider. It is necessary that the IsTrigger is enabled in this prefab and not in the special agent one because if not also collisions with other objects in the scene are detected (cylinders have mesh colliders attached).

- Objective: this is a very simple prefab with just the *transform* and *mesh collider* components; this prefab can be used to instantiate those objects needed by special agents, but not yet inserted in the environment, is represented as a red cylinder and is used by the *GameController* for instantiating objects defined in the specific section of the configuration file. It is important to maintain a low height for this prefab because it could happen that on an environment developed on more floors, as a house could be, for reaching an object an agent could try to get over it.

Roomfloor and Objective prefabs can be used just if the model is created directly in Unity; if this is not the case all the objects do not present the collider but the guide for importing a working model, presented in Section 6.2, provides all the information to overcome this problem.

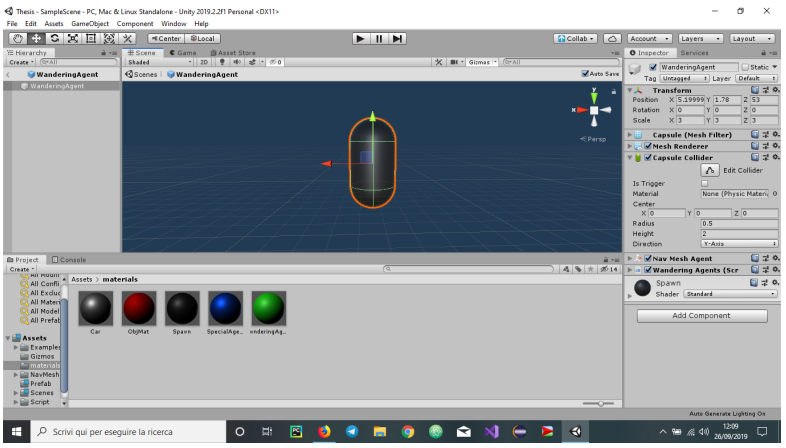

FIGURE 4.10: wandering agent collider.

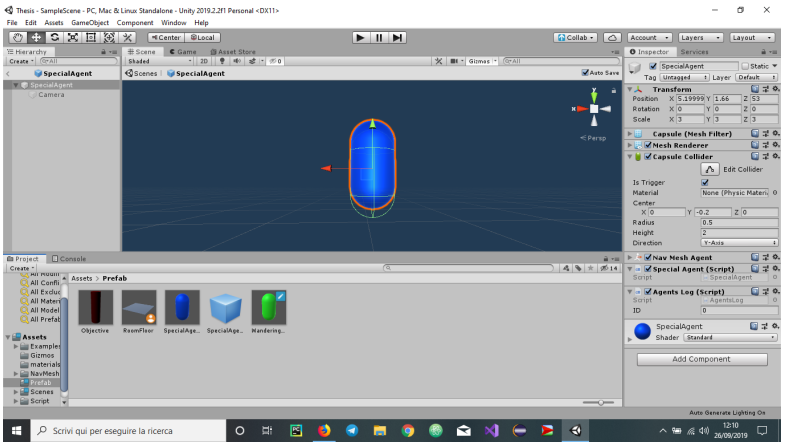

FIGURE 4.11: Special agent collider.

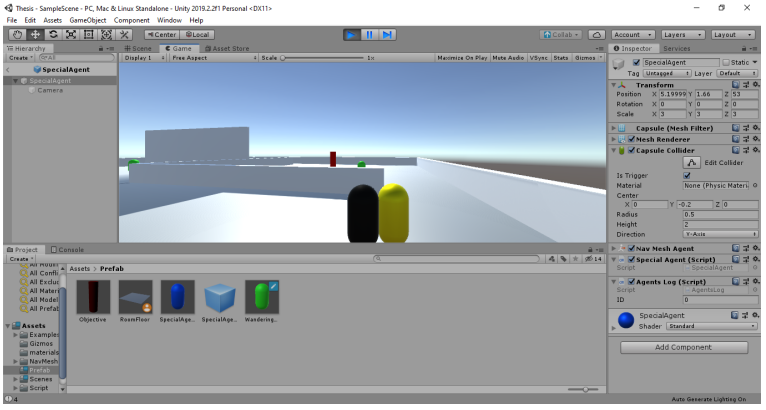

FIGURE 4.12: View from special agent camera.

4.1.6 Agents behavior

In this section, we describe how effectively agents work, talking more in detail about the code.

- **Wandering Agents:** As already said in Section 4.1.5 this type of agent has just the *WanderingAgents.cs* script attached that describes their movement. Thanks to the radius parameter passed by the configuration file each one of these agents compute a circle around himself in which he chooses a random point as its target for the time defined by another parameter. Going more in the details we can see a bit of code and in code 1 is shown the function that creates the sphere around the agent: **dist** is the radius and **origin** is the center of the sphere (that is also the position of the agent); these two parameters allow the first two lines of code to obtain a random point inside the considered sphere. The layermask is a mask specifying which NavMesh areas are allowed for finding the nearest point to the one chosen randomly inside the sphere using the **SamplePosition(...)** function. Looking at the Figure 4.13 we can see what happens at each frame during the execution: if the time passed is lower than the time parameter passed by configuration file the agent keeps moving toward the taken point on the NavMesh, while if the time is equal or greater than the given one the agent chooses a new target point, executing again the procedure done for obtaining the previous one. Since the fact that a wandering agent stops until it has to compute the new target point if it has reached its destination, it is important to choose good values for its parameters in order to simulate a constantly moving crowd.

```
void Update()
{
    timer += Time.deltaTime;
    if (timer >= wanderTimer)
    {
        Vector3 newPos = RandomNavSphere(transform.position, wanderRadius, -1);
        agent.SetDestination(newPos);
        timer = 0;
    }
}
```

FIGURE 4.13: Function called at every frame by Wandering agents.

- **Special Agents:** It has been already said that these agents have two scripts attached to them, one for movement and one for printing their log file. Differently from wandering ones they have well-defined behavior and move towards points defined by the configuration file. These agents have some parameters in common with wandering agents as *speed* and *target* but this last one for a special agent is not chosen randomly. These agents move following the shortest path straight to their destination. Looking at Figure 4.14 we can see a warp function, done also in the Start() function of wandering agents,

that is necessary for instantiating objects in the correct position on NavMesh also if the 3D model representing the environment is not placed at (0,0,0). Then the start function sets the agent camera and the current target that depends on the *obj* variable that during the start function is equal to 0 to take the first element of the target list. The *rest* parameter defines the time that the agent has to spent next to the current target and is placed in the next cell of the *objectives* array. Then we can have a look at what a special agent does during the simulation, looking at Figure 4.15, where we can observe its update function. The first thing that a special agent does at each frame is checking its distance from the objective using the *remainingDistance & stoppingDistance* value; these two values can be set by the user, but are defined by the *Nav Mesh Agent* component; there is also another check: *agent.hasPath()*, this is needed because at the beginning the remaining distance has to be computed so the first check is not immediately valid. *The hasPath()* function check if the agent has found its path or not and so if the remaining distance has been set. If the agent is close enough to its target it stops for the time defined in the *rest* variable and then it switches the target with the next one in its target list. Once that a special agent has reached its last target it has to be removed by the simulation thanks to the **Destroy()** function in the last line, to have a cleaner output file. It is important to notice that there is a check on the type of collider of the target; this because if it is a capsule collider it means that the object represents a person in the simulation, so the agent has a moving target, which position has to be computed at each frame to enable the agent to follow it. One last thing that the *SpecialAgent.cs* script do is to manage the camera attached to the agent, thanks to a predefined function called *OnClick()* that is triggered when clicking with the mouse on the agent; if this happens the main camera of the scene is deactivated and the agent's one is activated. Clicking again on the same agent the application comes back to the main camera. When an agent's camera is active it is not possible to activate the ones of the other agents without coming back to the main camera.

The other script is *AgentsLog.cs*. This one, after a certain interval of time prints the agent position in the environment; the interval of time, looking at Figure 4.6, is the first parameter passed in the configuration files. Each special agent has its ID set during the initialization by the GameController; the problem is that the ID does not give any significant information to the

```
0 riferimenti
void Start()
{
    agent.Warp(gameObject.transform.position);
    agentCamera = transform.GetChild(0);
    target = GameObject.Find(objectives[obj]);
    if(target != null)
        agent.SetDestination(target.transform.position);
    rest = float.Parse(objectives[obj + 1]);
}
```

FIGURE 4.14: Function that initialize the special agent.

```
if (agent.remainingDistance <= agent.stoppingDistance && !agent.pathPending || target == null)
{
    obj += 2;// Change next target position on the objectives array
    if (obj >= objectives.Count)// Check if going out of index with the array
        target = null;
    else
    {
        target = GameObject.Find(objectives[obj]);
        /*startTime = Time.time;
        done = false;*/
        if (target != null)
            agent.SetDestination(target.transform.position);
        rest = float.Parse(objectives[obj + 1]);
        agent.isStopped = true;//Changing the agent state
    }
}
if (target != null)
{
    if (target.GetComponent<Collider>().GetType() == typeof(CapsuleCollider))
        agent.SetDestination(target.transform.position);
}

if (agent.isStopped) // rest countdown
    rest -= Time.deltaTime;
if (rest <= 0f) // when countdown ends the agent start moving again and the rest time is set again at 3 seconds
{
    rest = 3f;
    agent.isStopped = false;
}
if (target == null && obj >= objectives.Count) // Here we check if the agent has still something to do,
                                               // and in case it has finished its work it will be destroyed.
                                               //when there are no more agents in the scene the simulation will shut down
{
    GameController.inst.totAgentsaux--;
    CameraControl.inst.Enable();
    agentCamera.gameObject.SetActive(false);
    GameController.inst.following = false;
    Destroy(gameObject);

}
```

FIGURE 4.15: Function called at every frame for Special agents.

user that is reading the log file, in fact, he can not be able to understand to which agent the ID is assigned. This is why it is better to use the agent name as identifier in the log file, making also easier the use of DigForReason where the user can ask questions about a certain agent using its name. In the *AgentsLog.cs* there is a simple function that prints in a file named "**OutputN**", where N is the ID of the agent, a string with all the useful pieces of information for future analysis of the log file. An example of log file is shown in Figure 4.16.

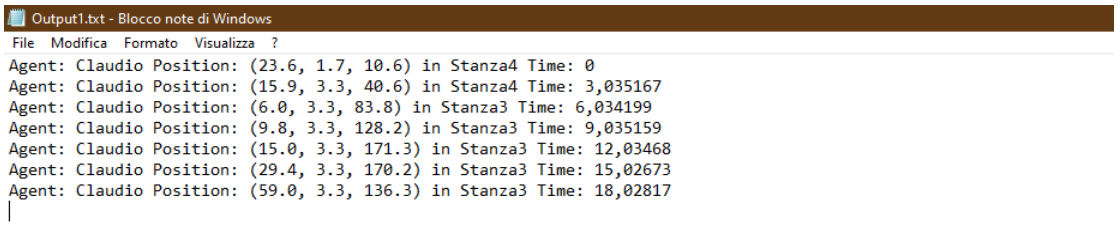

FIGURE 4.16: Example of single agent's log file.

4.1.7 GameController & SceneController

We can define the GameController as the manager of the scene containing the simulation. This is an empty object with just two components: the transform and the script; the first one is useless because the GameController can be placed in any position without influencing the simulation, while the script is a very important part because it is the one that manages everything. As the simulation start the GameController creates or (if already exist) empties the *AgentsList.txt* file (it contains the name of all agents in the simulation and is necessary to work with DigForReason), creates the navigation mesh on the environment and removes all the single agents log file merging them in a general output file for the simulation. It manages the creation of the objects passed by the configuration file and of both types of agents. For the creation of wandering agents the GameController takes their total number, and then starts to calculate how many wandering agents for each type looking to the percentages passed in the configuration file (slow, fast and medium speed). The main problem is that the conversion from percentage to the effective number could be misleading, in fact with 30 Agents and with these percentages 50%, 25% and 25% we should have 15 agents of the first type and 7,5 of second and third type. We can notice that 7,5 is not useful as information, in fact, we need whole numbers, so we have to round this result. To reach a total number that is as closest as possible to the one passed in the configuration file, there is an alternation between a round-up and a round-down method. This is not an optimal solution, but it provides good results in most of the cases, and since the fact that these agents are not truly important for the simulation it is better to focus our attention on other problems. Another thing to say is that if the sum of the 3 percentages inserted in the configuration file is different from a hundred percent the application stops, showing an error message that says: "*Check percentages, their sum is different from 100%*".

The creation of special agents is easier because all their parameters are passed by

the script and the GameController has just to set them up, keeping into account their total number to print it in the simulation's LogFile. Creation of Wandering agents and special agents is done by two different functions called by the GameController: the *CreateWanderingAgents(agent to create, agent pieces of information)* and *CreateSpecialAgent(agent to create, agent pieces of information).* The first one takes a random position using a function similar to the one used for wandering agents movement; it creates a sphere having origin in the center of the terrain underlying the environment, then takes a random point inside this sphere that is used for taking its closest point on the Navigation Mesh and create the agents. This function is called a number of times equal to the number of wandering agents of the current type (for example the slow ones). The second one is very simple and is technically a sequence of values assignments to variables. while the one used for special agents is called once for each line of the configuration file defining a special agent. Both these functions are shown in Figure 4.17 where we can see that for assigning the color to the special agents' material is used a dictionary; this is done because in Unity there is not the possibility to assign the color using a string that defines the chosen color, as happens in javascript, but there is to use some properties of the Color structure as *Color.red, Color.magenta, Color.green* and so on. Since the fact that it was not possible to understand the agent color selecting it by script, we have created a structure that allowed us to do it. This simple structure is shown in Figure 4.18. There are few available colors for agents and they are listed in the README.TXT file.

