POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Design and Implementation of a Graphical Interface for NeeMAS: A Multi-Agent Behavioral Simulator Driven by Needs

Tesi di Laurea Magistrale in
Computer Science and Engineering

Author: **Marco Luca Previtera**

Student ID: 233054
Advisor: Prof. Sara Comai
Co-advisors: Prof. Fabio Salice
Academic Year: 2025-2026

Abstract

With the global population aging, healthcare assistance for elderly people is becoming increasingly important. Information technologies can support healthcare by monitoring activities of daily living (ADLs) and providing useful insights into behavioural patterns. However, collecting and analysing large amounts of human behavioural data remains an expensive and time-consuming task.

This thesis presents the development of a graphical user interface for NeeMAS (Needs-based Multi-Agent Simulator), an agent-based simulator designed to reproduce human behaviour in virtual environments using digital twins. The simulator models physiological needs and social interactions through a set of configurable parameters that drive agent decision-making and movement within a virtual environment. The graphical interface was designed and implemented to simplify the environment creation, parameter configuration, simulation execution and result analysis.

To generate realistic agent profiles, a questionnaire targeting elderly individuals was used. The resulting profiles were then employed to configure and validate the simulator through individual, social and wandering behavior scenarios, demonstrating the effectiveness of both the parameter generation workflow and the graphical interface.

The developed framework provides a flexible tool for behavioural simulation and represents a step towards the creation of realistic digital twins for assisted living environments and behavioural analysis applications.

Keywords: Behavioural simulator, graphical user interface, simulation of activities of daily living, agent-based simulator

Abstract in lingua italiana

Con l'invecchiamento della popolazione mondiale, l'assistenza sanitaria per le persone anziane sta assumendo un'importanza sempre crescente. Le tecnologie dell'informazione possono supportare il settore sanitario attraverso il monitoraggio delle attività della vita quotidiana (Activities of Daily Living, ADL) e la fornitura di informazioni utili sui modelli comportamentali. Tuttavia, la raccolta e l'analisi di grandi quantità di dati comportamentali umani rimangono attività costose e dispendiose in termini di tempo.

Questa tesi presenta lo sviluppo di un'interfaccia grafica per NeeMAS (Needs-based Multi-Agent Simulator), un simulatore basato su agenti progettato per riprodurre il comportamento umano in ambienti virtuali mediante digital twin. Il simulatore modella i bisogni fisiologici e le interazioni sociali attraverso un insieme di parametri configurabili che guidano le decisioni degli agenti e i loro movimenti all'interno dell'ambiente virtuale. L'interfaccia grafica è stata progettata e implementata per semplificare la creazione dell'ambiente, la configurazione dei parametri, l'esecuzione delle simulazioni e l'analisi dei risultati.

Per generare profili realistici degli agenti è stato realizzato un questionario rivolto alle persone anziane. I profili ottenuti sono stati successivamente impiegati per configurare e validare il simulatore attraverso scenari individuali, sociali e di wandering, dimostrando l'efficacia sia del processo di generazione dei parametri sia dell'interfaccia grafica.

Il framework sviluppato fornisce uno strumento flessibile per la simulazione comportamentale e rappresenta un passo verso la creazione di digital twin realistici per scenari di vita assistita e applicazioni di analisi comportamentale.

Parole chiave: Simulatore comportamentale, interfaccia grafica, simulazione delle attività della vita quotidiana, simulatore multi-agente

Contents

1 | Introduction

1.1. Population Ageing and Healthcare Challenges

The significant improvements in living conditions, sanitation, nutrition and medical care that occurred during the last century have led to a remarkable increase in human life expectancy. According to the World Health Organization (WHO), global life expectancy reached approximately 73.3 years in 2024, increasing by more than eight years since the mid-1990s and almost doubling compared to the beginning of the twentieth century [27].

While increased longevity is one of the greatest achievements of modern society, it also introduces new demographic challenges. According to the United Nations, the number of people aged 60 and over is expected to grow from about 1 billion in 2020 to 1.4 billion by 2030 and over 2 billion by 2050 [25].

This demographic shift is expected to place considerable pressure on healthcare systems and long-term care services, as older adults often experience chronic conditions, functional limitations and increased healthcare needs [26]. For these reasons, technological solutions such as assistive technologies, smart environments and digital health systems are increasingly being proposed to support independent living and improve the quality of life of the ageing population.

1.2. Behavioural Analysis and Activities of Daily Living

In the daily lives of older adults, their actions and behaviours can provide important indicators of health and functional status. In general, behavioural patterns in elderly individuals tend to remain relatively stable over time and significant deviations from these patterns may indicate changes in health conditions. For this reason, analysing behavioural drifts can be useful for identifying early signs of cognitive impairment or neurodegenerative diseases such as Alzheimer's disease or dementia [8].

Human daily behaviour is commonly described through two categories of activities: Activities of Daily Living (ADLs) [12] and Instrumental Activities of Daily Living (IADLs) [14].

- Activities of Daily Living (ADLs) represent essential self-care tasks that individuals perform daily to maintain independence and personal hygiene, such as bathing, dressing, eating and sleeping.
- Instrumental Activities of Daily Living (IADLs), on the other hand, include more complex activities required to live independently within a community, such as cooking, cleaning, shopping, or managing finances.

Monitoring these activities through smart environments and IoT-based sensing systems enables the automatic recognition of human activities and the analysis of behavioural patterns over time. Such approaches are increasingly used in smart home environments to support healthcare monitoring and assistive services for older adults [2].

1.3. Digital Twins for Behaviour Simulation and Data Generation

Collecting data for behavioral analysis is a delicate and complex task. In fact, due to the large variability in human behaviours, a large amount of data is needed to adequately represent the different situations that may occur in real environments. However, collecting large-scale behavioural datasets from real-world environments using sensors can be time-consuming, expensive and often limited by privacy and logistical constraints [11].

A possible alternative to obtain the required amount of data in a short time is to simulate a virtual living environment [20]. In particular, digital twin technologies allow the creation of virtual replicas of real users, allowing researchers to simulate behaviours in a controlled and scalable way [7].

In this work, the NeeMAS simulator [5] is used to emulate the behaviours and interactions of a group of individuals within a closed environment. The simulation models the evolution of primary physiological needs, such as eating, sleeping and using the bathroom, as well as social needs, which influence the priorities of the simulated agents over time. Based on the evolution of these needs, agents move within the environment and perform activities to satisfy them.

Each need is regulated by configurable parameters that define when and where it can be satisfied. For example, eating activities can be restricted to specific time intervals

during the day, while other needs, such as going to the bathroom, may vary in frequency depending on temporal factors.

Another important component of the digital twin concerns social interactions among agents. Individuals can establish relationships and interact with each other based on compatibility and friendship parameters initialized at the beginning of the simulation. These relationships evolve dynamically during the simulation: friendship levels increase when agents interact positively and decrease otherwise, influencing the likelihood of future interactions and the tendency of individuals to seek alternative social contacts.

1.4. Contributions of the Thesis

The main contributions of this work can be summarized as follows:

- *Design and development of a configurable interface* that allows users to easily define and modify simulation parameters, enabling the creation of different behavioural scenarios without requiring direct modifications to the simulation code.
- *Definition of representative personas* describing typical profiles of older adults. These personas are derived from common behavioural patterns and habits collected through questionnaires.
- *Validation of the proposed interface* through the generation and analysis of simulation scenarios corresponding to the identified personas. The validation assesses whether the interface is capable of producing coherent simulation parameters and reproducing the expected behavioural patterns associated with each persona.

1.5. Structure of the Thesis

The thesis is structured as follows:

- Chapter 2 reviews the related work and provides an overview of the challenges associated with collecting behavioral data. It also discusses the main approaches proposed in the literature for human behavior simulation, multi-agent systems and digital twins.
- Chapter 3 presents the methodology underlying this work. It introduces the NeeMAS simulator, describes its theoretical foundations and explains the models used to represent needs, social interactions, wandering behavior and the environment. The chapter also details the simulator configuration process, the methods used to gener-

ate simulation parameters and the methodology that was used to design, distribute and analyze the questionnaire.