We have already named a simulation log file that is different from a single agent's log file, in fact, it contains movements of all th agents, we can define it as a merging of all the single agent's log files. This simulation's log file is written by the GameController when the simulation is switched off (by pressing the **ESC** button on the keyboards, pressing the editor's stop button or when all the special agents have been removed from the scene); when this happens the GameController creates a new file called *Output.txt* 4.19 (if it does not exist) or empties it. Firstly it writes the total number of special agents in the simulation and the maximum time reached, then starts copying in it the first line of each single agent's log file, then the second line, and so on until it reaches the end of all the files. Since the fact that an agent that has finished its job is removed from the simulation, some agent log files can be longer than the others, but this does not make any problem. Another object with the same components of the GameController is the SceneController, which can be defined as the manager of the scene containing the menu.

```
}

1 riferimento
private void CreateSpecialAgent(GameObject person, string fileline)
{
    var line = fileline.Split(' ');
    agentLister.WriteLine(line[0]);
    person = Instantiate(specAgent) as GameObject;
    person.name = line[0];
    var script = person.GetComponent<SpecialAgent>();
    var renderer = person.GetComponent<Renderer>();
    renderer.material.color = colors[line[2].ToLower()];
    script.agent.speed = int.Parse(line[3]);
    for (int k = 4; k < line.Length; k++)
    {
        script.objectives.Add(line[k]);
    }
    persScript = person.GetComponent<AgentsLog>();
    persScript.ID = counter; // set ID of each agent with an incremental number
    Vector3 spawnPos = GameObject.Find(line[1]).transform.position;
    person.transform.position = spawnPos;
    counter++;
}

1 riferimento
private void CreateWanderingAgents(GameObject person, string fileline)
{
    var line = fileline.Split(' ');
    person = Instantiate(wAgent) as GameObject;
    var script = person.GetComponent<WanderingAgents>();
    script.wanderRadius = float.Parse(line[2]); // set radius of the zone
    script.wanderTimer = float.Parse(line[3]); // set time passed in that zone
    script.agent.speed = float.Parse(line[4]); // set agent speed movement
    /*persScript = person.GetComponent<AgentsLog>();
    persScript.ID = counter; // set ID of each agent with an incremental number*/
    Vector3 spawnPos = GetRandomPoint(terrainCenter, (terrainSize.x/2) - 10);
    person.transform.position = spawnPos;
    //counter++;
}
```

FIGURE 4.17: Functions for creating both types of agent.

```
colors["blue"] = Color.blue;
colors["red"] = Color.red;
colors["yellow"] = Color.yellow;
colors["green"] = Color.green;
colors["cyan"] = Color.cyan;
colors["black"] = Color.black;
colors["white"] = Color.white;
colors["magenta"] = Color.magenta;
colors["grey"] = Color.grey;
```

FIGURE 4.18: The color dictionary.

This object populates the dropdown menu present in the scene with all the files present in the *ConfigFiles* directory, manages the button that allows the user to choose the configuration file to use for the next execution of the simulator and keeps this last one in memory. The scene containing the menu is very simple and all the objects, except for the SceneController, are destroyed when loading the scene with the simulator. The SceneController is maintained because it keeps in memory the path of the configuration files needed by the GameController.

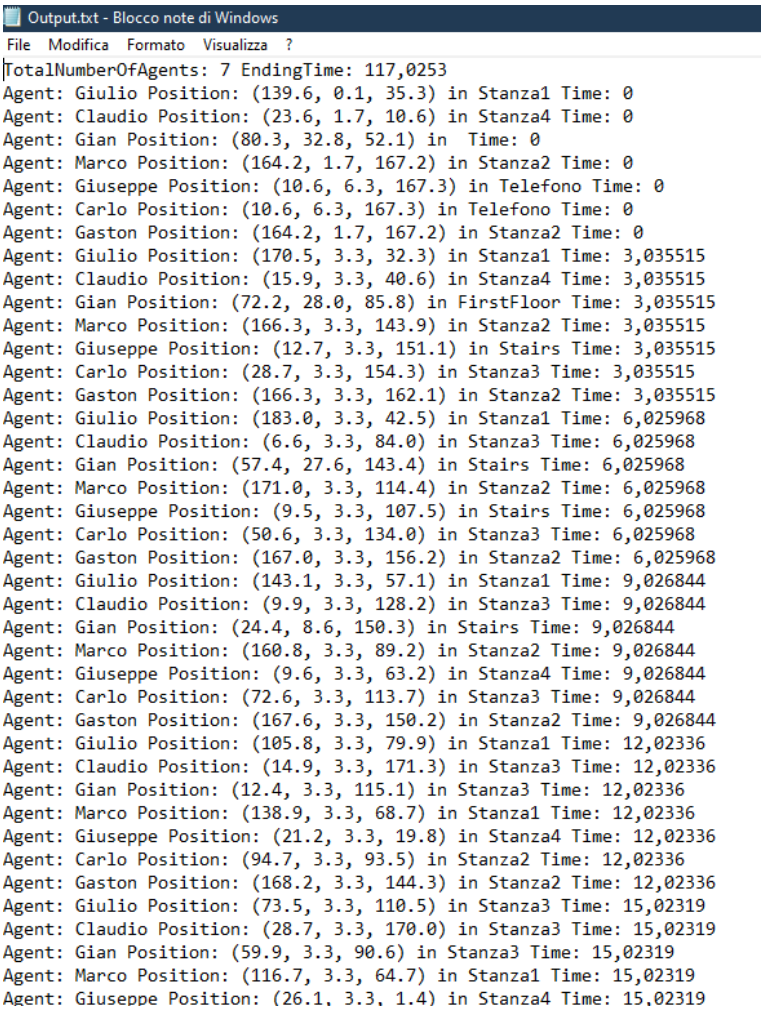

FIGURE 4.19: Example of general log file.

4.1.8 The Main Camera

Talking in general of the simulator there is to say that there is the possibility to explore the environment using the main camera for navigating around it. Before talking about how this camera works it could be useful to define what a virtual trackball is: This is a sphere created around an Object, in our case the environment in which the simulation takes place), and that with mouse inputs allows the user to move around the object and to zoom in and zoom out. So the main camera that is used for watching the simulation from the above is mounted on a virtual trackball, and by holding the left button of the mouse and moving the mouse icon around, the camera moves on this sphere; then it is also possible to get closer or to get further by rolling the mouse wheel. This virtual trackball is implemented in a script attached to the main camera: *TrackBallCamera.cs*;

4.2 How to Install and Use DigForSim

There are two ways for using this application, the first one is directly from the Unity editor, while the second one is creating an executable file. The first method is the one used by programmers, but allow also to a user (if he owns Unity on his PC) to run the project directly from the editor, after that the project directory has been placed in the right location for enabling Unity to find it. When starting the editor there are 3 buttons on the top of the Unity window, as could be seen in Figure 4.7; those buttons are: **Play**, **Pause** and **Skip**:

- **Play:** Pressing this button the user starts the execution changing its functionality from play to stop. Stopping the execution interrupts the application that will have to be restarted.
- **Pause:** Pressing this button pauses the execution that can be restarted repressing it.
- *Skip:* This button can be very useful for debugging and seeing what happens frame by frame. It allows to execute the application frame by frame.

The execution starts from the scene containing the menu for selecting the configuration file that the user wants to use. After clicking the select button the simulation starts, loading the scene containing the environment and the agents. The project directory contains different subdirectories and some of them are important for the user: *AgentsLog, ConfigFiles* and *SimulationLogs*. In the first one are saved all the single agent's log files, in the second one are saved different configuration files that the user could use for the simulation and in the last one the simulation saves the general simulation's log file. Those are all the things that a user has to know about the usage of the simulation thanks to the Unity editor.
If the user wants to create the executable file for the application the first thing to do is to go in the GameController script and comment (putting `/*` before the first line of code and `*/` after the second) the two lines of code that manage to stop the editor when the application finishes; this because if these lines are kept uncommented Unity does not allow to build an executable file. Then there is to create or import the map representing the environment wanted for the simulation. Then there is to put it on the terrain already present in the scene. In case the terrain is too big or too small it needs to be resized, to let the simulation create

all the wandering agents in a better way; the creation of wandering agents, in fact, is based on terrain's boundaries. After having done all the previous things the executable file can be created by going in the File tab on top left corner of the editor window and selecting the *Build Settings...* option; at this point the editor opens a window with all building settings and there is to click on the button **Add Open Scenes** because there are multiple scenes to build, put them in the right order, and then select one of the two option Build or Build and Run creating the executable file.

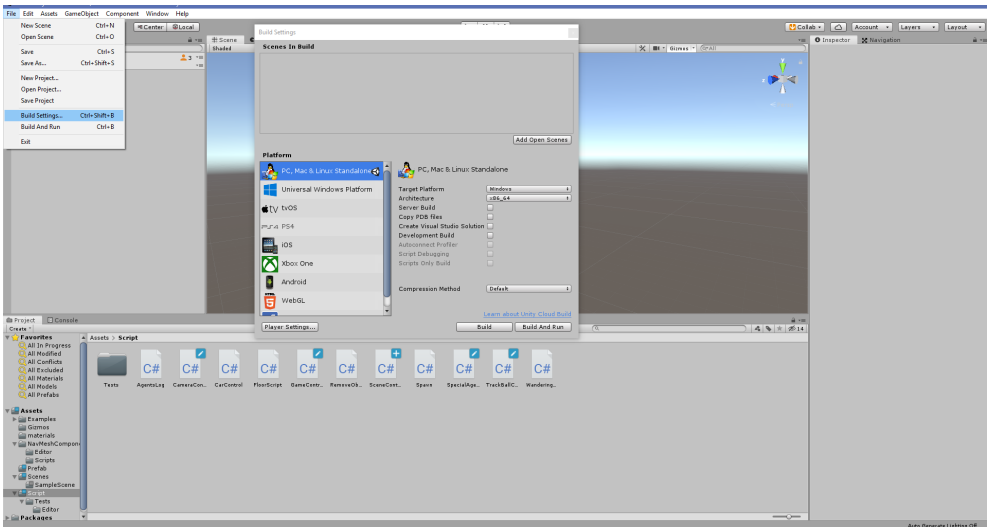

FIGURE 4.20: The window for creating the executable file.

It is important to build the executable file in a directory containing a copy of all the directories that are useful for the project, as the one for agents log files. Then the simulation can be run and thanks to the selection menu it is possible to choose the configuration file to use for the current execution. Figure 4.20 shows the window from which building the executable file. The implementation of DigForSim required more or less 700 lines of code for what is used in the simulation and other 150 lines for other scripts that should have controlled objects that we have decided not to use in the simulation, as a script for a agents of type *car*.

Chapter 5

DigForReason: Reasoning on DigForSim Output

5.1 DigForReason Implementation

5.1.1 Tool Used

For implementing this tool we had two main possibilities: using *html & javascript* or using *java* with the *java.swing* library for interfaces. The first choice could have given a better visual output, but we should have had to create a fake web page not requiring an internet connection for the usage. The second one seemed to be more scarce in terms of visualization, but thanks to the window builder tool it gave the possibility to graphically build the interface by dragging and dropping elements as *text label, buttons, and dropdown menu*, allowing us to focus just on the code for answering functions. We had to download eclipse and to extend it with the *SWING* libraries and the *Window Builder* tool, going in the "**Install New Software window**" (figure 5.1) under the "**Help**" tab; then there is to choose the the version of eclipse that we possess in the "**Work with**" dropdown menu and selecting from *general purpose tools* all the elements starting from the first "Swing" element to the last element of *general tools* list(figure 5.2). Implementation of DigForReason required more or less 750 lines of code of which about 300/350 were automatically created by the window builder tool to create the GUI.

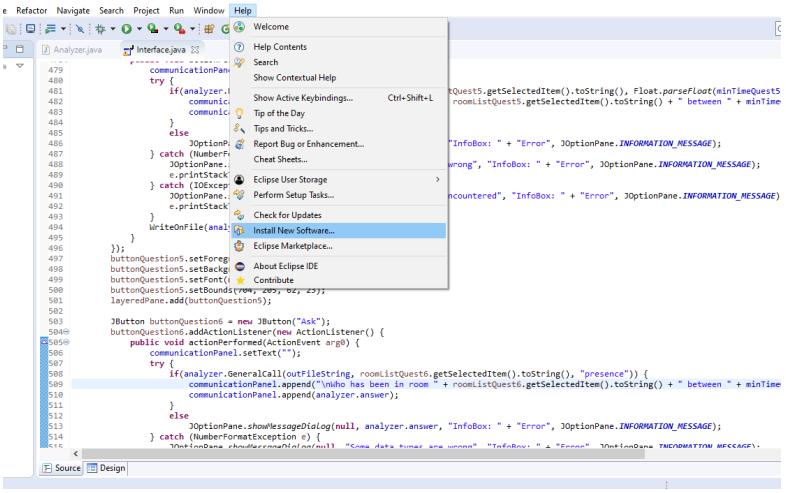

FIGURE 5.1: How to go in the Install New Software window.

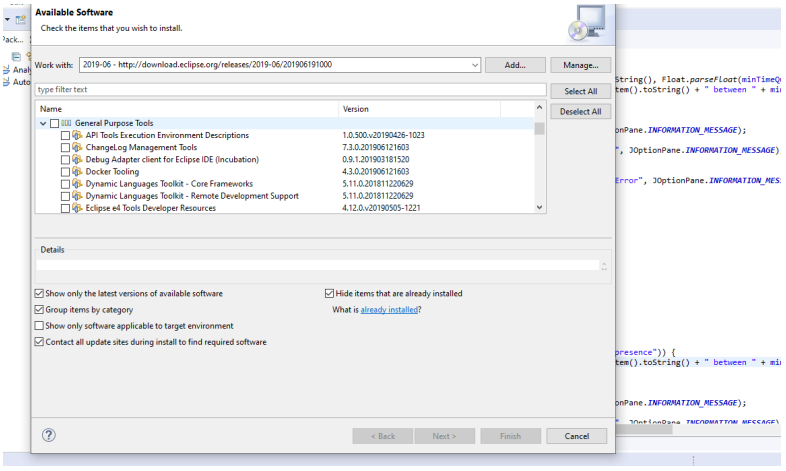

FIGURE 5.2: Where to find general purpose tool element.

This application can be executed like any other java application using eclipse, other IDE or using a terminal. Once that an IDE is installed and all the important directory are set in the program directory this can be executed (the first step it is needed just if the user wants to use an IDE).

5.1.2 Questions

The project is divided into two files, one containing the GUI, *Interface.java*, and the other containing all the functions for answering to available questions *Analyzer.java*. Initially, we have inserted a label saying to chose the three files useful for answering questions and the choice of these files is entrusted to three drop-down menus. Files that the user has to choose are the scene file, that is one

of the obtained simulation's output file, the population file that is simply one of the AgentList file created by DigForSim and that contains name of all entities in the simulation, and then the rooms file, this is a file containing names of all rooms/zones present in the simulation and should be created by the user for each environment (it should be exactly as a population file but with names of locations). These dropdown menus are populated taking the lists of files *OutputFiles, PopulationFiles & RoomsFiles* directories and inserting all the elements of each one in the correct menu. Then, thanks to file names contained in these menus, is possible to take the related file from its directory reading it in order to obtain pieces of information from its content. It is important that all these files are coherent with each other in order to answer correctly to questions; in case of lack of coherence answers are completely useless, or worst, DigForReason can return some errors. In order to avoid this problem, a suggestion is to name files contained in each directory in a similar way as *SceneA, PopulationA & RoomsA* in order to understand which files to choose together.
Once that the choice of files worked we moved to questions' implementation, for example, *Which were movements of a certain agent during all the simulation?*. For this question, the GUI presents a label for the text of the question, a dropdown menu for choosing the agent that we want to study and a button for asking that question to the reasoner. A similar question but based on the room is: *Who has been in a certain room during all the simulation?*; this question presents the same GUI components of the previous one, but in this case, the dropdown lets the user choose one of the rooms/zones in the environment instead of an agent. Both these questions are solved with a single call to a function named **GeneralCall(fileName, element, type)** 5.3 where parameters passed are three string defining: the file that the reasoner has to analyze, the element to be considered (Room's name or Agent's name) and the type that means if we are asking something relative to agents or to rooms/locations. In order to better understand the usage of this third parameter, we have to explore the specific versions of those two questions: *Where has been a certain agent during a certain interval of time? & Who have been in a certain room/zone in a certain interval of time?*. First of all, there is to say that for these questions the GUI presents two elements more than the dropdown menu and the question's label on the screen: two textfileds defining the boundaries of the interval of time (MinTime and MaxTime). Functions that are

```
public boolean GeneralCall(String fileName, String elem, String type) throws IOException {
    String[] lineElems;
    BufferedReader br = GetOutputFile(fileName);
    answer = "";
    line = br.readLine();//Reading the MaxSimulationTime in order to pass it to functions as MaxTime
    lineElems = line.split(" ");
    br.close();
    if(type.equals("movements"))
        return movementsInInterval(fileName, elem, 0f, Float.parseFloat(lineElems[3].replace(",", ".")));//replacing the "," with the "." because
                                                                                                          //parseFloat does not understand the "," char
    else
        return PresenceInRoomInterval(fileName, elem, 0f, Float.parseFloat(lineElems[3].replace(",", ".")));

}
```

FIGURE 5.3: The GeneralCall() function.

supposed to answer to these questions are:**MovementsInInterval(fileName, element, MinTime, MaxTime) & PresenceInRoomInterval(fileName, element, MinTime, MaxTime)** and the type parameter seen for the **GeneralCall(...)** decide which one of these two functions to call with **MinTime = 0 & MaxTime = MaxTimeOfSimulation** depending on its value: *movements* (Calling the **MovementsInInterval(...)**) or *presence* (Calling the **PresenceInRoomInterval(...)**).

Talking more in the detail of these two last functions there is to say that fileName and element are exactly the same things said for **GeneralCall(...)** while MinTime and MaxTime define the interval boundaries. This two functions, as all the others used for answering questions, read the selected simulation's output file line by line, but before doing this they first check that *MinTime* is lower than or equal to *MaxTime*, and in case this condition is not satisfied a little window pops up asking to modify values. Then they check for each line of the analyzed file and if it is in line with given parameters it generates a text to attach to the **answer** variable present in the analyzer. Once that the current function has checked all the lines, the answer to the question is ready and the interface can show it to the user. In addition to these two functions, there is a third one that makes use of an interval of time: *Is it possible that two entities have met in a certain interval of time?*. The GUI for this question shows a question's label, two dropdown menus, one for each agent to select, two textboxes for the time interval, exactly as before, and the button to execute the **Meeting(fileName, Agent1, Agent2, MinTime, MaxTime)**(the code can be seen in Section 7 function 9) function. This is probably the more complex question; it starts checking that selected entities are different and checking that *MinTime & MaxTime* are valid exactly as the two previously mentioned functions. Then it reads the chosen file line by line and once it has found a correspondence for each chosen agent it creates a list of relevant elements for the answer using the **CreateMeetingInfo(current analyzed line)**

5.4 function, then it compares them to decide if it is possible that they have met or not.

```
public List<String> CreateMeetingInfo (String[] info){//This function will create the ArrayList with useful information for the Meeting function
    List<String> aux = new ArrayList<String>();
    aux.add(info[1]);
    aux.add(info[7]);
    info[3] = info[3].replaceAll(",", "");
    info[5] = info[5].replaceAll(",", "");
    aux.add(info[3].replace("(",""));
    aux.add(info[5].replace(")",""));
    aux.add(info[info.length -1]);
    return aux;
}
```

FIGURE 5.4: Function that creates a structure with just useful information for meeting question.

The choice is done considering if agents are in the same room and at a certain distance considering their coordinates **(x,y,z)** (this is defined checking if: x2 - 10 $\leq$ x1 $\leq$ x2 + 10 and same for z). Once found if two agents have met or not it attaches a text to the **answer** variable, so that interface can use it. The last question to which DigForReason can answer is *Who has met a certain entity? where and when they have met?*.The GUI for this function shows the question's lable, just one dropdown menu for choosing the agent, and the button for executing the **AllMeetings(fileName, Agent, ListOfAgents)** 5.5 function. This is the general version of the last function described, in fact calls, once for each element of *ListOfAgent*, the **Meeting(fileName, Agent, ListOfAgentElement, MinTime, MaxTime)** with **MinTime = 0 & MaxTime = MaxTimeOfSimulation**.

```
public String AllMeetings(String fileName, String person, ArrayList<String> population) throws NumberFormatException, IOException {
    String[] lineElems;
    String partialAnswer = "";
    BufferedReader br = GetOutputFile(fileName);
    answer = "";
    line = br.readLine();
    lineElems = line.split(" ");
    for(int i = 0; i < population.size(); i++) {
        if(Meeting(fileName, person, population.get(i) ,0f, Float.parseFloat(lineElems[3].replace(",", ".")))){
            if(!answer.contains("have never met"))
                partialAnswer = partialAnswer.concat(answer);
        };
    }
    if(partialAnswer.isEmpty())
        answer = person + " have met noone";
    else
        answer = partialAnswer + "\n";
    return answer;

}
```

FIGURE 5.5: The AllMeetings Function.

5.1.3 Communication Panel and Output file creation

DigForReason needs to be able to visualize in some way what it discovers analyzing the simulation's output files, and for doing this it uses what we have called communication panel, a bigger textfield put on the right side of the window that

shows the question and the answer obtained. There are also some error messages that are not shown in the communication panel. Here is a list of possible answers and error messages:

- *Is it possible that two entities have met in a certain interval of time?:*
  - *Communication Box:* If all the data inserted are correct and coherent it prints in the communication panel: **Agent1 and Agent2 have met in LocationN at Timestamp**. This happens for each different Timestamp of the interval in which they have met. If they have not met during the simulation it prints **Agent1 and Agent2 have never met**.
  - *Pop Up:* If the two agents chosen are the same or if the boundaries of the time interval are inverted the function returns a warning pop up saying: **MaxTime is greater than MinTime or you have chosen twice the same person. Clicking on the OK button the application returns to the initial GUI**
- *Where was a certain entity in a certain interval of time?:*
  - *Communication Box:* If all data are inserted correctly it prints: **AgentN has been in LocationN at Timestamp** for each different Timestamp in included in the interval.
  - *Pop Up:* If MinTime and MaxTime are inverted it pops up a window with the warning: **MaxTime is greater than MinTime.** or if the specified agent is not present in the simulation in the specified interval of time it pops up this warning message: **AgentN is not present in the simulation or the specified time interval for that agent is not represented in the simulation. Please check that Scene and Population files are correctly related and if not switch them**.
- *Where has been a certain agent during the simulation?:* The answer to this question cannot give errors, so there is no possibility to Pop Up a warning message, and it always answer: **AgentN has been in LocationN at Timestamp** for each Timestamp of the simulation.
- *Who have met a certain entity? Where and when?:* The answer to this question, as the previous one, cannot give errors, and it always print

- *Who has been in this room in a certain interval of time?:*
  - *Communication Box:* If time boundaries are inserted correctly it simply prints in the communication box a list of the Agents that have been in the specified location during the specified interval of time. If no agent has been in that room it prints: **noone has been in LocationN**.
  - *Pop Up:* Also this one in case of inverted boundaries for time interval pops up a warning message saying: **MaxTime is greater than MinTime**.
- *Who has been in this room during the simulation?:* The answer to this question cannot give errors and it prints exactly the same answers of the question seen before.

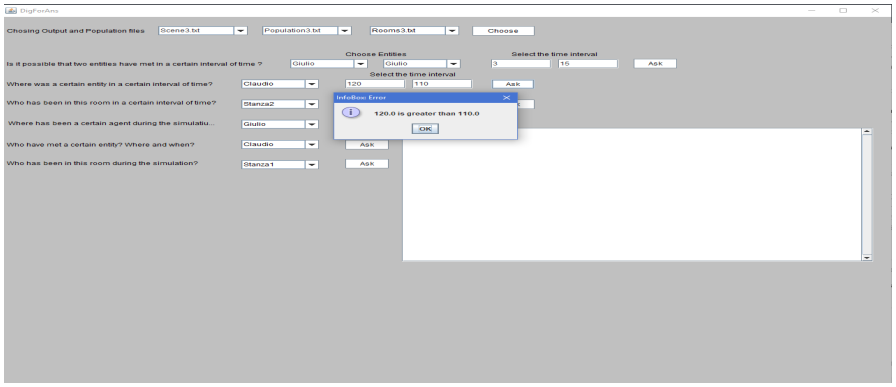

FIGURE 5.6: The warning pop up is.

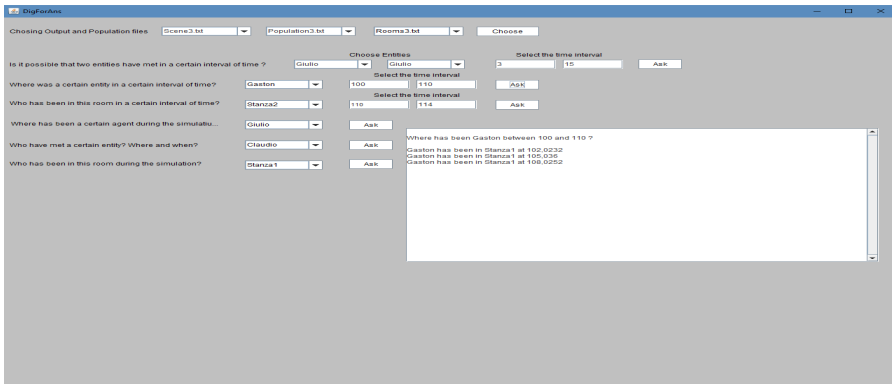

FIGURE 5.7: Example of communication panel output.

Since the fact that each time a question occurs the communication panel is cleared, answers are also saved in some text files contained in the *CommunicationFiles* directory. Every time that a triple of files is chosen, a fourth file is created and its name is the concatenation of the three files' names; if the file already exists is updated with the new answer. These files are created just in order to keep a trace

of what has been already asked to DigForReason keeping a sort of Log of obtained answers. These files are useful also because it would have been more difficult to do this analysis using the communication box, that if not cleaned can be very difficult to read and understand. An example of this file is shown in Figure 5.8

```
Who have been met by Gaston

Gaston and Gian have met in Stanza2 at 24,02107
Gaston and Giuseppe have met in Stanza2 at 36,02591
Gaston and Carlo have met in Stanza2 at 39,02724


Who have been met by Giuseppe

Giuseppe and Gaston have met in Stanza2 at 36,02591


Who have been met by Marco

Marco and Carlo have met in Stanza1 at 15,03511


Who have been met by Claudio
Claudio have met noone
```

FIGURE 5.8: Example of file created from the contents of the communication panel.

Chapter 6

Experiments

6.1 Experiment: Working on a simple map created in Unity

This test is done on an environment created by us in Unity; this represents two floors house with *LivingRoom, DiningRoom, Kitchen* and *Bathroom* positioned at the ground floor and *Bedroom1, Bedroom2* and *FirstFloorCorridor* at the first floor. This is a simple map but it is good enough for testing this application. Some little cylinders are positioned inside the environment: they represent objects in the scene where is possible to spawn or direct agents. As suggested in Section 4.1.3 we have used Room prefab for the ground of each room, cubes that does not follow physical rules for walls and the prefab provided for the creation of objects in the scene that are: *DiningRoomWindow, sink(in the kitchen), BathroomSink, Table* and *Phone*. Once that the environment has been created there is to test DigForSim and DigForReason with different populations, starting from a scarce one for reaching a dense enough one to see if both applications can to work on different quantities of data. The first population test has 10 agents: *Giulio, Claudio, Gian, Loredana, Marco, Giuseppe, Carlo, Gaston, Filippo* and *Franco*. They all have different objectives, speed and path to follow, one of them has another agent in its objectives list. We set as printing **time interval 3,00 seconds** and we have inserted also 4 Wandering agents.