- Chapter 4 describes the design and implementation of the proposed graphical user interface. It presents the system architecture, the main software components and the functionalities developed to support the creation and management of simulation scenarios.
- Chapter 5 presents the experimental evaluation of the proposed approach. It describes the identified personas, the generated simulation scenarios and the results obtained through the interface. The chapter also discusses whether the resulting simulations are consistent with the expected behavioral profiles.
- Finally, Chapter 6 concludes the thesis by summarizing the main results of this work and outlining possible directions for future research.

2 | Related Work

In the context of behavioral analysis, a review of the existing literature reveals different approaches for obtaining the amount of data required for model development and validation. In general, the data used can be divided into two main types: real-world data, collected with various sensors from environments such as smart homes and simulated data, generated through computational models designed to reproduce realistic behavioral patterns.

2.1. Real World Data: Smart Homes

Smart homes can be defined as residential environments equipped with digitally connected technologies and automation systems that provide enhanced services to occupants [22]. These systems integrate sensors, communication infrastructures and intelligent devices to monitor, control and optimize different aspects of the home environment, often with limited or no direct user intervention. The term is frequently associated with home automation, as it includes technologies capable of automating everyday tasks and adapting to users' needs and behaviors [22].

Smart home technologies is a rapidly growing market [13] and have gained increasing attention due to their potential impact on several domains, including energy efficiency, healthcare, assisted living and sustainability.
In a smart home environment, the data generated by connected devices and sensors can be analyzed either in real time or offline to improve the performance of the systems and take valuable overviews of the systems. Real-time analysis enables the fast identification and management of scheduled events, allowing the system to react dynamically to changing conditions. In contrast, offline analysis exploits historical data to identify patterns and develop algorithms aimed at improving the overall functionality of the smart home. However, the design and validation of such algorithms require the availability of large datasets. Examples of datasets derived from real smart homes installations are CASAS [6] and TigerPlace [21].

2.2. Digital Life Simulation

Digital life simulation refers to the modelling and reproduction of human behaviours and interactions within virtual environments in order to generate synthetic data representing activities performed in real-world settings. In recent years, these approaches have increasingly been used to model daily behaviors and activities to support activity recognition and behavioral analysis in smart environments [4, 5].

Although real environments provide highly realistic data, they require the deployment and maintenance of instrumented homes, which can be costly and time-consuming [18]. Simulation tools, instead, reproduce human–environment interactions in virtual settings without requiring physical infrastructures or human participants. The synthetic datasets generated through simulation can be used to train and evaluate algorithms for activity recognition.
Moreover, digital life simulations can support the development of assistive technologies, such as systems designed to support individuals with cognitive impairments or dementia, by modelling human behaviours within smart home environments [17]. Simulation-based approaches enable also the creation of controlled scenarios in which behavioural patterns and their variations can be analysed, facilitating the study of behavioural drifts and cognitive decline in older adults [5].

Simulation tools, as can be seen in Figure 2.1 are categorized into four groups: model-based simulations, interactive simulations, data-driven simulations and hybrid simulations [24].

- *Model-based simulations* rely on predefined statistical or probabilistic models to generate synthetic activity data efficiently, which enables the creation of large datasets, although often at the expense of behavioural realism. They can be divided in need-based simulators [4, 16] and context-based simulators [3, 9, 19].
- *Interactive simulations* use avatars controlled by real users moving and acting within a virtual environment to reproduce human behaviour and interactions in real time, producing more detailed and behaviourally realistic data. In contrast to other simulation paradigms, the creation of large datasets is generally slower and more expensive due to the need for continuous human interaction. Interactive simulators are based on a *human-in-the-loop* paradigm, where users are not passive observers but active agents whose actions directly influence the simulation state. This tight coupling between user and system enables more realistic behavioural dynamics. It also supports the study of socio-technical systems and adaptive behaviour under

changing conditions. Examples of interactive simulations frameworks are IEsim [23] and OpenSHS [1].

- *Data-driven simulations* use real-world datasets to generate new behavior patterns, often supported by machine learning methods; while they can better reflect complex human dynamics, their performance strongly depends on the availability and quality of the underlying data. An example of data-driven simulator is MoSen [28].
- *Hybrid simulations* combine two or more simulation techniques, typically integrating model-based approaches with interactive simulation paradigms. In practice, hybrid simulators often use scripted or algorithmic components to generate large portions of the data, while relying on user-driven or agent-based interaction to capture more complex dynamics. An example of hybrid approach is OpenSHS simulator [1] which combines automatic data generation with direct user interaction.

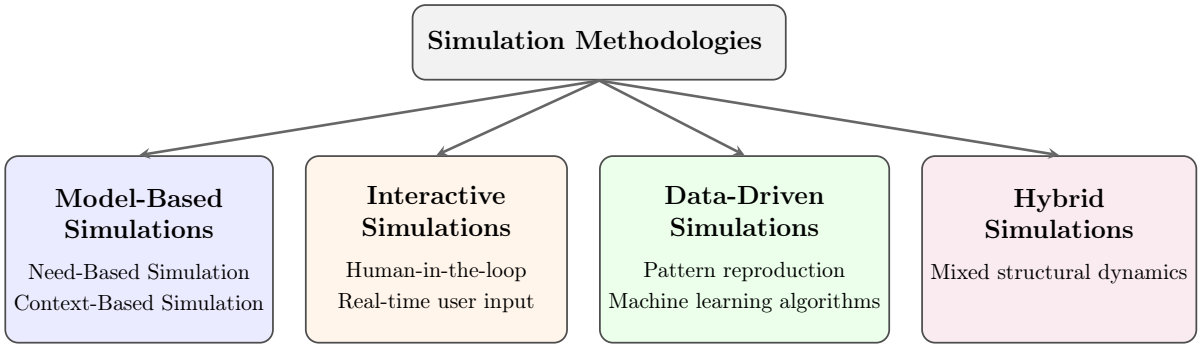

Figure 2.1: Classification hierarchy of simulation methodologies.

2.2.1. Model-based Simulation

Model-based simulation approaches generate synthetic behavioral data through predefined rules, probabilistic formulations, or statistical models describing human activities. These methods typically represent user behavior through activity schedules, transition probabilities or temporal patterns derived from empirical observations. Their main advantage lies in their computational efficiency, as they can generate large amounts of data with relatively low computational cost.

A common approach in behavioral simulation consists of defining activity models through probabilistic transitions between states or routines, often using techniques such as Markov chains, temporal rules, or activity distributions. Such methods allow the generation of repetitive and structured daily patterns while maintaining a certain degree of variability. However, because behaviors are derived from simplified representations, these approaches

often struggle to capture the complexity and unpredictability of real human behavior. Social interactions, contextual adaptations and spontaneous decisions are generally difficult to model accurately, which may reduce the realism of the generated datasets.

Within the model-based category, simulation approaches can be further distinguished into *need-based* and *context-based* paradigms, which differ primarily in the mechanisms that drive agent behavior and in the representation of internal state. In need-based simulators, agents are governed by internal variables (e.g., physiological or psychological needs) that evolve over time and directly determine actions when specific thresholds are reached; behavior is therefore largely endogenous, emerging from the continuous competition between multiple internal needs. In contrast, context-based simulators emphasize external environmental conditions as the main driver of behavior, where agents react to factors such as location, events, or the presence of other entities.
As a result, need-based approaches are better suited to capturing long-term, goal-driven behavior and persistence (e.g., hunger or fatigue accumulation), whereas context-based approaches are more effective for modeling immediate, reactive decisions triggered by the surrounding environment.

Need-based Simulation Frameworks

For the scope of this thesis, we will focus on need-based simulators. In particular, the proposed NeeMAS simulator [5] belongs to the need-based category, as its behaviour is primarily defined through predefined rules and structured representations without any real time user driven interactions. The NeeMAS simulator adopts a need-driven paradigm, where the behaviour of each agent is determined by its internal daily needs, such as hunger or sleepiness. These needs evolve over time and are the main drivers of the selection and execution of activities within the simulation.

Another example of need-based simulator framework is *SiSHoDit* [4], a framework for creating digital twins of smart environments and generating synthetic ADL datasets. Built on the Godot game engine, it provides a 3D visual editor that allows users to design environments and configure virtual sensors through a graphical interface. Agent behaviour is driven by a needs-based model, where actions emerge from the evolution of internal needs and probabilistic parameters. The simulator generates both activity and sensor data while incorporating environmental uncertainty. The resulting datasets are stored in JSON format for interoperability and further analysis.

The simulator proposed by *Lee et al.* [16], is another need based simulator, designed as a simulation environment for ubiquitous computing and smart home environments. Built

using the Unity3D engine, it provides a 3D graphical interface that allows users to configure layouts, furniture and virtual sensors through drag-and-drop interactions. Simulation models and environment configurations can be stored and reloaded using dedicated files. Agent behaviour is generated by a deterministic, rule-based algorithm. Decisions are driven by a motivation hierarchy that selects the most urgent internal state, which is then mapped through a fixed motivation–action database into a predefined sequence of actions.

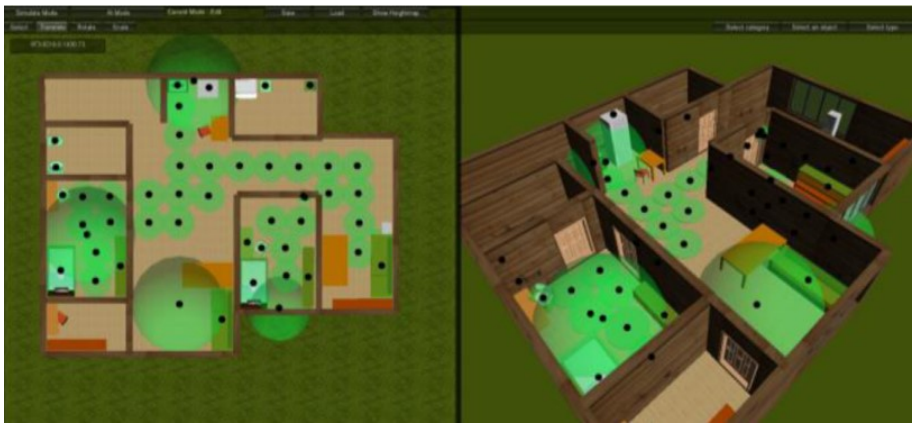

Figure 2.2: Screenshot from the Lee et al. simulator

A similar work is finally the simulator proposed by *Jiang and Mita* [10] which focuses specifically on modeling the daily routines of elderly residents living alone. The simulator does not provide a GUI and is configured directly using internal Python data structures. The system relies on an evolving need-driven model where an agent's internal needs dynamically changing over time, driving the probability of what activity sequence they will perform next. This approach addresses a common limitation in synthetic data generation, eliminating gaps in the timeline, producing continuous and realistic multi-day activity schedules. Finally, the generated schedules are fed into a location tracker that maps out the resident's physical paths, which is then translated into virtual passive infrared (PIR) sensor triggers.

Context-based Simulation Frameworks

An example of model-based simulator is *CASS* [19], a platform designed to generate realistic sensor-like data inside a virtual environment. It provides a graphical interface that allows users to monitor sensor activity and system states in real time. Users can configure the environment by placing virtual devices within a 2D visual layout, which is then saved in XML format. System behaviour is defined through rules that specify conditions and the actions triggered when those conditions are met.

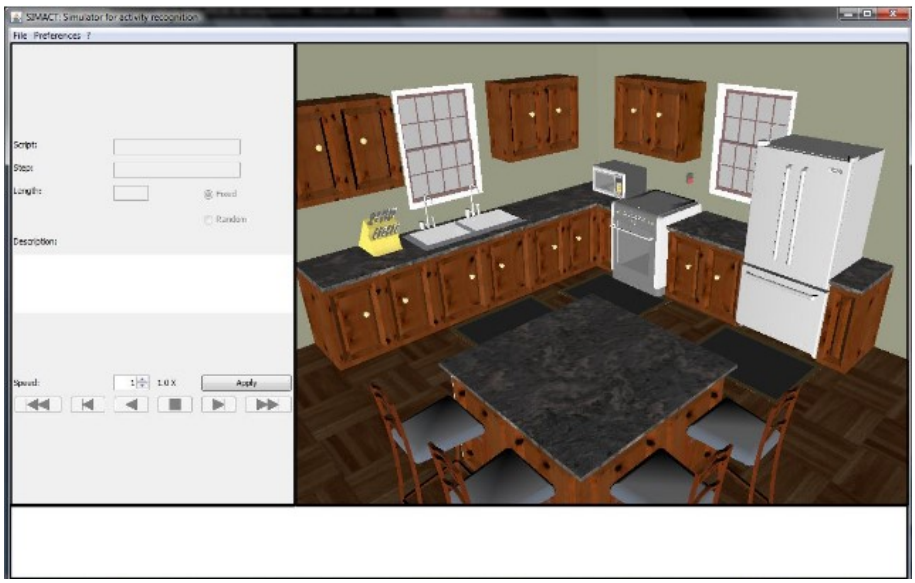

Figure 2.3: Screenshot of the SIMACT simulator

In *Persim 3D* [15], the system is structured into modular interface panels that support environment design, action and activity specification, context graph construction and simulation control within a Unity 3D based graphical interface. Project data are stored in a binary format enabling incremental editing, while simulation outputs are exported as CSV logs of sensor events and state changes.

Another simulator based on model is *SESim* [9], an open-source 3D simulation tool for generating large datasets used in Activity of Daily Living (ADL) recognition. Developed in Unity, it uses a specialized library to process activity sequences that links pre-implemented actions with high-level behaviour definitions written in simple text scripts. The system combines a graphical interface for environment design with the text-based scripts for configuring layouts, activities and simulation logic. Daily behaviours are defined through scripts specifying actions, timing and transitions between activities.

Lastly *SIMACT* [3] is a 3D open-source smart home simulator designed to validate activity recognition algorithms implementing behavioural scenarios derived from clinical studies. It provides a 3D visualiziation for user activities. The simulation scenario is configured through XML files, where user behaviours are specified through predefined activity sequences and probabilistic action patterns. This allow users to define environments, create objects and behavioural scenarios without modifying the core code. Other simulation settings can be adjusted through simple `.ini` files or directly via the graphical user interface.

Framework	Type	Interface	Behaviour Model	Multi Agent	I/O Format
SiSHoDit [4]	Need-based	3D GUI (Godot)	Dynamic needs, Probabilistic decision-making	No	JSON
Lee et al. [16]	Need-based	3D GUI (Unity3D)	Deterministic needs hierarchy, Rule-based action mapping	No	Binary
Jiang and Mita [10]	Need-based	CLI	Dynamic probability sampling via evolving motivation values, Layout-dependent activity filtering	No	Python objects, Text logs
CASS [19]	Context-based	2D GUI	Rule-based condition action system	Yes	XML
Persim 3D [15]	Context-based	3D GUI (Unity3D)	Context-driven, Rule-based simulation	No	Binary, CSV
SESim [9]	Context-based	3D GUI (Unity3D), script-based configuration	Script-driven activity sequences	No	Text scripts
SIMACT [3]	Context-based	3D GUI (Java)	Probabilistic activity scenarios	No	XML, .INI

Table 2.1: Summary of the main characteristics of the analyzed simulation frameworks.

3 | Methodology - The Digital Twin

Simulators that mimic human activities in closed environments offer a cost-effective alternative to equipping real-world spaces, which involve significant expenses and the need to engage human participants for data generation. Our approach features NeeMAS (NEEd-based Multi-Agent Simulator), a simulator created at the Politecnico di Milano. NeeMAS replicates the actions of one or more people and their interactions, driven not just by physiological needs but also by social needs. The output of the simulation is the graphical representation of the evolution of the needs of the person and its movements over time in the defined map.