This first execution of DigForSim took **117,026** seconds in which we know for sure that *Giuseppe* has met *Gaston* multiple times since the fact that has him

```
Log Time interval
3

Objective Creation:
Table DiningRoom

Agents Quantity
AgentsInSimulation: 5
FastWanderingAgents: 30 30 1 20
MediumWanderingAgents: 40 20 1 10
SlowWanderingAgents: 30 15 1 5

Special Agent
Giulio Kitchen red 15 Phone 4
Claudio FirstFloorCorridor Green 15 Phone 1 Carlo 1 Kitchen 3
Gian Bedroom1 yellow 20 DiningRoomWindow 2
Loredana Bedroom2 Green 15 Sink 2
Marco DiningRoomWindow White 10 Phone 3 Sink 2
Giuseppe Phone black 15 Sink 1 Gaston 1 DiningRoomWindow 3
Carlo Phone grey 10 FirstFloorCorridor 1 Bedroom2 2
Gaston Phone red 2 Sink 4
Filippo DiningRoom magenta 20 Bedroom1 3 Phone 1
Franco Bedroom1 yellow 18 Stairs 0 BathroomSink 1 Kitchen 3
```

FIGURE 6.1: First configuration file used.

as a target. Another thing that we know directly from the configuration file is that *Marco* and *Filippo* have been in the DiningRoom at least once because they start there, and for the same motivation we know that *Franco* and *Gian* have been in Bedroom1 at least once. First, we check if DigForReason agree with these statements. The first question asked is if *Gaston* and *Giuseppe* have met in the interval between seconds 13 and 60 of the simulation in order to take a large portion of time, and as we expected *Gaston* and *Giuseppe* have met (figure 6.2). As we can notice there is the timestamp between 45 and 51 that is not shown, this is correct for how this application has been implemented, since the fact that looking at the simulation's output file the two agents in that instant are close but in two different rooms, and DigForReason answer that two agents has met just if they are in the same room. This has been done to avoid the situation in which two agents are close, considering their coordinates, but separated by a wall and DigForReason answers that they have met. Then we have asked to the application who have passed for DiningRoom during the entire simulation and we have discovered that three entities have been there: *Marco, Filippo, Gian* and *Giuseppe*; this is exactly in line with what we thought about the possible answer. We have also asked to DigForReason who has been in the Bedroom1 and we discovered that as we thought *Filippo, Gian* have been there, but also *Franco* passed in Bedroom1 (figure 6.3 &

6.4).

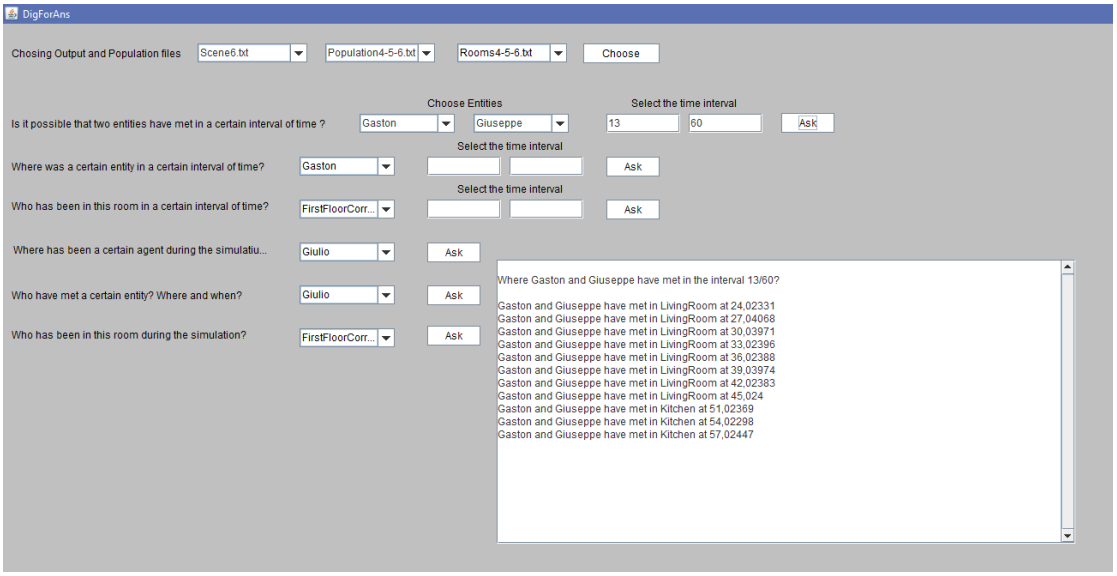

FIGURE 6.2: Has Gaston met Giuseppe in this scenario?

```
Where Gaston and Giuseppe have met in the interval 13/60?

Gaston and Giuseppe have met in LivingRoom at 24,02331
Gaston and Giuseppe have met in LivingRoom at 27,04068
Gaston and Giuseppe have met in LivingRoom at 30,03971
Gaston and Giuseppe have met in LivingRoom at 33,02396
Gaston and Giuseppe have met in LivingRoom at 36,02388
Gaston and Giuseppe have met in LivingRoom at 39,03974
Gaston and Giuseppe have met in LivingRoom at 42,02383
Gaston and Giuseppe have met in LivingRoom at 45,024
Gaston and Giuseppe have met in Kitchen at 51,02369
Gaston and Giuseppe have met in Kitchen at 54,02298
Gaston and Giuseppe have met in Kitchen at 57,02447
```

Looking at the configuration file 6.1 we also know that *Claudio* has surely been in the *FirstFloorCorridor* and in the *Kitchen*, but knowing that these two rooms are positioned on two different floors we also know that he also used stairs for reaching the first floor. Looking at *Claudio's* objectives we notice that he has to reach the phone located in the LivingRoom, so also this room should be in the answer; he has also to reach *Carlo* so he probably passed in some of the rooms in which *Carlo* has passed. First of all to test the correctness of the answer we have to know where

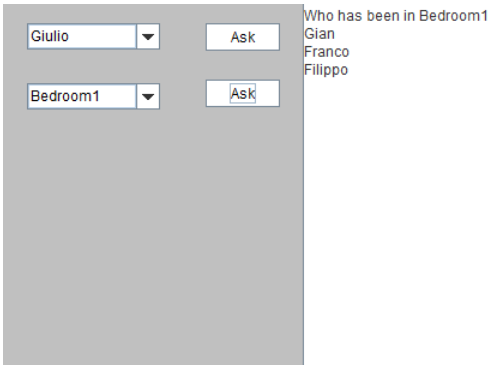

FIGURE 6.3: Who have been in Bedroom1 during the simulation?

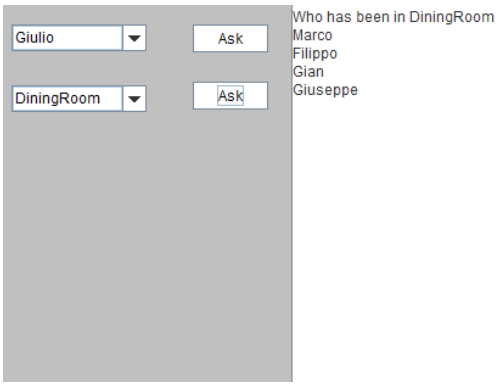

FIGURE 6.4: Who have been in the dining room during the simulation?

these two agents have met so we ask to DigForReason: *Where Claudio and Carlo have met in the interval 0/115 ?* using 115 as max time because the max time of this scenario is 117; the answer to this question is that they have met on *Stairs* and in *Bedroom2*, so we know that *Claudio* has also passed in Bedroom2. Now we know that *Claudio* has been in *FirstFloorCorridor, Bedroom2, Stairs, Kitchen and LivingRoom* and we can ask to DigForReason: *In which room has been Claudio in this scenario?* obtaining that he has been exactly in the mentioned rooms (figure 6.5). In all questions taken into account, we have not considered timestamps because we just wanted to know where the agent has been.

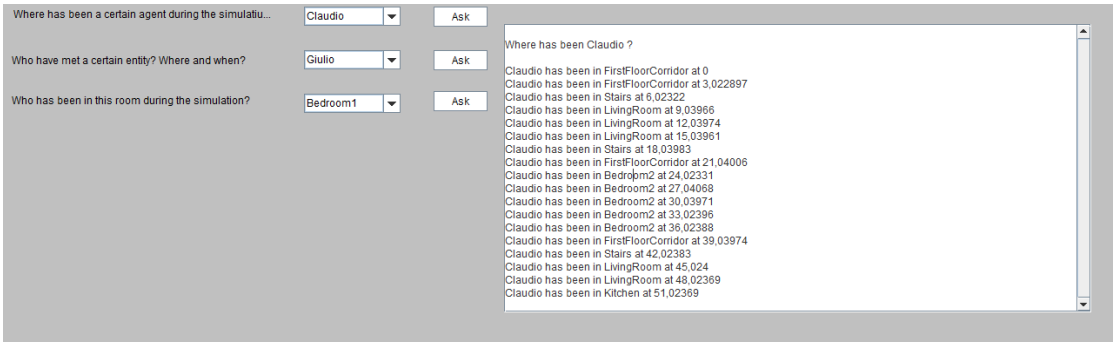

FIGURE 6.5: Where has been Claudio in this scenario?

```
Where has been Claudio ?

Claudio has been in FirstFloorCorridor at 0
Claudio has been in FirstFloorCorridor at 3,022897
Claudio has been in Stairs at 6,02322
Claudio has been in LivingRoom at 9,03966
```

```
Claudio has been in LivingRoom at 12,03974
Claudio has been in LivingRoom at 15,03961
Claudio has been in Stairs at 18,03983
Claudio has been in FirstFloorCorridor at 21,04006
Claudio has been in Bedroom2 at 24,02331
Claudio has been in Bedroom2 at 27,04068
Claudio has been in Bedroom2 at 30,03971
Claudio has been in Bedroom2 at 33,02396
Claudio has been in Bedroom2 at 36,02388
Claudio has been in FirstFloorCorridor at 39,03974
Claudio has been in Stairs at 42,02383
Claudio has been in LivingRoom at 45,024
Claudio has been in LivingRoom at 48,02369
Claudio has been in Kitchen at 51,02369
```

The last check that we can do is on timestamp, and we can use the case seen previously. Asking to DigForReason and looking at the simulation's output file created by DigForSim of this scenario we notice that the two last timestamps in which Carlo compares are *24,02331 & 27,04068* in Bedroom2 that is in line with the moment in which *Carlo* has met *Claudio* and the moment in which this last agent has been in Bedroom2. We can also check if the code works correctly with this question, in fact these two agents have met in Bedroom2 just at *24,02331* and it is correct, because at this timestamp their coordinates **(x, y, z)** are : **Claudio (223.9, 28.0, -203.0)** and **Carlo (228.1, 28.0, -199.9)** so they respect the fact that both X and Z are in a range of -10/+10 the one from the other, while for timestamp *27,04068* they are **Claudio (244.3, 28.0, -206.6)** and **Carlo (248.0, 28.0, -218.9)** and in this case the Z coordinate does not respect the *InRange* constraint.

6.2 Experiment: Working on a simple map imported in Unity

With this experiment, we want to show how to import a model representing the environment more in the detail, to let the user understand which are the fundamental steps for making it work, and also to show that DigForSim and DigForReason

work fine on imported maps. First of all, there is to download an editor for 3D models (if possible one of the ones able to create a *.fbx* file that is the one used for our trials). We have used Blender, which is free for everyone, and allow to do both complex and simple models; for our purpose also simple models work fine. The creation of simple models in blender works more or less as in Unity, but it has also other functionalities that we have not seen for the model created for this second experiment. We have decided to use the model of a single floor house; it is important to pay attention to the orientation and the size of the elements inserted, because these are computed differently than in Unity, with a correspondence that is more or less **1m** (for blender) *10* size units in Unity for planes while is **1m : 1unit** for what concern terrains. A good idea could be to make the terrain a little bigger than the model. Figure 6.6 and 6.7 shows the model before and after being imported.

Once that the model has been imported, it does not have any *Collider* attached so there is to click on the model in the project inspector (on the bottom of the Unity window) opening the inspector of the model, as it happens for every prefab in the scene; here there is to check two voices that are not checked by default: **Read/Write Enabled & Generate Colliders** as shown in Figure 6.8 (here the model is in the prefab directory but by default is inserted in the asset directory). Once that the model has been imported it is important to go inside each floor object (planes) and through the object inspector on the right there is to check the **convex & IsTrigger** options of the *Mesh collider* component. Finally, the user can populate the scene with all the objects needed for the simulation using the provided cylinder prefab.

When everything is set up we can launch the simulation using the configuration file shown in Figure 6.9. This model presents six rooms: *Kitchen, Bedroom, Bathroom, Corridor, LivingRoom, DiningRoom* and seven agents *Giulio, Jack, Rosa, Damiano, Alex, Carl, Kraig* so it is very simple, in fact, the aim of this experiment is showing that this application works fine also importing the environment from an external tool. The first question is: *Where has been Giulio during the simulation?*; looking at the configuration file we know that this agent has been at least in the Kitchen because it is created in that room and in the Living room because its objective is the sofa, and this is located in the LivingRoom. The answer of DigForReason is (figure 6.10):

```
Where has been Giulio ?
```

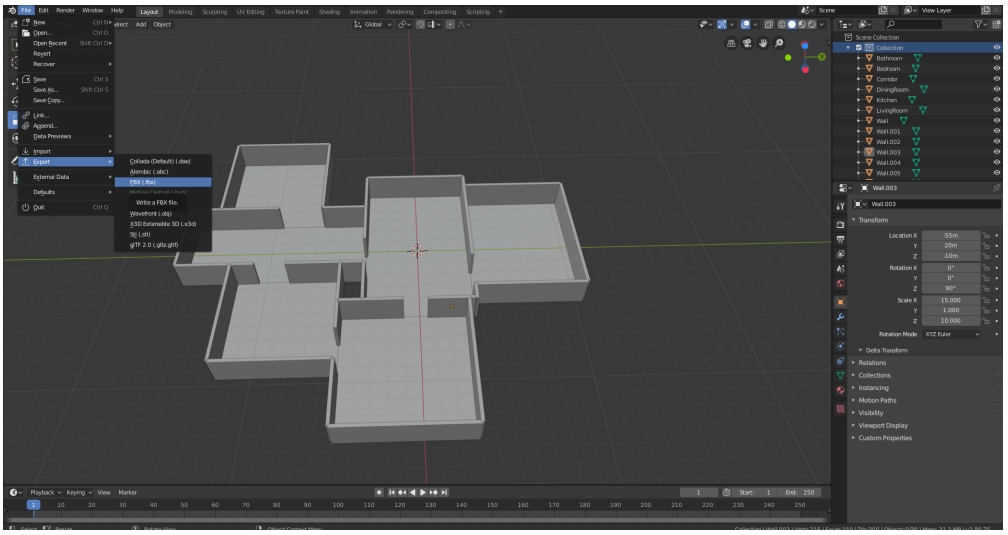

FIGURE 6.6: Model created in blender.