3.1. Overview of the Simulator

NeeMAS is a model-based simulation framework designed to reproduce human behaviour in closed environments through the interaction of autonomous agents. As can be seen in Figure 3.1 the architecture follows a modular design composed of five main components: the simulation controller, the person model, the needs manager, the environment model and the social interaction module.

At the core of NeeMAS, the *Simulation controller* is responsible for initializing the environment, loading configuration data, creating agents and advancing the simulation clock. Human behaviour is represented through autonomous *Person* agents that are continuously calculating their internal state and make decisions based on the urgency of their needs.

The behavior of each agent is driven by the *Needs manager*, which manages physiological, social and wandering needs and determines the actions required to satisfy them. These actions are performed within an environment modelled as a network of interconnected places managed by the *Map manager*, finally social interactions are regulated by the *Sociality manager* through compatibility and friendship relationships. Throughout the

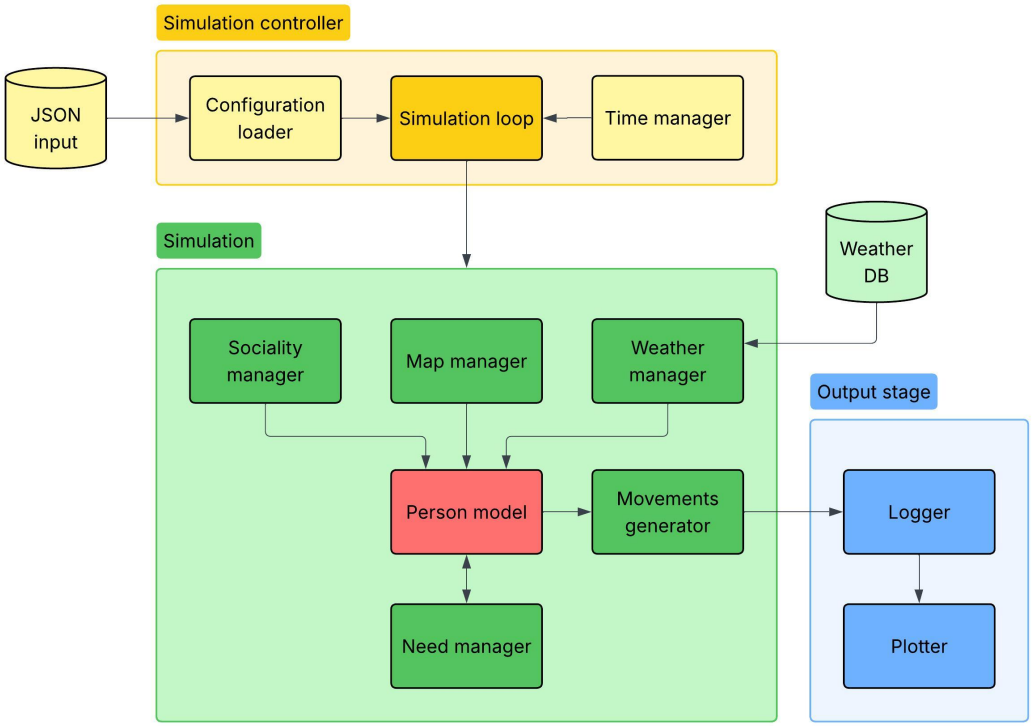

Figure 3.1: NeeMAS simulator architecture diagram

simulation, all relevant events and agent activities and movements are recorded by the *Logger* module to support subsequent analysis and evaluation.

The simulation process consists of three main phases: (i) initialization of the environment and agents, (ii) temporal evolution of the system over a fixed time horizon and (iii) generation of output data describing agent behavior. During the execution, each agent updates its internal state based on its needs and interacts with both the environment and other agents.

The simulator is implemented in Python and aims to reproduce human behavior by modeling both physiological needs (e.g., sleeping, eating) and social needs. These needs influence the decision-making process of each agent, determining both the actions performed and the movement within the environment.

Simulation Configuration

The entire simulation is configured through a structured input file, which specifies the components of the model. An example configuration is shown below:

```
{
```

```
  "people_json": [
    "person_Mario.json",
    "person_Anna.json",
    "person_Giorgio.json"
  ],
  "sociality_json": "social_matrices.json",
  "output_folder": "out_data",
  "places_json": "places.json",
  "begin_datetime": "2024-01-01T00:00:00.000000",
  "end_datetime": "2024-01-02T00:00:00.000000"
}
```


In the simulation file are specified all the configuration files needed to run the simulation. These include:

- The population of people that participates in the simulation, each one with its needs.
- The set of places that are visitable by the people, divided between outdoor and indoor places, each one with a different availability time ranges.
- The social matrices that are used for compatibility and initial friendship values.
- The duration of the simulation and the output folder.

While this configuration provides a flexible and expressive way to define complex scenarios, it also introduces a significant level of complexity, as users are required to manually specify low-level parameters and multiple interdependent files.

For this reason, the main objective of this thesis is to design a graphical user interface that abstracts this complexity. The interface allows users to define high-level behavioral characteristics, automatically translating them into consistent simulation parameters, thus enabling the use of the simulator also by non-expert users.

3.2. Population of the Simulation

The population of the simulation is modeled as a set of agents $A = \{a_1, a_2, ..., a_N\}$, where each agent represents an individual whose behavior evolves over time according to internal states and external interactions. Each agent is characterized by a set of needs and social attributes, which together determine its decision-making process and actions within the environment.

Maslow's Theory in the Simulator

Maslow's hierarchy of needs provides a conceptual framework to organize human behavior into different levels, ranging from basic physiological requirements to higher-level needs such as self-actualization. In this work, only the lower and intermediate levels of the hierarchy—namely physiological and social needs—are explicitly modeled.

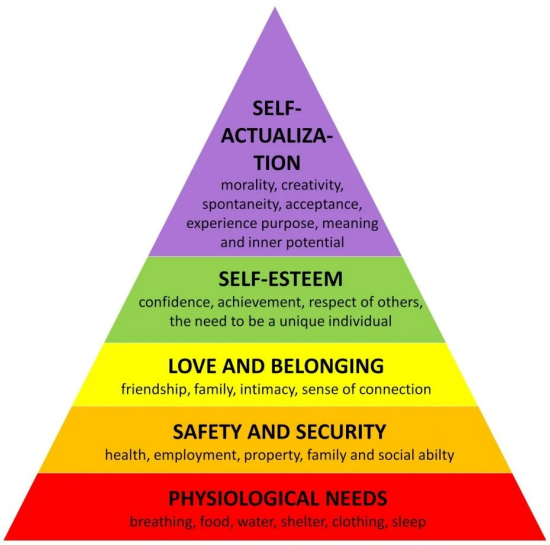

Figure 3.2: Maslow's Pyramid of Needs

This design choice is closely related to the objective of the simulator, which is to reproduce and analyze the movements of individuals within a controlled and safe environment, such as a care facility. In this context, physiological needs play a primary role, as they directly drive the majority of daily movements (e.g., going to eat, sleep, or use the bathroom), thus determining the spatial and temporal patterns of agents.

Social needs are included as they influence movement in a complementary way, encouraging agents to visit shared spaces and interact with others. This allows the simulation to capture more realistic trajectories, where movements are not only goal-oriented but also socially driven.

Higher-level needs, such as esteem and self-actualization, are not considered in the current model. These needs are more abstract, less directly observable in terms of movement and difficult to translate into spatial behaviors within the environment. Including them would increase the complexity of the model without significantly improving the accuracy of movement simulation.

Therefore, focusing on physiological and social needs represents a deliberate trade-off, enabling the simulator to effectively model movement patterns in a safe environment while maintaining a manageable level of complexity.

Person Configuration

The configuration of each agent is defined through a structured input file, which encodes its initial state and behavioral parameters. An example of this configuration is reported below:

```
{
    "id":1,
    "name":"Anna",
    "initial_position":2,
    "basic_needs":[...],
    "social_needs":[...],
    "wandering_needs":[...],
    "pathing_probabilities":[1,0,0,0]
}
```

This representation allows the decoupling between the simulation engine and the behavioral definition of agents, enabling flexibility but also introducing complexity in parameter specification.

The main components of each agent are the following:

- **Identifier and name:** the *id* uniquely identifies the agent and is used to define relationships in the social matrices, while the name is used for interpretability in logs and outputs. The separation between identifier and label allows consistent indexing while maintaining human-readable results.
- **Initial position:** defines the starting node of the agent in the environment graph. This parameter influences early interactions and movement patterns, especially in short simulations, where initial conditions can significantly affect the observed behavior.
- **Basic needs:** represent physiological requirements such as eating, drinking, sleeping, or using the bathroom. These needs can be modeled as internal state variables $n_i(t) \in [0, 1]$, whose evolution over time drives the agent's behavior. The choice of modeling needs as bounded continuous variables allows a gradual activation mechanism, avoiding unrealistic binary behaviors.

These needs are inspired by Maslow's hierarchy, where physiological needs constitute the base level and have higher priority in the decision process. This design ensures that essential activities dominate agent behavior when required, improving realism.

- **Social needs:** represent the tendency of agents to interact with others. Unlike physiological needs, these are influenced not only by internal state but also by the presence and compatibility of other agents. Their inclusion allows modeling emergent behaviors such as grouping or isolation, which would not be captured by purely individual dynamics.
- **Wandering needs:** model non-goal-oriented movement, typical of certain conditions such as dementia. This component introduces stochasticity in agent trajectories, preventing overly deterministic paths. The probability distribution over different movement patterns controls the degree of randomness: for example, changing these probabilities leads to less predictable and more exploratory behaviors.

3.2.1. Person: Basic Needs

Each agent in the simulation is associated with a set of basic needs, whose configuration is defined through the corresponding input file. These needs represent internal state variables that evolve over time and drive the behavior of the agent.

Each need in the simulator is represented as a continuous variable $n(t) \in [0, 1]$, representing its intensity at time t. The evolution of the need follows a cyclic behavior: it increases over time when it is not being satisfied and decreases during its execution phase.

More precisely, the update rule applied at each simulation step (with time discretized in seconds) is the following:

$$n(t+1) = \begin{cases} n(t) + \Delta_{inc}(t) + \epsilon_{inc}(t) & \text{if the need is not being executed} \\ n(t) - \Delta_{dec}(t) + \epsilon_{dec}(t) & \text{if the need is being executed} \end{cases}$$

Since $n(t)$ represents a normalized value, it is always kept within the interval $[0, 1]$. This means that, after each update, the value is clipped if it goes beyond the bounds.

Need Increment and Decrement

The terms $\Delta_{inc}(t)$ and $\Delta_{dec}(t)$ define how fast the need grows or decreases, while $\epsilon_{inc}(t)$ and $\epsilon_{dec}(t)$ introduce a variable amount of randomness. The evolution of these variables is represented over time t, as their intensity varies dynamically throughout the day.

The choice of $\Delta_{inc}(t)$ and $\Delta_{dec}(t)$ values depends on how long a phase is expected to last. In particular, the variation of a need is defined over the interval from $n = 1$ (completely unsatisfied) to $n = 0$ (fully satisfied). For example, if we consider the sleep need and assume that it goes from completely unsatisfied ($n = 1$) to fully satisfied ($n = 0$) in about 8 hours, the decrement value can be estimated by dividing 1 by the total number of seconds:

$$\Delta_{dec} = \frac{1}{8 \cdot 3600} \approx 3.47 \cdot 10^{-5}$$

In the simulator configuration files, $\Delta_{inc}(t)$ and $\Delta_{dec}(t)$ values are defined on an hourly basis to improve readability and clarity and then are internally converted into their equivalent per-second representation.

Randomness Component

The stochastic components $\epsilon_{inc}(t)$ and $\epsilon_{dec}(t)$ are usually defined as a small percentage of the $\Delta_{inc}(t)$ and $\Delta_{dec}(t)$ values. Its value is a number extracted randomly in the possible interval. For instance, it can be set to vary within $\pm 1\%$ of Δ, meaning that:

$$\epsilon(t) \in [-0.01\Delta,\ 0.01\Delta]$$

Even though these variations are small, they help make the simulation less rigid and more similar to real-world behavior, where actions are not perfectly regular.

Minimum and Maximum Thresholds

The activation of a need is regulated by two thresholds:

$$\text{if } n(t) > \theta_{max}(t) \Rightarrow \text{execution starts}$$

$$\text{if } n(t) \leq \theta_{min}(t) \Rightarrow \text{execution stops}$$

The use of a strict inequality for the activation condition is motivated by the need to control when certain activities can actually be performed. For example, eating is typically constrained to specific time windows and therefore the execution should only start in that specific moment. In this way, setting $\theta_{max}(t)$ to 1 prevents the execution of the need, being $n(t) \in [0, 1]$ and so impossible for it to be greater than 1.

On the other hand, once the execution has started, the satisfaction process is always carried out until the minimum threshold is reached. This ensures that the need is consistently reduced and avoids premature interruptions.

This mechanism introduces a hysteresis effect, preventing unstable oscillations between activation and deactivation. The separation between increment and decrement phases ensures that the need increases only when it is not being satisfied and decreases only during execution, resulting in a more realistic behavior.

When multiple needs exceed their activation threshold, the agent selects which one to execute based on their priority values, ensuring that more urgent needs are addressed first.

Basic Need Configuration

The overall configuration of a need, including thresholds, increment and decrement rates and stochastic parameters, is specified through a Json configuration file, as shown below:

```
{
    "name":"sleep",
    "initial_val":0.76,
    "min_exec_time":900,
    "priority":0.7,
    "preemptible":true,
    "th_min":[...],
    "th_max":[...],
    "increments":[...],
    "eps_inc":0.021,
    "decrements":[...],
    "eps_dec":0.016,
    "eps_interval":2880,
    "places":[
        {
            "id":45,
            "practicability":1.0
        }
    ]
}
```

As we can see from the example, each need is defined by the following parameters that can be set to obtain the desired behavior:

- **Name of the need**: used for clarity in the output graphs and in the logs.

- **Initial value**: Sets the initial value for the need, this is the starting value that will be incremented.
- **Minimum execution time**: it is the minimum amount of time in seconds required to execute the need.
- **Priority**: sets the need's priority relative to the others. It is a value between 0 and 1, with higher numbers meaning greater urgency.
- **Preemptibility**: defines if a higher priority need can interrupt or pause the execution of the need
- **Minimum threshold (θ_{min})**: defines the lower threshold of the need. When the value of the need falls below this level, the execution phase is terminated and the need starts increasing again. The threshold is defined in the range $[0, 1]$.
- **Maximum threshold (θ_{max})**: defines the upper threshold of the need. When the value of the need exceeds this threshold, the execution phase is triggered and the need starts decreasing. The threshold is defined in the range $[0, 1]$; setting it to 1 effectively prevents the activation of the need, as the threshold becomes unreachable. This can be used to trigger the execution of the need at specific times, blocking it even if saturated until the desired moment.
- **Increments(Δ_{inc})**: defines how much the need grows when it is higher than the minimum threshold and is not being executed. The value represents the need's variation over the period of time of one hour.
- **Decrements(Δ_{dec})**: defines how much the need decreases when is executed. The value represents the need's variation over the period of time of one hour.
- **Increment epsilon(ϵ_{inc})**: sets the level of random noise to be added to the incrementing updates. The value is usually a small percentage of the increment. Its purpose is to improve realism.
- **Decrement epsilon(ϵ_{dec})**: sets the level of random noise to be added to the decrementing updates. The value is usually a small percentage of the decrement. Its purpose is to improve realism.
- **Epsilon interval**: specifies the interval of time for which the values of epsilon (for increments and decrements) are cached in memory and not recomputed. It avoids skipping the recalculation of the parameters if the calls are close in time.
- **Places**: defines in which places the person can satisfy the need. Each place is

defined by its id and the practicability, that is a value between 0 and 1 and can be defined with hourly intervals. The practicability is used to choose when different places are available at the same time.