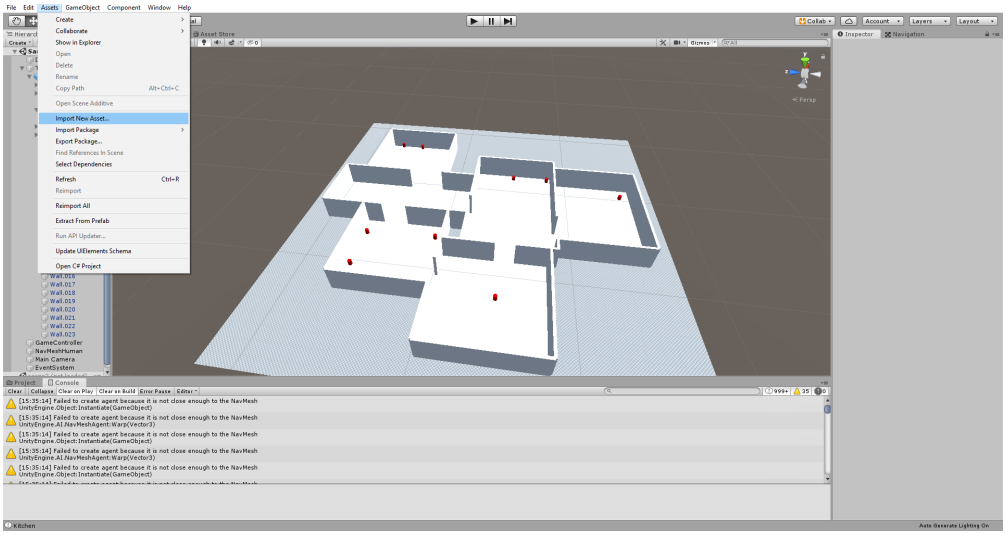

FIGURE 6.7: Model imported in Unity and populated with objects.

```
Giulio has been in Kitchen at 0
Giulio has been in Corridor at 5,032429
Giulio has been in LivingRoom at 10,02925
```

Then we asked to DigForReason: *Who has met Giulio during the simulation?* and the answer was:

```
Who have been met by Giulio

Giulio and Damiano have met in Kitchen at 0
```

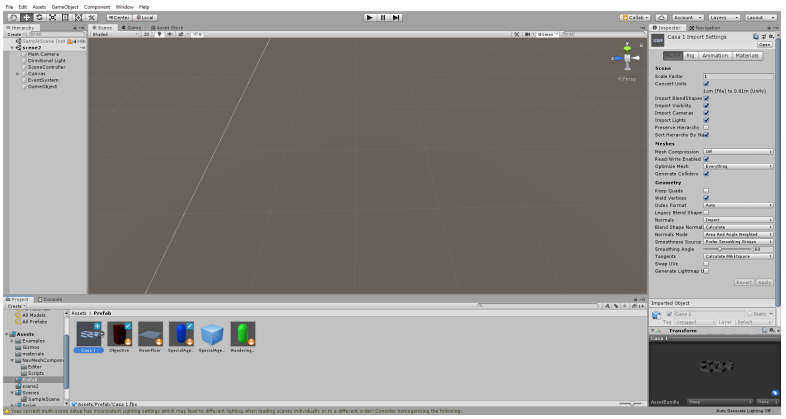

FIGURE 6.8: On the right is shown how to set the imported object.

```
Log Time interval
1

Objective Creation:
KitchenTable Kitchen

Agents Quantity
AgentsInSimulation: 15
FastWanderingAgents: 30 30 1 20
MediumWanderingAgents: 40 20 1 10
SlowWanderingAgents: 30 15 1 5

Special Agent
Giulio Kitchen red 15 Sofa 10
Jack LivingRoom yellow 10 Fridge 3 Table 15 Bed 10
Rosa Bathroom green 8 KitchenSink 4 Television 1 Sofa 10
Damiano Kitchen blue 15 Corridor 1 LivingRoom 1 Table 5 WC 10
Alex Bed grey 16 Corridor 1 Shelf 5 Table 10
Carl Fridge black 20 BathroomSink 7 Shelf 6 Bedroom 5
Kraig Corridor white 7 Shelf 2 Fridge 3 KitchenTable 10
```

FIGURE 6.9: Configuration file for experiment 2.

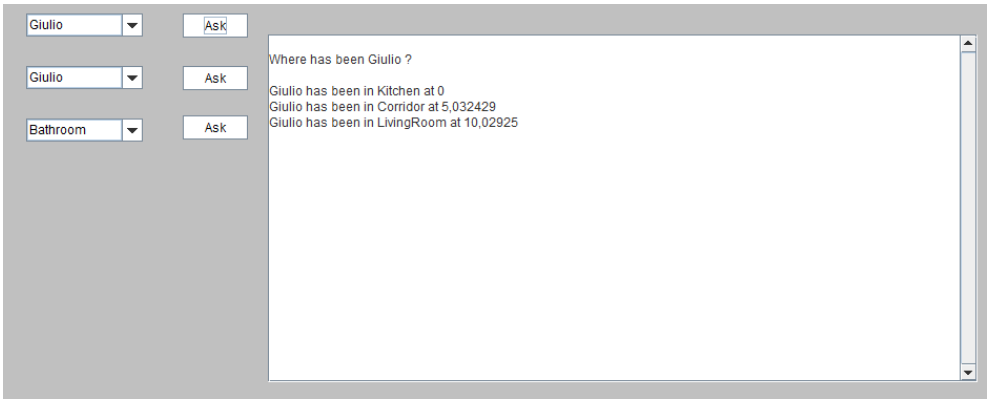

FIGURE 6.10: Where has been Giulio during the simulation?

```
Giulio and Damiano have met in Kitchen at 1,029443
Giulio and Damiano have met in Kitchen at 2,03416
Giulio and Damiano have met in Kitchen at 3,028367
Giulio and Damiano have met in Corridor at 4,038116
```

```
Giulio and Damiano have met in Corridor at 5,039979
```

Then we have asked: *Who has been in Bathroom during the simulation?* and the answer was:

```
Who has been in Bathroom

Rosa
Carl
Damiano
```

Now from the configuration file, we know that the agent Rosa was created in the bathroom, so it is correct that this agent compares in the answer to that last question. We now check if also the other two have been there looking at the simulation's log file:

```
Agent: Carl Position: (311.8, 3.3, -156.5) in Bathroom Time: 24,03323

Agent: Damiano Position: (320.0, 3.3, -151.9) in Bathroom Time: 33,03125
```

Here we wrote just these two lines, but both agent spent more than just one timestamp in this room.

6.3 Experiment on DigForSim performance

With this experiment we want to test performances of DigForSim in different cases, considering data as: the average dimension of the environment, the number of rooms, the number of wandering agents, the number of special agents; since the fact that we do not know which data really influence the simulation we try to understand it[1]. Our first environment test is the one used for 6.2 and we start tests by using data of Experiment 2 except for the number Wandering agents, that

[1]This tests, as the ones done for DigForReason are done on a Desktop with these components: Intel Core i5 7600 @ 3.50GHz QuadCore, 16,0GB Dual-Channel Unknown @ 1067MHz (15-15-15-36), Motherboard Gigabyte H270-HD3-CF (U3E1), NVIDIA GeForce GTX 1070 (Gigabyte).

is initially reduced to 0 to be increased in the next tests. We want to see the time spent by each agent for finding the optimal path to reach its first objective; this time is the difference between the moment in which the agent finds the shortest path and the time in which the agent has been created. In the first case, we have used 7 Special agents, 0 Wandering agents, a time interval of 1 second and 10 objects; with these constraints, agents take an average time of **0,014** to find the optimal path for their first objective. In the second case, we have used the same constraints except for wandering agents that were 20 and we have obtained an AVG time of **0.015**. If we use 60 wandering agents we have an average time for finding the first path that is **0,018**. We can say that this change is quite irrelevant since the fact that in all cases it takes very little time. One thing that we have to notice is that all these times are in line with how long a single agent takes to evaluate its shortest paths for other objectives, which is always a time between the **0,016 & 0,017**. We have also tried with 200 wandering agents and we obtained that the AVG time is **0,013**, so we can conclude that the number of wandering agents in the scene does not influence performances of the application. All the time values are obtained by taking the average of 5 trials on each case that we have considered, and in all the cases each agent for finding the first shortest path took a time that was between the **0,007 & 0,022**, usually first special agents created took less than the others[2].

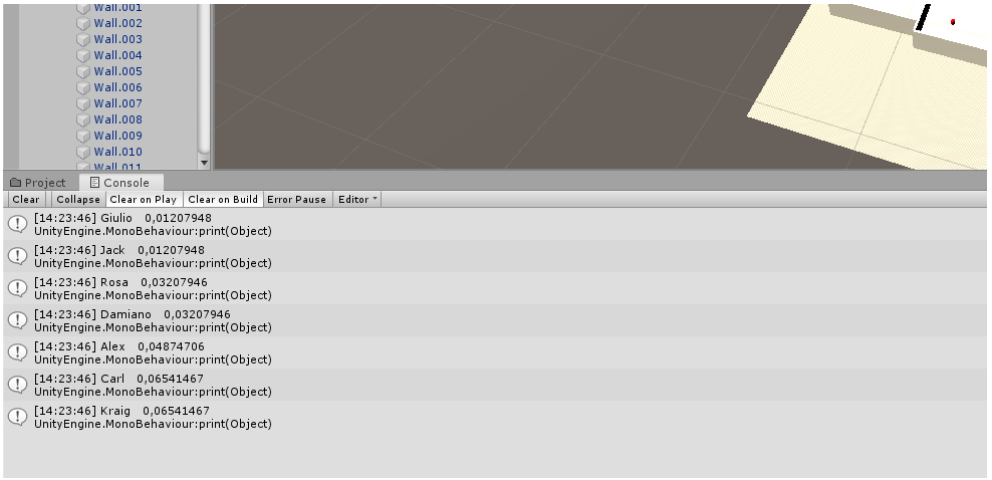

FIGURE 6.11: The effective time each agent takes for finding its first shortest path.

[2]Each average shown in the text is done in this way: for each execution we take the average time that all agents take for finding the first shortest path; when we have this average for five different trials we make the average on them, and this is the final result.

We also tried to decrease the time interval and we have noticed that: the time taken for finding the initial shortest path was pretty much the same, but the simulation does not work fine anymore; using a very low time interval, as *0,01*, timestamps in the output file were not correct, while with a time interval of *0,1* the output file was ok; in both cases the visualization was not fluid anymore and because of that we have decided to keep the time interval equal to one or higher. Then we tried to modify the number of objects in the scene, passing from 10 to 26 to 36, and in these cases we have found higher time values for shortest paths finding. This probably happens because the NavMesh became more complex increasing the number of objects placed on it. Then we tried to change the environment, using the one mentioned in experiment one: this presents two different floors and a quite bigger surface for letting Agents move. We used all the tests that have been useful for testing environment 2, so we have increased the number of objects in the scene from 10 to 26 to 36. With this environment we have noticed that times obtained still increase basing on the number of objects in the scene, but they have also increased because of the bigger surface, so we can conclude that the number of objects and the size of the environment terrain are meaningful for performances of the application; a more complex environment requires more times for setting up the agents. We want also to check if the number of special agents in the scene (so a higher number of shortest path computation) is relevant; for this experiment we ill keep fixed to 40 the number of the wandering agents. The first trial was with 15 special agents, and this gave higher times in all the three cases (10/26/36 objects in the scene); then we tried with 20 agents and also with them we had an increase in the time for finding the first shortest path. We have noticed that all the agents during the simulations took more or less the same time for finding the shortest path for all the other objectives and that is the same of the taken in the other environment (figure 6.12). All results of experiments are shown in table 6.1.

Now we want to try if for a single agent the time spent to find its first shortest path changes basing on the distance between the agent and the objective, that is why we have executed both the simulations already used (for experiment 1 and experiment 2) with just one agent and a few objectives. In this case, we noticed that the time does not depend on the distance from the object. Considering that except for the first objective every agent takes the same time to find the shortest path we could have imagined that. Anyway we have tried on the environment used for experiment 2 and we have increased the distance of the first objective

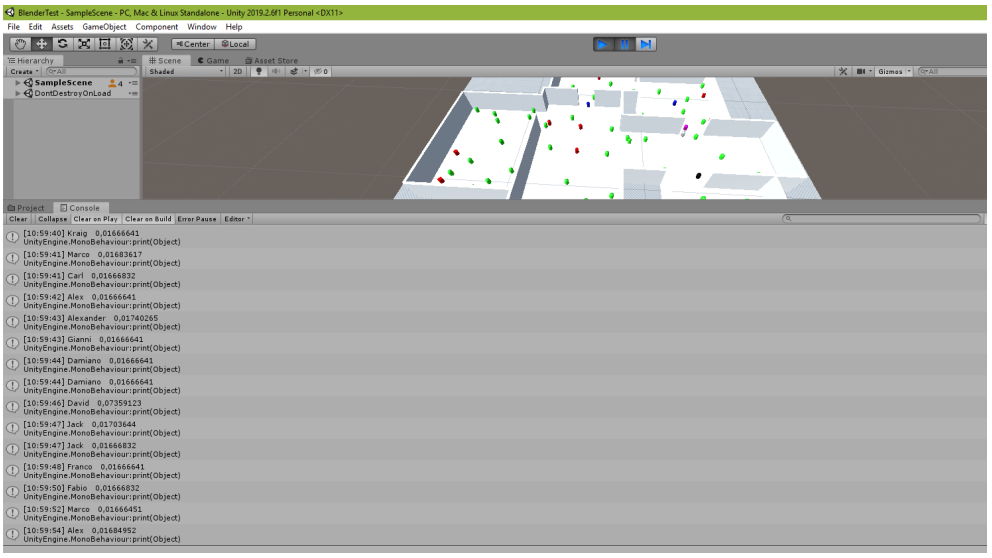

FIGURE 6.12: The usual path finding time for all the other objectives after the first one.

every 5 trials, to make the AVG time; we obtained always a result in between *0,005 & 0,01*. Since the fact that for previous tests we have always changed the dimensions of the *NavMesh* and the number of objects, in this case, we have always used the same map with the same number of objects inside in order not to modify the complexity of the map. In conclusion we can say that performances can be computed just on the time that each agent takes for finding the shortest path to its first objective and that this property depends on the number of objects in the scene, on the dimensions of the *navigation mesh* used (complexity of the scene) and on the number of special agents making this computation. To see the maximum number of agents that can work together in the scene without overloading the memory of the computer, we have inserted in the scene 1000 wandering agents and this gave no problem to the fluidity of the simulation.