For all the thresholds, increments/decrements and epsilon parameters the values can be defined with an hourly precision using the following syntax: [initial time, end time, value], where initial and end time indicate the hours of the day, while the value range is [0,1]. This feature of the simulator allows to vary the behavior of the people during the different parts of the day. Changing the increments and decrements it is possible to change the speed of execution of the needs.
For example, considering the eating need, changing the decrements throughout the day it is possible to set the behavior of spending more time eating during the main meals and less time during breakfast and snacks.

Basic Need Example

To explain better how the needs evolve over time, we will see an example of the output of a need, in this case the eat need. To better illustrate the temporal evolution of a need,

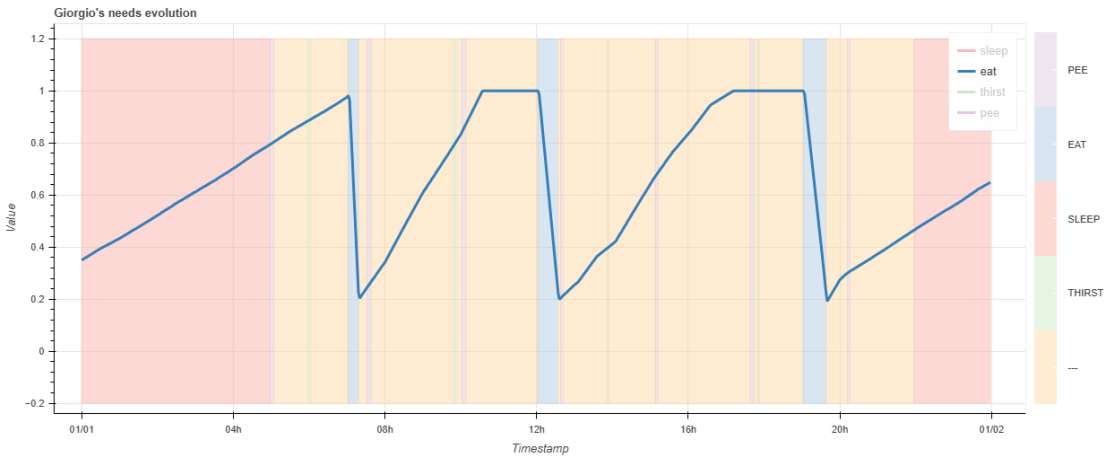

Figure 3.3: Eat need evolution during a day

Figure 3.3 shows an example of the output generated by the simulator, in this case for the *eat* need over the course of a day. The plotted curve represents the value of the need $n(t)$, which evolves continuously between 0 and 1.

During periods in which the need is not being satisfied, its value increases approximately linearly according to the increment rate, reflecting the gradual rise of urgency. When the value overtakes the upper threshold θ_{max}, the execution of the need is possible and is executed based on its priority. In the case of the example, the execution of the need

is controlled by setting θ_{max} to 1, blocking the execution of the need until the allowed eating time is reached. To do so, the value of θ_{max} is lowered at the corresponding time, starting in this way the execution of the need. When executing, its value starts decreasing at a faster rate, determined by the decrement parameter, until the lower threshold θ_{min} is reached, at which point the execution stops and the growth phase begins again.

The resulting pattern is a cyclic behavior characterized by repeated rises and sharp drops, corresponding to alternating phases of accumulation and satisfaction. Small irregularities in the curve are due to the stochastic component introduced by the epsilon parameters, which adds variability and prevents perfectly deterministic dynamics.

3.2.2. Person: Social Need

Sociality is another important aspect in the simulation. In fact people, after satisfying their physiological needs, can interact with each other and develop friendship or antagonism relationships. Differently from physiological needs, social needs are not confined to specific time intervals but are distributed over the entire duration of the simulation. As a result, their effects emerge over longer temporal horizons and a single day of simulation may not be sufficient to fully capture their dynamics.

In the simulator, social behavior is modeled through two matrices: the compatibility matrix and the friendship matrix. Both matrices have dimensions $\frac{N(N-1)}{2} \times 3$, where N is the number of agents in the simulation. This structure allows representing all unique unordered pairs of agents, with each row encoding a pair of individuals and the corresponding relationship value.

This representation ensures that the matrices contain the minimum number of rows required to cover all unique unordered pairs of agents. Each row consists of three elements: the first two columns store the identifiers of the two agents forming the pair, while the third column encodes the corresponding relationship value. This value stays in the range $[-1, 1]$ for the compatibility matrix, representing the degree of affinity or conflict between agents and in the range $[0, 1]$ for the friendship matrix, representing the initial strength of their social bond.

We can now analyze the usage of two matrices to understand their role in the simulator:

- **Compatibility matrix**: defines the relationship tendency between each pair of agents. The parameters of this matrix remain constant throughout the simulation. The values range between -1 and 1, where positive values indicate a tendency to strengthen the relationship over time, while negative values lead to a progressive

weakening or distancing between the two agents. In particular, values close to 1 result in a rapid increase in friendship, whereas values close to -1 may prevent the formation of stable relationships and the social avoidance of that person. For simplicity in the developing of the simulator, the matrix is symmetric, meaning that the compatibility between agent i and agent j is equal to that between j and i.

- **Friendship matrix**: represents the level of friendship between each pair of agents. The values range between 0 and 1, where 0 indicates no relationship, 0.5 a neutral relation and 1 a strong friendship. This matrix is initialized at the beginning of the simulation and evolves over time as a result of interactions between agents.

  The evolution of friendship depends on both the compatibility between agents and the execution of social interactions. In particular, when two agents interact, their friendship value is updated according to their compatibility: positive compatibility leads to an increase in friendship, while negative compatibility causes a decrease. The magnitude of the change depends on the strength of the compatibility value, resulting in a faster or slower evolution of the relationship. On the other hand, when two agents are not interacting, their level of friendship gradually decreases over time. As a result, the final friendship level reflects a balance between periods of interaction and periods of separation throughout the day. Similar to the compatibility matrix, the friendship matrix is symmetric, meaning that the friendship level between agent i and agent j is equal to that between j and i.

It is important to distinguish between the evolution of friendship and the dynamics of the social need. These two processes are modeled independently. The friendship levels are updated regularly by the *update_friendship()* function in the *SocialityManager* class regardless of whether the social need is executed or the agents interact with each other. In contrast, the execution of the social need follows its own dynamics determined by the set of parameters specified in the Json file and is satisfied exclusively during social interactions. In fact if an agent is in an environment where no other agents are present, the social need is not satisfied and therefore does not decrease, reflecting the absence of social engagement.

Friendship Update Method

The variation Δf_{ij} of friendship between two agents depends on the following parameters:

- The values of the increment and decrement ratios r_{inc} and r_{dec} that control the speed of friendship growth and decay. (The default values are $r_{inc} = 1.0$, $r_{dec} = 0.5$).

- Their compatibility c_{ij} as set in the compatibility matrix.
- The hourly friendship increment or decrement, that is a parameter hard-coded in the simulator.
- The time Δt elapsed between two updates, expressed in seconds.

When agents are located in the same place, friendship increases according to:

$$\Delta f_{ij} = r_{inc} \cdot c_{ij} \cdot \frac{1}{600} \cdot \frac{\Delta t}{3600}$$

Conversely, when agents are not in the same place, friendship decays as:

$$\Delta f_{ij} = -r_{dec} \cdot |c_{ij}| \cdot \frac{1}{1800} \cdot \frac{\Delta t}{3600}$$

The updated value is then clipped to ensure that friendship remains bounded within $[0, 1]$.

Friendship Configuration

The matrices that manage social relations are configured in the *social_matrices* Json file, with a structure like the one shown in the following example:

```
{
  "compatibilities": [
    [0, 1, 0.5], // Maria & Teresa
    [0, 2, 0.7], // Maria & Edoardo
    [0, 3, -0.7], // Maria & Assunta
    [1, 2, 1.0], // Teresa & Edoardo
    [1, 3, 0.3], // Teresa & Assunta
    [2, 3, 0.1]  // Edoardo & Assunta
  ],
  "friendships": [
    [0, 1, 0.5], // Maria & Teresa
    ...
  ]
}
```

From the matrices, Maria and Teresa show moderate compatibility, Teresa and Edoardo very high compatibility, while Maria and Assunta have negative compatibility indicating conflict or avoidance. All friendships start equal and evolve based on these interactions.