6.4 Experiment on DigForReason performance

With this experiment we want to see the time that DigForReason takes for answering a question that depends on the file length, so on the time interval passed to the configuration file, the duration of the simulation and the number of special agents. Before starting there is to say that our solution for DigForReason is not the optimal one because this application was not the focus of this thesis project; it is just a trial to give an overview of how the analyzer should work. We are

N° S.A	N° W.A	N° Objs	Time Interval	Env	AvgTimePF (seconds)	Vis
7	0	10	1	Exp2	0,014	OK
7	20	10	1	Exp2	0,015	OK
7	60	10	1	Exp2	0,018	OK
7	200	10	1	Exp2	0,013	OK
7	0	10	0,1	Exp2	0,02	OK
7	0	10	0,01	Exp2	0,017	NO
7	0	16	0,01	Exp2	0,021	NO
7	20	10	0,01	Exp2	0,011	NO
7	20	16	0,01	Exp2	0,018	NO
7	60	10	1	Exp2	0,018	OK
7	60	26	1	Exp2	0,036	OK
7	60	36	1	Exp2	0,063	OK
15	60	10	1	Exp2	0,039	OK
15	60	26	1	Exp2	0,079	OK
15	60	36	1	Exp2	0,114	OK
20	60	10	1	Exp2	0,051	OK
20	60	26	1	Exp2	0,099	OK
20	60	36	1	Exp2	0,147	OK
7	60	10	1	Exp1	0,021	OK
7	60	26	1	Exp1	0,052	OK
7	60	36	1	Exp1	0,069	OK
15	60	10	1	Exp1	0,063	OK
15	60	26	1	Exp1	0,123	OK
15	60	36	1	Exp1	0,149	OK
20	60	10	1	Exp1	0,082	OK
20	60	26	1	Exp1	0,161	OK
20	60	36	1	Exp1	0,223	OK

TABLE 6.1: Table with result of experiments done on DigForSim.

going to evaluate the time taken to answer each question on different simulation output files and if problems arise if the file is too long. Tests have been done on all the available questions, repeating the current one for 1000 times and returning the seconds used to complete the computation. These trials measures the time that the application takes to answer a question a thousand times writing it on the output file. For each question, the file is emptied to have comparable times. Test are done on three different simulation's output files of different lengths: 310 lines, 700 lines, and 1751 lines. To increase the length of output files the configuration file for the second simulation has been modified by slowing down the agents and giving them more objectives, while for the last simulation we have inserted also some other agents. The length of the answers influences the time taken to answer.

Here we show some and analyze some results obtained for each question available on DigForReason, taking into account an output file obtained by a simulation done on the environment used for experiment 2. Results are shown in table 6.2.

- ***QUESTION 1***

```
Where Damiano and Giulio have met in the interval 1/10?

Damiano and Giulio have met in Kitchen at 1,029443
Damiano and Giulio have met in Kitchen at 2,03416
Damiano and Giulio have met in Kitchen at 3,028367
Damiano and Giulio have met in Corridor at 4,038116
Damiano and Giulio have met in Corridor at 5,039979
```

From the table we can see that on a longer file, the time to answer is lower, this is because in the second simulation one of the agents has been slowed down so the second one went too far to let the program understand that they have met. The small difference of time is given by the fact that the first time DigForReason has to write a little more than in the second, in fact, there is a difference of two meeting statements in the two answers. For what concern the third trials we can see that the time is more than doubled and this is because the file is much longer than the first one, and all checks that the first question does have to be done on all lines.

- ***QUESTION 2***

```
Who have been met by Damiano

Damiano and Giulio have met in Kitchen at 0
Damiano and Giulio have met in Kitchen at 1,029443
Damiano and Giulio have met in Kitchen at 2,03416
Damiano and Giulio have met in Kitchen at 3,028367
Damiano and Giulio have met in Corridor at 4,038116
Damiano and Giulio have met in Corridor at 5,039979
```

```
Damiano and Jack have met in Corridor at 7,037501
```

Here the analyzer checks all the lines so increasing the number of lines we have an increment of the time spent for answering. Besides checks for this question are done on all the pairs of entities in the scene for all the timestamps.

- ***QUESTION 3***

```
Where has been Damiano between 35 and 40 ?

Damiano has been in Bathroom at 35,02987
Damiano has been in Bathroom at 36,03347
```

This question expected a short answer in all cases, so, as we can see from the table it took in all cases a similar time. This is probably because it had to do all checks just on a few lines, while the others are just skipped because out of the range of the time interval..

- ***QUESTION 4***

```
Where has been Damiano ?

Damiano has been in Kitchen at 0
Damiano has been in Kitchen at 1,029443
Damiano has been in Kitchen at 2,03416
Damiano has been in Kitchen at 3,028367
Damiano has been in Corridor at 4,038116
Damiano has been in Corridor at 5,039979
Damiano has been in Corridor at 6,041843
Damiano has been in Corridor at 7,037501
Damiano has been in Corridor at 8,028149
Damiano has been in Corridor at 9,031564
Damiano has been in Corridor at 10,03012
Damiano has been in Corridor at 11,03736
```

```
Damiano has been in LivingRoom at 12,04012
Damiano has been in LivingRoom at 13,03283
Damiano has been in LivingRoom at 14,04332
Damiano has been in LivingRoom at 15,03146
Damiano has been in LivingRoom at 16,02924
Damiano has been in LivingRoom at 17,03935
Damiano has been in LivingRoom at 18,03408
Damiano has been in LivingRoom at 19,03928
Damiano has been in LivingRoom at 20,02934
Damiano has been in LivingRoom at 21,04332
Damiano has been in LivingRoom at 22,03992
Damiano has been in LivingRoom at 23,02865
Damiano has been in LivingRoom at 24,03323
Damiano has been in LivingRoom at 25,02728
Damiano has been in LivingRoom at 26,03816
Damiano has been in LivingRoom at 27,02748
Damiano has been in Corridor at 28,03391
Damiano has been in Corridor at 29,04119
Damiano has been in Corridor at 30,03182
Damiano has been in Corridor at 31,03843
Damiano has been in Bathroom at 32,02787
Damiano has been in Bathroom at 33,03125
Damiano has been in Bathroom at 34,02831
Damiano has been in Bathroom at 35,02987
Damiano has been in Bathroom at 36,03347
```

We can say the same things as for question 3, in fact in this case we simply call the same functions but for all the possible timestamps, so in all the three trials we are doing all checks on all the lines, and increasing the number of these ones we have also an increment of time for answering. Besides having a longer input file leads also to a high possibility of a longer answer and this leads to an increment of the answering time.

- ***QUESTION 5***

```
Who has been in Corridor between 5 and 15

Giulio
Jack
Damiano
Carl
Rosa
Alex
```

This answer requires less effort because there is not much to write in the output file, so the application takes a very short time for answering, and it depends on the length of the answer and the length of the file. The time taken in the second trials is shorter because being slower some agents did not pass in the selected room during that interval of time, so the application had to write less then in the first case; so the major delay is given by the writing phase than by the reading phase.

- ***QUESTION 6***

```
Who has been in Corridor ?

Kraig
Giulio
Damiano
Jack
Carl
Rosa
Alex
```

In this case we have simply to take into account the length of the file, in fact the time increases with the increment of lines to read.

File lines	Question1	Question2	Question3	Question4	Question5	Question6
310	4,4	6,0	4,1	4,1	3,6	3,1
700	4	7,9	4,8	5	3	3,4
1751	9,8	24,6	6,1	5,9	5,7	7,2

TABLE 6.2: Table with result of experiments done on DigForReason Times are taken on 1000 trials for each question.

Chapter 7

Conclusions and Future Work

Looking at experiments done we can see that the two applications work correctly, and that all the answers given by DigForReason are coherent with each other, but both have some constraints. The simulation requires some previous knowledge, as the entities that were present on the scene and their behaviour. The structure of DigForSim has not being thought for representing large zone and a great amount of people; the aim was to provide an application for studying what happened in the location of the crime and its neighborhood without moving too far. In the same way also DigForReason is not thought for working on very long text file, that's why we have chosen to let it work directly on the text file reading it and finding the answer without using a more complex data structure as a database could be. There are anyway some things that can be improved to make the application work better and to improve also the visualization and the interaction that he user can have with it; some of the things that can be improved are:

- **Meetings:** In the application as it currently is if two agents are in different rooms they surely have not met, and this is not always correct; this is because in the time spent on this project there has not been enough time to find a better solution to manage this functionality and the motivation can be found in Section 6.1. A possible solution could be to create a map, with walls position, to pass to DigForReason so that when computing answers it could take it into account. This map or something that works for the same purpose can be anyway created in a future work on this application.
- **Intelligent Agents:** Defining agents as BDI ones, allowing us to model the cognitive stage of the agents, their goals, intentions, and motivations, hence

obtaining more realistic simulations where the psychological dimension is also taken into account. We are exploring two ways to achieve this objective. The first one is creating a bridge between Unity and Jason; Jason is in fact implemented in Java, and tools exist for converting .jar files into .net dynamic libraries[1], that could be used inside Unity. The second is reimplementing a BDI-style reasoning engine directly inside Unity,by exploiting UnityProlog, an ISO-compliant Prolog interpreter for Unity3D[2]

- **Runtime Questions:** It could be useful to give to the user the possibility to create the simulation passing to it also the event in which we are interested, modifying the simulator to let it point out this event. For example, we want to know if two entities have met, so the simulator could stop/pause if the event takes place notifying it in some way. This could be a good way to check how the scene could be in the moment of the analyzed event.

- **Different way to output files from DigForSim:** The text file provided as output in this version of the application is easy to use and simple to create, but it is not possible to search information efficiently since the fact that DigForSim takes a long time to read a very long file (billions of lines for example). A good way to make it more efficient could be to insert information in a database that will be questioned by DigForReason allowing to cross some questions thanks to SQL queries.

- **Enabling agents to interact with objects:** Giving agents the possibility to interact with objects, as the murder weapon, doors or windows could be very useful. For example, interactions with doors (that can be locked and that just certain agents can open or close) could give the possibility to change the conformation of the environment (creating new possible path for agents, removing some path).

- **Graphical improvements:** It could be of great impact to insert some human textures instead of capsules, and inserting textures for objects because it could give a more informative visualization that, combined with some of the previous cited improvements could make more relevant the possibility to explore the environment with agents camera and with the simulation camera.

[1] https://sourceforge.net/projects/ikvm/..
[2] https://github.com/ianhorswill/UnityProlog..

Appendix A: DigForSim Code

Here we will show some pieces of code but the entire directory of DigForSim is available at this link:
https://github.com/Cibie/DigForSim

```
public static Vector3 RandomNavSphere(Vector3 origin,
    float dist, int layermask)
{
    Vector3 randDirection = Random.insideUnitSphere * dist;
    randDirection += origin;
    NavMeshHit navHit;
    NavMesh.SamplePosition(randDirection, out navHit, dist, layermask);
    //It chose the nearest point to the
        chosen one on the sphere on the NavMesh
    return navHit.position;
}
```

Listing 1: Function creating the sphere in which Agent will move

```
//[HideInInspector]
public int ID;
//[HideInInspector]
public string area = "";
// Start is called before the first frame update
private bool ok;
private SpecialAgent spec;
void Start()
{
    //Each Agent generate a proper Log file

    /* if (File.Exists("./AgentsTimes/Times" + ID + ".txt"))
         File.WriteAllText("./AgentsTimes/Times" +
            ID + ".txt", string.Empty);*/
    PrintLog();
    InvokeRepeating("PrintLog", GameController.inst.logInvokeTime,
        GameController.inst.logInvokeTime);
    spec = gameObject.GetComponent<SpecialAgent>();
}

// Update is called once per frame
```

```
void Update()
{
    /*if (!ok && spec.done)// THESE ARE TESTS
    {
        StreamWriter writer = new StreamWriter("./AgentsTimes/Times" +
            ID + ".txt", true);
        var line = spec.timeNeeded;
        writer.WriteLine(line);
        writer.Close();
        ok = true;
    }*/
}

private void PrintLog()
{
    StreamWriter writer = new StreamWriter("./AgentsLog/Output" +
        ID + ".txt", true);
    var line = "Agent: " + gameObject.transform.name + " Position: " +
        transform.position + " in " + area +
        " Time: " + GameController.inst.time;
    GameController.inst.endingTime = GameController.inst.time;
    writer.WriteLine(line);
    writer.Close();
}
private void OnTriggerEnter(Collider other)
{
    if (other.GetType() == typeof(MeshCollider))
    {
        area = other.gameObject.transform.name;
    }
}
```

LISTING 2: Here is the AgentsLog script printing position of agents each time interval

```
public static SceneController inst;

void Awake()
{
    if (inst != null) // Creating an instance of the SceneController
                      // that can be called from other objects to retrieve
                      // the configuration file path
        return;
    inst = this;
}
// Start is called before the first frame update
void Start()
{
    PopulateList();
}

// Update is called once per frame
void Update()
{
```

```
}

public void PopulateList()
// Populate the dropdown menu of the menu scene with
// all files in the "ConfigFiles" directory
{
    //Clear the dropdown menu from default example options
    dropdown.ClearOptions();
    List<string> filesName = new List<string>() { };
    DirectoryInfo dir = new DirectoryInfo("./ConfigFiles/");
    // create an array with all files of the directory
    FileInfo[] files = dir.GetFiles("*.*");
    // for each file in the array add an element in the List
    foreach (FileInfo f in files)
    {
        filesName.Add(f.Name);
    }
    // Add all the elements of the list as option of the dropdown menu
    dropdown.AddOptions(filesName);

}

public void GetConfigFile()
{
    // The SceneController will be kept also when loading the other scene
    DontDestroyOnLoad(transform.gameObject);
    // Getting the selected item of the dropdown menu
    filename = dropdown.options[dropdown.value].text;
    //loading the other scene
    UnityEngine.SceneManagement.SceneManager.LoadScene("SampleScene");
}
```

LISTING 3: Here is the script of the scene controller managing the scene with the menu for selecting the configuration file to use

```
//Creating the NavMesh
    for (int i = 0; i < surfaces.Length; i++)
        surfaces[i].BuildNavMesh();
    //In those line The program skip the first line of the
    //config files and creates the given objective for the simulations
    for (int i = 0; i < filePaths.Length; i++)
    {
        File.Delete(filePaths[i]);
        File.Delete(filePathsType2[i]);
    }
    Cursor.lockState = CursorLockMode.None;
    //Locking the cursor on the foreground of
    // the screen and making it visible
    Cursor.visible = true;
    var terr = terrain.GetComponent<Terrain>();
    // Getting boundaries of the terrain pasesed to the GameController
    terrainSize = terr.terrainData.size;
    terrainCenter = terrain.GetComponent<Collider>().bounds.center;
    string[] filelines = File.ReadAllLines(filepath);
    // Getting the Time interval from the config file
```

```
logInvokeTime = float.Parse(filelines[1]);
print(logInvokeTime);
var line = filelines[4].Split(' ');
// Starting to take information from the Config file
for (int i = 0; i < line.Length; i += 2)
{// Getting objects to insert in the scene
    if (line[i] != "")
    {
        GameObject obj;
        obj = Instantiate(objective) as GameObject;
        obj.name = line[i]; // elem i is the object
        // elem i+1 is the location of which the program
        Vector3 sP = GameObject.Find(line[i + 1]).transform.position;
        // has to take coordinates for creating the obj
        sP.y += 2;
        obj.transform.position = sP;
    }
}
// In those line The program skip description lines and take the
// total number of agents from the fifth line of the file
line = filelines[7].Split(' ');
wanderingAgentsNumber = int.Parse(line[1]);
```

LISTING 4: Here the game controller set up the environment: creating the navigation mesh, setting the time interval and inserting objects

```
// Spawn a certain number of each type of agents
// Here manage the creation of Agents
for (int i = 5; i < filelines.Length; i++)
{
    GameObject person = null;
    if (i >= 13)// This manage the creation of special Agents
    {
        CreateSpecialAgent(person, filelines[i]);
        totAgents++;
    }
    //this manage creation of all Wandering agents
    else if (i >= 8 && i <= 10 && wanderingAgentsNumber != 0)
    {
        line = filelines[i].Split(' ');
        // Here we alternate rounding up and rounding
        // down for managing cases of values as 7,5
        if (!RoundCeiling)
        {// number of current agents is the number of
         // agents of the current type
            numberOfCurrentAgents =
                (int)Mathf.Floor((wanderingAgentsNumber
                 / 100)* float.Parse(line[1]));
            RoundCeiling = true;
        }
        else
        {
            numberOfCurrentAgents =
                (int)Mathf.Ceil((wanderingAgentsNumber
                 / 100) * float.Parse(line[1]));
```

```
                RoundCeiling = false;
            }
            checkPercentage += int.Parse(line[1]);
            for (int j = 0; j < numberOfCurrentAgents; j++)
            {
                CreateWanderingAgents(person, filelines[i]);
            }
        }
    }
    // If the Sum of all percentages it's more than 100%
    if (checkPercentage != 100 && wanderingAgentsNumber != 0)
    {// application is automatically quitted
        print("Check percentages, their sum is different from 100%");
        QuitApp();
    }
    totAgentsaux = totAgents;
    agentLister.Close();
}
```

LISTING 5: Here the GameController populates the environment with Special and wandering agents

```
// this function create a single log file
// that merge all the agents log file
private void QuitApp()
{
    //This if have to be uncommented if working on the editor
    // and commented if building the executable file
    /*if(Application.isEditor)
        EditorApplication.isPlaying = false; // stop the editor*/
    //Mergin all the Log file in a single one
    string[] outputs = Directory.GetFiles("./SimulationLogs");
    nLines = (int)Mathf.Floor(time / logInvokeTime);
    for (int i = 0; i < outputs.Length; i++)
    {
        File.Delete(outputs[i]);
    }
    StreamWriter writer =
        new StreamWriter("./SimulationLogs/Output.txt", true);
    writer.WriteLine("TotalNumberOfAgents: " + totAgents
        + " EndingTime: " + endingTime);
    // <= because we have one additional line for each
    // output file, that is the one of the initial position
    for (int i = 0; i <= nLines; i++)
    {
        // If the file is shorter than the others the program will
        // look for the line and will manage the exception doing nothing
        for (int j = 0; j < (int)totAgents; j++)
        {
            datas = File.ReadAllLines("./AgentsLog/Output" + j + ".txt");
            try
            {
                writer.WriteLine(datas[i]);
            }
            catch
```

```
            {}
        }
    }
    writer.Close();
    //stop the application when launched by executable file
    Application.Quit();
}
```

LISTING 6: Here is the quit function managed by the game controller. here the GameController merge all the agents output files in a single one and switch off the simulation

Appendix B: DigForReason code

Here we will show some pieces of code but the entire directory of DigForSim is available at this link:
https://github.com/Cibie/DigForReason

```
public boolean movementsInInterval(String fileName,
String person, float minTime, float maxTime) throws IOException {
        String[] lineElems;
        //array that will contain info about selected entity
        List<String> personInfo = new ArrayList<String>();
        // take the file on which DigForReason has to work
        BufferedReader br = GetOutputFile(fileName);
        answer = "";
        br.readLine();
        if(minTime <= maxTime){
                while((line = br.readLine()) != null) {
                        lineElems =  line.split(" ");
                    //Check if the line we are considering is good
                    // for our parameters (person and time interval)
                    if(lineElems[1].toString().equals(person) &&
                        Float.parseFloat(lineElems[lineElems.length
                            - 1].replace(",",".")) >= minTime &&
                        Float.parseFloat(lineElems[lineElems.length
                            - 1].replace(",",".")) <= maxTime){
                        //Checkinf if info are in line with data inserted
                        personInfo.clear();
                        // we use the same function used for meeting
                        personInfo = CreateMeetingInfo(lineElems);
                        answer = answer.concat("\n" + person
                            + " has been in " + personInfo.get(1) +
                                " at " + personInfo.get(4));
                    }
                }
                // if the person is not found it will return a warning
                if(personInfo.isEmpty()) {
                        answer = answer.concat(person + " is
                            not present in the simulation
                            or the time interval is not
                                represented in the simulation.
                            \nPlease check that Scene and
```

```
                                Population files are correctly
                            related and if not switch them");
                        return false;
                }
                return true;
        }
        else {//if inserted data are wrong it will return the error
                answer = answer.concat(minTime +
                   " is greater than " + maxTime);
                return false;
        }
}

public boolean PresenceInRoomInterval(String fileName,
String room, float minTime, float maxTime) throws IOException {
        String[] lineElems;
        //take the file on which DigForReason has to work
        BufferedReader br = GetOutputFile(fileName);
        answer = "";//empty the answer string
        br.readLine();
        if(minTime <= maxTime) {
                while((line = br.readLine()) != null) {
                        lineElems =  line.split(" ");
                        // check that info are in line with inserted data
                        if(lineElems[7].toString().equals(room) &&
                        Float.parseFloat(lineElems[lineElems.length
                        - 1].replace(",", ".")) >= minTime &&
                        Float.parseFloat(lineElems[lineElems.length
                         - 1].replace(",", ".")) <= maxTime) {
                        if(!answer.contains(lineElems[1])) {
                        answer = answer.concat(lineElems[1] + "\n");
                                }
                        }
                }
                if(answer.isEmpty())
                        answer = answer.concat("noone
                            has been in " + room);
                return true;
        }
        else {
                answer = answer.concat(minTime +
                    " is greater than " + maxTime);
                return false;
        }
}
```

LISTING 7: Here there are the movementsInInterval and the PresenceInInterval fucntions that can be called by the general function when the question si asked on all the simulation time

```
public void actionPerformed(ActionEvent e) {
        startTime =  System.currentTimeMillis();
        for(int i = 0; i < 1000; i++) {
        if(!minTimeQuest1.getText().isEmpty() &&
```

```
        !maxTimeQuest1.getText().isEmpty()) {
                try {
                        communicationPanel.setText("");
                        if(analyzer.Meeting(outFileString,
                        peopleList1Quest1.getSelectedItem().toString(),
                        peopleList2Quest1.getSelectedItem().toString(),
                        Float.parseFloat(minTimeQuest1.getText()),
                        Float.parseFloat(maxTimeQuest1.getText())) {
                                communicationPanel.append("\nWhere "+
                                peopleList1Quest1.getSelectedItem().toString()+
                                " and " +
                                peopleList2Quest1.getSelectedItem().toString()+
                                " have met in the interval "+
                                minTimeQuest1.getText() +
                                "/" + maxTimeQuest1.getText() +"? \r\n"  );
                                communicationPanel.append(analyzer.answer);
                        }
                        else
                                JOptionPane.showMessageDialog(null,
                                analyzer.answer, "InfoBox: " + "Error",
                                JOptionPane.INFORMATION_MESSAGE);
                } catch (IOException e1) {
                        JOptionPane.showMessageDialog(null,
                        outFileString + " does not exist anymore",
                        "InfoBox: " + "Error", JOptionPane.INFORMATION_MESSAGE);
                        e1.printStackTrace();
                }
                catch (NumberFormatException e2) {
                        JOptionPane.showMessageDialog(null,
                        "Insert numbers, not symbols or letters",
                        "InfoBox: " + "Error",
                        JOptionPane.INFORMATION_MESSAGE);
                }
        }
        else
                JOptionPane.showMessageDialog(null,
                "Inserisci dute estremi temporali",
                "InfoBox: " + "Error", JOptionPane.INFORMATION_MESSAGE);
        WriteOnFile(analyzer.currentFile);
        }
        endTime = System.currentTimeMillis();
        System.out.println(endTime - startTime);
}
```

LISTING 8: Here is the code of the function called pressing a button, check is always the same, in fact functions are all boolean

```
public boolean Meeting(String fileName, String person1,
String person2, float minTime, float maxTime) throws IOException{
        String[] lineElems;
        //List of info for Person 1
        List<String> person1Info =
            new ArrayList<String>();
        //list of info for Person 2
        List<String> person2Info =
```

```
        new ArrayList<String>();
    answer = "";
    //If inserted data are correct it goes one, otherwise
    // it will return to check inserted data
    if(minTime <= maxTime && !person1.equals(person2)) {
    BufferedReader br = GetOutputFile(fileName);
    br.readLine();
    //unitl it reaches the end of the file
    while((line = br.readLine()) != null){
        lineElems =  line.split(" ");
        //Check if the line we are considering its
        // good for our parameters (people and time interval)
        if(lineElems[1].toString().equals(person1) &&
            Float.parseFloat(lineElems[lineElems.length
            - 1].replace(",",".")) >= minTime &&
                    Float.parseFloat(lineElems[lineElems.length
                    - 1].replace(",",".")) <= maxTime) {
        // If info in the current line are in the range
        // of our parameters it will modify the list of info
            person1Info.clear();
            person1Info = CreateMeetingInfo(lineElems);
        }
        // as for person1
        else if(lineElems[1].toString().equals(person2) &&
            Float.parseFloat(lineElems[lineElems.length
            - 1].replace(",",".")) >= minTime &&
                    Float.parseFloat(lineElems[lineElems.length
                    - 1].replace(",",".")) <= maxTime) {
            person2Info.clear();
            person2Info = CreateMeetingInfo(lineElems);
        }
        if(!person1Info.isEmpty() && !person2Info.isEmpty() &&
            person1Info.get(4).equals(person2Info.get(4)) &&
                    person1Info.get(1).equals(person2Info.get(1)) &&
                    inRange(person1Info.get(2), person2Info.get(2)) &&
                    inRange(person1Info.get(3), person2Info.get(3))) {
            // If info TimeStamp and Place of person1 and person2
            // match it will return that they have met the current
            // place at the current timestamp
            if(!answer.contains("\n" + person1Info.get(0) + " and " +
                person2Info.get(0) + " have met in " + person1Info.get(1) +
                " at " + person1Info.get(4))){
                    //it does not insert duplicated string in the answer
                    answer = answer.concat("\n" + person1Info.get(0) +
                        " and " + person2Info.get(0) + " have met in " +
                        person1Info.get(1) + " at " + person1Info.get(4));
            }
        }
    }
    br.close();
    //If one of the two agents does not match inserted parameter
    //the program will return to check inserted data
    if(person1Info.isEmpty() || person2Info.isEmpty()){
            answer = answer.concat("One of the two entities that you have
            chosen does not exist in the simulation or in that time interval"
```

```
                        + " is not represented in the simulation.
                        \nPlease check that Scene and Population file are
                        correctly related and if not switch them");
                return false;
        }
        // if no answer is found it will return that these
        // entities have never met
        if(answer.isEmpty())
                answer = answer.concat(person1Info.get(0) +
                " and " + person2Info.get(0) + " have never met");
        return true;
        }
        else{
                answer = answer.concat(minTime + " is greater than " +
                maxTime + " or you have chosen twice the same person");
                return false;
        }
}
```

Listing 9: Here we have the function that checks if two people have met

Bibliography

[1] DFRWS. A road map for digital forensic research (report from the first digital forensic research workshop, dfrws), 2001. URL http://www.dfrws.org/sites/default/files/session-files/a_road_map_for_digital_forensic_research.pdf. Collective work of all DFRWS attendees.

[2] VirTra. Virtualtraining, 2018. URL https://www.virtra.com/overview-le.

[3] Steven Warburton. Second life in higher education: Assessing the potential for and the barriers to deploying virtual worlds in learning and teaching. *British journal of educational technology*, 40(3):414–426, 2009.

[4] Suzanne C. Baker, Ryan K. Wentz, and Madison M. Woods. Using virtual worlds in education: Second life® as an educational tool. *Teaching of Psychology*, 36(1):59–64, 2009. doi: 10.1080/00986280802529079. URL https://www.tandfonline.com/doi/abs/10.1080/00986280802529079.

[5] Jonathan Crellin and Sevasti Karatzouni. Simulation in digital forensic education. In *Third International Conference on Cybercrime Forensic Education and Training (CFET3)(BCS SIG)*, 2009.

[6] Digital Forensics and Incident Response Lab. Dfirlabs, 2014-2017. URL https://www.dfirlabs.com/.