After executing the simulation, the output of this process is visualized through friendship heatmaps, which show the evolution of relationships among individuals during the simulation. These heatmaps provide an intuitive representation of how social ties strengthen or weaken over time as a result of interactions. The final heatmap of the simulation configured with the previous parameters is shown in Figure 3.4

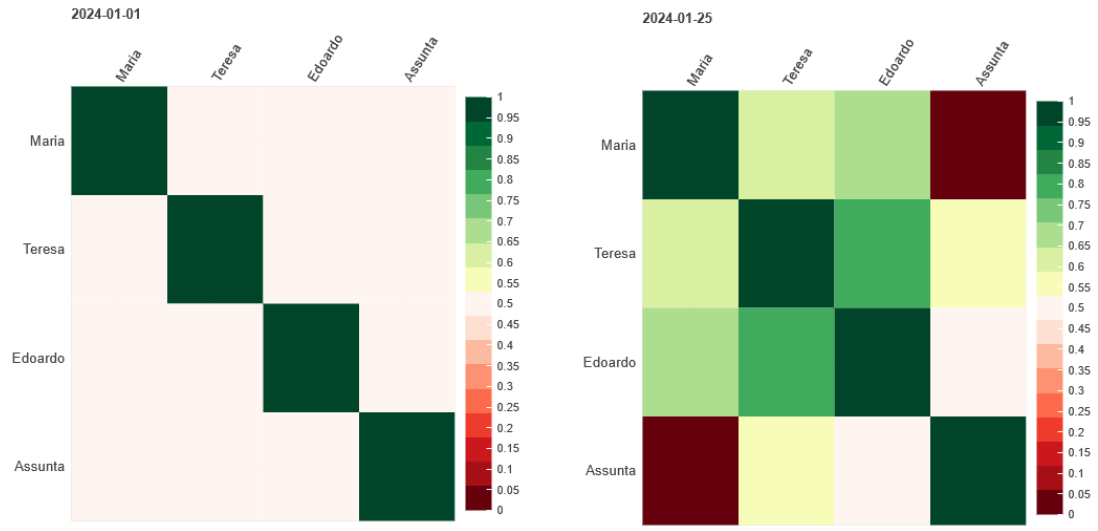

Figure 3.4: Friendship heatmap at the beginning and end of the simulation

Checking the heatmap, it is clear that agents with higher compatibility tend to strengthen their relationships more rapidly, while lower or negative compatibility results in slower growth or a decrease in friendship.

3.2.3. Person: Wandering Needs

Besides physiological and social needs, the simulator also includes wandering, defined as a tendency to move without a specific goal. Although wandering is not a fundamental need in the traditional sense, it is modeled as a need-like process to capture spontaneous and non-goal-oriented movement.

This behavior is particularly relevant in the context of elderly individuals affected by cognitive impairments, such as dementia, where wandering patterns are commonly observed. Modeling this component allows the simulator to reproduce more realistic movement dynamics in care environments.

In the simulator, two types of wandering behaviors are defined:

- **Wandering**: simulating a searching behavior where the person can't visit the same place more than two times in a row.

- **Pacing**: where the person paces back and forth between a small set of familiar places simulating an anxious behavior.

Wandering Configuration

Wandering needs are configured in the same way as the other needs, with the main difference that the places array defines the places the person will move between and the minimum execution time represents the seconds at each location.
An example of a wandering need is reported here:

```
wandering_needs: [
  {
    name: "wandering",
    min_exec_time: 600,
    priority: 0.2,
    th_max: 0.15,
    th_min: 0.02,
    increments: 0.02,
    decrements: 0.01,
    places: [...],
  }
]
```

When configuring wandering, another important parameter is the pathing probabilities. It is a special attribute set directly in the person's Json and is an array with the following structure:

```
pathing_probabilities: [0.6, 0.2, 0.1, 0.1]
```

The parameter determines the likelihood of choosing specific movement patterns whenever the agent navigates the environment. When setting the pathing_probabilities attribute in a person, it will affect the way that person is moving during the simulation, even when the wandering need is disabled or not executed. The array contains the probabilities associated with the different possible movements. The sum of the probabilities has to be 1.

- The first value represents the probability of walking in a *direct path*, meaning that the person will follow the shortest path.
- The second value represents the probability of a *disoriented path*, meaning that the person will take wrong turns but will eventually arrive.

- The third value represents the probability of a *looping path*, meaning that the person will take a circular deviation in the route.
- The last value represents the probability of a *pacing path*, meaning that the person will walk back and forth on a segment.

3.3. Places

To run the simulation, besides defining the people and their needs, another crucial aspect to set is the map where the agents can move and interact with each other.
The simulator is designed to represent an artificial world where are present indoor and outdoor places, connected with paths. Each place can also be defined as a social place, where interaction with other people is possible.
This allows the accurate representation of a senior care facility, with its rooms, hallways, kitchens, social areas and outdoor areas. The structure of the environment is modeled as a graph, where nodes represent places and edges represent the accessible connections between them, each weighted by their distance. This representation is particularly suitable because it allows capturing both the spatial layout and the connectivity constraints of the environment in a formal and efficient way. Modeling the map as a graph enables the use of well-established algorithms for pathfinding and navigation, allowing agents to compute optimal or alternative routes between locations.

Moreover, this abstraction naturally supports complex layouts, where not all places are directly reachable and movement is constrained by the structure of the environment. As a result, the graph representation provides a flexible and scalable framework to model indoor and outdoor spaces while preserving computational efficiency.

Map Configuration

The overall structure of the map's json configuration file is the following:

```
{
  nodes: [
    {
      name: "Parrucchiere 01",
      id: 1,
      x: 900,
      y: 828,
      social: true,
```

```
      outdoor: false,
    },
    ...
    ],
  weighted_edges: [
    [1, 2, 85],
    [2, 4, 121],
    ...
}
```

Each place is defined in the json file as a node and is configured using these parameters:

- **Name**: the name of the place, used for clarity in the outputs of the simulation.
- **Id**: the unique id of the place, used to identify the place in the other configuration files.
- **Coordinates**: the coordinates of the place in the space.
- **Social**: boolean parameter that defines if the place is suitable for social interactions.
- **Outdoor**: boolean parameter that defines if the place is indoor or outdoor.

The connections between places are represented in the weighted edges array, indicating the first place, the second place to which it connects and the distance between them.
For example *[1, 2, 85]* in the array means that the place 1 is connected to the place 2 with a distance of 85 meters. The distance will be used from the simulator to find the best path to reach the desired place.

3.4. Generation of the Parameters

After describing which are the parameters that allow the simulation to be configured and to run, we can understand that the generation of the mathematical parameters starting from the habits of the people is a complex task due to the large number of variables involved.
The purpose of this thesis is the creation of a graphical user interface which allows the generation of the parameters in an easy and intuitive way.
To achieve this, two main categories of needs are identified:

- **Scheduled needs**, which occur within predefined time intervals and are driven by explicit temporal constraints.

- **Periodic needs**, which emerge regularly over time and are characterized by frequency and duration rather than fixed schedules.

This distinction is crucial, as it leads to two different parameter generation strategies: one based on time windows and one based on temporal cycles. The following sections describe these approaches in detail.

3.4.1. Needs Defined at Scheduled Times

Some needs in the simulation are closely linked to specific times of the day, such as meals or scheduled activities. In these situations, behavior is influenced not by how often something happens, but by the set time frames when the activity is expected. Therefore, it is important to use a parameter generation strategy that directly reflects these time-based limits. In Figure 3.5 it is shown the eat need variation over the course of the day, which is an example of scheduled need.