[7] Brian D. Carrier. Defining digital forensic examination and analysis tool using abstraction layers. *IJDE*, 1(4), 2003. URL http://www.utica.edu/academic/institutes/ecii/publications/articles/A04C3F91-AFBB-FC13-4A2E0F13203BA980.pdf.

[8] Mark Reith, Clint Carr, and Gregg H. Gunsch. An examination of digital forensic models. *IJDE*, 1(3), 2002. URL http:

//www.utica.edu/academic/institutes/ecii/publications/articles/A04A40DC-A6F6-F2C1-98F94F16AF57232D.pdf.

[9] Eoghan Casey. *Handbook of Digital Forensics and Investigation*. Academic Press, Inc., Orlando, FL, USA, 2009. ISBN 0123742676, 9780123742674.

[10] Scientific Working Group on Digital Evidence. Swgde, 2015. URL https://www.swgde.org/.

[11] National Institute of Justice. Nij, 2019. URL https://nij.ojp.gov/digital-evidence-and-forensics.

[12] American Bar Association. Digitalforinfo, 2006. URL https://www.lawtechnologytoday.org/2018/05/digital-forensics/.

[13] Daniele Theseider Dupré Esra Erdem and Martìn Dìeguez. Wg5: Platform integration and multi-dimensional environments, 2018.

[14] Brian Carrier. Autopsy and the sluthkit, 2003-2019. URL https://www.sleuthkit.org/autopsy/.

[15] SANS Institute. Sift, 2008 - 2019. URL https://digital-forensics.sans.org/community/downloads#overview.

[16] Oxygen Forensics Inc. Ofs, 2018-2019. URL https://www.oxygen-forensic.com/en/.

[17] Cellebrite. Cel, 2019. URL https://www.cellebrite.com/en/about/.

[18] Cellebrite. Celufed, 2019. URL https://www.cellebrite.com/en/ufed-ultimate/.

[19] CrowdStrike: Breaches stop here. Crowdstrike, 2019. URL https://www.crowdstrike.com/.

[20] The Volatility Foundation. Volatility, 2013. URL https://www.volatilityfoundation.org/releases.

[21] AccessData. Ftk imager, 1994. URL https://marketing.accessdata.com/ftkimager4.2.0.

[22] Hyacinth S. Nwana. Software agents: an overview. *Knowledge Eng. Review*, 11(3):205–244, 1996. doi: 10.1017/S026988890000789X. URL https://doi.org/10.1017/S026988890000789X.

[23] W. Schermer. *Software Agents, Surveillance, and the Right to Privacy: A Legislative Framework for Agent-Enabled Surveillance*. LUP Meijersreeks Series. Leiden University Press, 2007. ISBN 9789087280215. URL https://books.google.it/books?id=q1Li7-W7S4MC.

[24] James Ingham. What is an Agent? Technical report, University of Durham, Center for Software Maintenance, 06 1999.

[25] Yoann Kubera, Philippe Mathieu, and Sébastien Picault. Everything can be agent! In *9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010), Toronto, Canada, May 10-14, 2010, Volume 1-3*, pages 1547–1548, 2010. URL https://dl.acm.org/citation.cfm?id=1838474.

[26] Michael J. Wooldridge. *Introduction to multiagent systems*. Wiley, 2002. ISBN 978-0-471-49691-5.

[27] J. L. Austin. *How to Do Things with Words*. Clarendon Press, 1962.

[28] Rodger Kibble. Speech acts, commitment and multi-agent communication. *Computational & Mathematical Organization Theory*, 12(2-3):127–145, 2006. doi: 10.1007/s10588-006-9540-z. URL https://doi.org/10.1007/s10588-006-9540-z.

[29] Mark Torrance and Paul Viola. *The AGENT0 manual*. Stanford University, Department of Computer Science, 1991.

[30] Hamed Rezaee and Farzaneh Abdollahi. Average consensus over high-order multiagent systems. *IEEE Trans. Automat. Contr.*, 60(11):3047–3052, 2015. doi: 10.1109/TAC.2015.2408576. URL https://doi.org/10.1109/TAC.2015.2408576.

[31] Long Ma, Haibo Min, Shicheng Wang, Yuan Liu, and Shouyi Liao. An overview of research in distributed attitude coordination control. *IEEE/CAA Journal of Automatica Sinica*, 2(2):121–133, 2015.

[32] Gregory Dudek, Michael R. M. Jenkin, Evangelos E. Milios, and David Wilkes. A taxonomy for multi-agent robotics. *Auton. Robots*, 3(4):375–397, 1996. doi: 10.1007/BF00240651. URL https://doi.org/10.1007/BF00240651.

[33] Manal Khayyat and Anjali Awasthi. An intelligent multi-agent based model for collaborative logistics systems. *Transportation Research Procedia*, 12:325–338, 2016.

[34] Charles M Macal and Michael J North. Agent-based modeling and simulation: Desktop abms. In *2007 Winter Simulation Conference*, pages 95–106. IEEE, 2007.

[35] John H Holland. Complex adaptive systems. *Daedalus*, 121(1):17–30, 1992.

[36] Paulo Leitão and Stamatis Karnouskos. *Industrial Agents: Emerging Applications of Software Agents in Industry*. Morgan Kaufmann, 2015.

[37] Massive Software. Masinfilm, 2002. URL http://www.massivesoftware.com/film.html.

[38] P. Leitão, S. Karnouskos, L. Ribeiro, J. Lee, T. Strasser, and A. W. Colombo. Smart agents in industrial cyber–physical systems. *Proceedings of the IEEE*, 104(5):1086–1101, May 2016. ISSN 0018-9219. doi: 10.1109/JPROC.2016.2521931.

[39] Xiao-Feng Xie, Stephen F Smith, and Gregory J Barlow. Schedule-driven coordination for real-time traffic network control. In *Twenty-Second International Conference on Automated Planning and Scheduling*, 2012.

[40] Tamás Máhr, Jordan Srour, Mathijs de Weerdt, and Rob Zuidwijk. Can agents measure up? a comparative study of an agent-based and on-line optimization approach for a drayage problem with uncertainty. *Transportation Research Part C: Emerging Technologies*, 18(1):99–119, 2010.

[41] Nicole Lang Beebe and Jan Guynes Clark. A hierarchical, objectives-based framework for the digital investigations process. *Digital Investigation*, 2(2):147–167, 2005.

[42] JADE Team. Jade, 2019. URL https://jade.tilab.com/.

[43] The AnyLogic Company. Anylogic, 2002. URL https://www.anylogic.com/use-of-simulation/agent-based-modeling/.

[44] Marco Pruckner, David Eckhoff, and Reinhard German. Modeling country-scale electricity demand profiles. In *Proceedings of the Winter Simulation Conference 2014*, pages 1084–1095. IEEE, 2014.

[45] L. M. S. Dias, A. A. C. Vieira, G. A. B. Pereira, and J. A. Oliveira. Discrete simulation software ranking — a top list of the worldwide most popular and used tools. In *2016 Winter Simulation Conference (WSC)*, pages 1060–1071, Dec 2016. doi: 10.1109/WSC.2016.7822165.

[46] Jingsi Huang, Lingyan Liu, and Leyuan Shi. Auction policy analysis: An agent-based simulation optimization model of grain market. *2016 Winter Simulation Conference (WSC)*, pages 3417–3428, 2016.

[47] C. W. Weimer, J. O. Miller, and R. R. Hill. Agent-based modeling: An introduction and primer. In *2016 Winter Simulation Conference (WSC)*, pages 65–79, Dec 2016. doi: 10.1109/WSC.2016.7822080.

[48] Jonathan Ozik, Nicholson T. Collier, John T. Murphy, and Michael J. North. The relogo agent-based modeling language. In *Winter Simulations Conference: Simulation Making Decisions in a Complex World, WSC 2013, Washington, DC, USA, December 8-11, 2013*, pages 1560–1568, 2013. doi: 10.1109/WSC.2013.6721539. URL https://doi.org/10.1109/WSC.2013.6721539.

[49] David Camacho, Ricardo Aler, César Castro, and José M Molina. Performance evaluation of zeus, jade, and skeletonagent frameworks. In *IEEE International Conference on Systems, Man and Cybernetics*, volume 4, pages 6–pp. IEEE, 2002.

[50] Hyacinth Nwana, Divine T. Ndumu, Lyndon C. Lee, and Martlesham Heath. Zeus: An advanced tool-kit for engineering distributed multi-agent systems, 1998.

[51] Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, Keith Sullivan, and Gabriel Balan. Mason: A multiagent simulation environment. *Simulation*, 81(7): 517–527, 2005.

[52] Uri Wilensky. Netlogo, 1999-2016. URL http://ccl.northwestern.edu/netlogo/index.shtml.

[53] Patrick Taillandier, Benoit Gaudou, Arnaud Grignard, Nghi Quang Huynh, Nicolas Marilleau, Philippe Caillou, Damien Philippon, and Alexis Drogoul. Building, composing and experimenting complex spatial models with the GAMA platform. *GeoInformatica*, 23(2):299–322, 2019. doi: 10.1007/s10707-018-00339-6. URL https://doi.org/10.1007/s10707-018-00339-6.

[54] Patrick Taillandier, Mathieu Bourgais, Philippe Caillou, Carole Adam, and Benoit Gaudou. A BDI agent architecture for the GAMA modeling and simulation platform. In *Multi-Agent Based Simulation XVII - International Workshop, MABS 2016, Singapore, May 10, 2016, Revised Selected Papers*, pages 3–23, 2016. doi: 10.1007/978-3-319-67477-3_1. URL https://doi.org/10.1007/978-3-319-67477-3_1.

[55] JSON Developer. Json, 2017. URL http://jason.sourceforge.net/wp/description/.

[56] Alexander Pokahr, Lars Braubach, and Winfried Lamersdorf. Jadex: A BDI reasoning engine. In *Multi-Agent Programming: Languages, Platforms and Applications*, pages 149–174. Springer, Boston, MA, 2005.

[57] V Mastronardi and M Dellisanti Fabiano Vilardi. Ricostruzione della scena del crimine in 3d. *Mondo Digitale*, page 2, 2014.

[58] Autodesk Inc. 3dmax, 2019. URL https://www.autodesk.com/products/3ds-max/overview.

[59] ContentEngine. Poser pro, 2018. URL http://poser.smithmicro.com.

[60] Robert McNeel and Associates. Rhino, 2019. URL http://www.rhino3d.com.

[61] Francesco Siddi Pablo Vazquez. Blender, 2000. URL https://www.blender.org/about/.

[62] David Lillis, Brett Becker, Tadhg O'Sullivan, and Mark Scanlon. Current challenges and future research areas for digital forensic investigation. *arXiv preprint arXiv:1604.03850*, 2016.

[63] Sriram Raghavan. Digital forensic research: current state of the art. *CSI Transactions on ICT*, 1(1):91–114, Mar 2013. ISSN 2277-9086. doi: 10.1007/s40012-012-0008-7. URL https://doi.org/10.1007/s40012-012-0008-7.

[64] Jayavardhana Gubbi, Rajkumar Buyya, Slaven Marusic, and Marimuthu Palaniswami. Internet of things (iot): A vision, architectural elements, and future directions. *Future generation computer systems*, 29(7):1645–1660, 2013.

[65] Luigi Atzori, Antonio Iera, and Giacomo Morabito. The internet of things: A survey. *Computer networks*, 54(15):2787–2805, 2010.

[66] Elizabeth R. Groff, Shane D. Johnson, and Amy Thornton. State of the art in agent-based modeling of urban crime: An overview. *Journal of Quantitative Criminology*, 35(1):155–193, Mar 2019. ISSN 1573-7799. doi: 10.1007/s10940-018-9376-y. URL https://doi.org/10.1007/s10940-018-9376-y.

[67] Tibor Bosse, Charlotte Gerritsen, and Jan Treur. Agent-based simulation of episodic criminal behaviour\m{1}. *Multiagent and Grid Systems*, 9(4):315–334, 2013. doi: 10.3233/MGS-130211. URL https://doi.org/10.3233/MGS-130211.

[68] Tibor Bosse, Charlotte Gerritsen, and Jan Treur. Grounding a cognitive modelling approach for criminal behaviour. In *Proceedings of the second European cognitive science conference, EuroCogSci*, volume 7, pages 776–781, 2007.

[69] Lawrence E Cohen and Marcus Felson. Social change and crime rate trends: A routine activity approach. *American sociological review*, pages 588–608, 1979.

[70] C Gerritsen. Caught in the act: investigating crime by agent-based simulation. 2010.

[71] Tibor Bosse and Charlotte Gerritsen. Agent-based simulation of the spatial dynamics of crime: on the interplay between criminal hot spots and reputation. In *7th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2008), Estoril, Portugal, May 12-16, 2008, Volume 2*, pages 1129–1136, 2008. URL https://dl.acm.org/citation.cfm?id=1402378.

[72] Tibor Bosse, Charlotte Gerritsen, and Jan Treur. Cognitive and social simulation of criminal behaviour: the intermittent explosive disorder case. In *6th International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS 2007), Honolulu, Hawaii, USA, May 14-18, 2007*, page 58, 2007. doi: 10.1145/1329125.1329195. URL https://doi.org/10.1145/1329125.1329195.

[73] Tibor Bosse, Charlotte Gerritsen, Michel C. A. Klein, and Frank M. Weerman. Development and validation of an agent-based simulation model of juvenile delinquency. In *Proceedings of the 12th IEEE International Conference on Computational Science and Engineering, CSE 2009, Vancouver, BC, Canada,*

August 29-31, 2009, pages 200–207, 2009. doi: 10.1109/CSE.2009.136. URL https://doi.org/10.1109/CSE.2009.136.

[74] Knud Henriksen, Jon Sporring, and Kasper Hornbæk. Virtual trackballs revisited. *IEEE Transactions on Visualization and Computer Graphics*, 10 (2):206–216, 2004.

[75] Steven L Lytinen and Steven F Railsback. The evolution of agent-based simulation platforms: a review of netlogo 5.0 and relogo. In *Proceedings of the fourth international symposium on agent-based modeling and simulation*, page 19, 2012.

[76] Esri shapefile technical description. Technical report, Environmental Systems Research Institute, 07 2007.