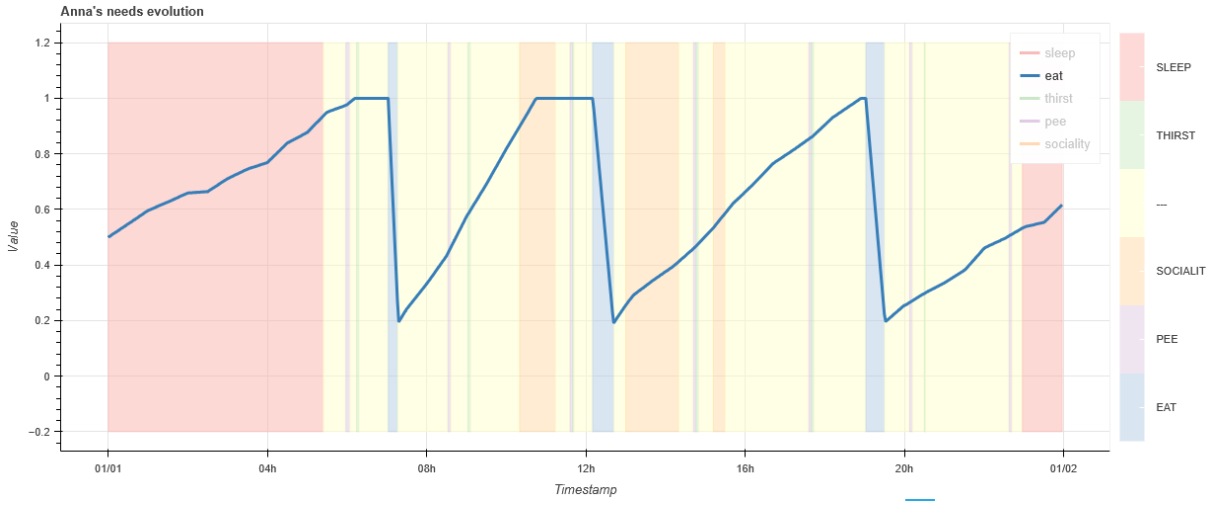

Figure 3.5: Example of scheduled need

The main function that generates the parameters from a list of scheduled times is the generate_params_from_list function, which can be found in the simulation/tools.py file. The function's header is reported:

```
generate_params_from_list(
    activity_list,
    inactive_max_thr,
    active_max_thr,
    inactive_min_thr,
    active_min_thr,
```

```
        default_incr,
        default_decr
    )
```

The parameters that are needed for the function are the following:

- *activity_list:* represents the set of scheduled activities, stored as a list of dictionaries with the keys `start_time`, `end_time`, `tot_duration_hour`, `type` and `enabled`. This list is fundamental because it defines when the need can be satisfied. Changing this list directly alters the temporal behavior of the need, enabling or disabling specific activity windows.
- *inactive_max_thr:* defines the maximum threshold during inactive periods (default value: 1). This prevents the activation of the need outside the allowed time intervals. If reduced, the need could be triggered even when the activity should not occur, leading to unrealistic behavior.
- *active_max_thr:* defines the maximum threshold during active periods (default value: 0.6). This value determines when the need becomes urgent within the scheduled interval. Lower values anticipate the activation, while higher values delay it.
- *inactive_min_thr:* defines the minimum threshold during inactive periods (default value: 0.6). This ensures that the need does not decrease excessively when it cannot be satisfied. Modifying this value affects how "stable" the need remains outside activity windows.
- *active_min_thr:* defines the minimum threshold during active periods (default value: 0.01). It controls how much the need is reduced after execution. Higher values result in partial satisfaction, while lower values allow a more complete reset.
- *default_incr:* is the default increment rate (default value: 0.2), used when no activities are provided. This ensures that the system still behaves consistently even in the absence of scheduled inputs.
- *default_decr:* is the default decrement rate (default value: 0.1), also used when no activities are defined. It guarantees a fallback behavior for need satisfaction.

During the process, the parameters are calculated by setting the corresponding values passed in the function, during the active and inactive times of the activities, calculating the corresponding increments and decrements that produce the desired behavior.

3.4.2. Periodic Needs

Unlike needs defined at scheduled times, periodic needs are not tied to fixed time intervals but instead emerge regularly throughout the day based on frequency and duration parameters. These needs are characterized by a cyclic behavior in which their activation depends on how often and how long an activity is performed, rather than on predefined time slots. An example of periodic need is the thirst need, as seen in Figure 3.6, which repeats regularly during the day. The generation of parameters for periodic needs is handled

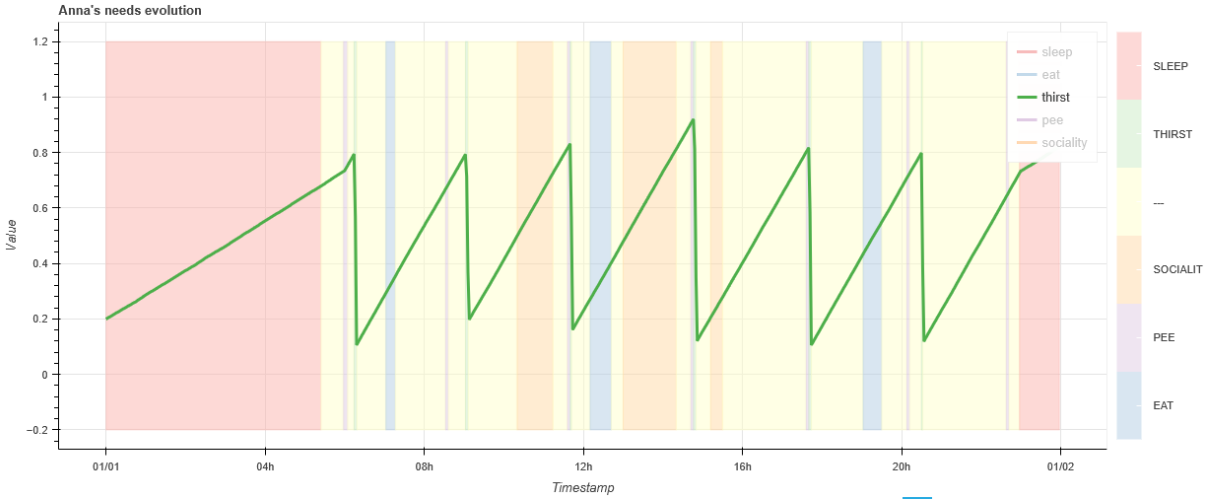

Figure 3.6: Example of periodic need

through the `compute_parameters` function, which derives the mathematical parameters starting from high-level user inputs, such as the daily frequency of an activity and its average duration.

In this approach, the goal is to translate intuitive behavioral descriptors (e.g., "a person drinks 6 times per day for about 5 minutes each time") into the parameters required by the simulator, namely thresholds, increments and decrements.

To calculate the parameters firstly the available time for the activity is computed, excluding periods in which the need cannot occur (e.g., nighttime sleep). This time window is then divided by the desired daily frequency to estimate the average interval between consecutive executions of the activity.
Then the increment and decrement rates are calculated so that the need gradually increases to its activation threshold within the computed interval and is fully reduced during the execution phase, according to the specified activity duration.

This approach allows the definition of realistic and flexible behaviors starting from simple and intuitive inputs, making it particularly suitable for integration within a graphical user

interface. The user can specify high-level habits without dealing directly with low-level simulation parameters, while still obtaining consistent and predictable results.

3.5. The Questionnaire and the Identification of the Personas

To create realistic profiles for the simulation and map their behaviors to the agents, the concept of *persona* is introduced. A persona is an abstract profile that represents the typical characteristics, habits and behaviors of a group of individuals with similar lifestyles. It is described as if it were a real person, including attributes such as name, gender, age, occupation, interests and daily habits. Instead of modeling each individual separately, personas provide a simplified yet effective way to translate real-world observations into a set of parameters used to configure the agents within the simulation.

The Questionnaire

To define these personas, a questionnaire (reported in Appendix A.1) was designed as part of the NeeMAS project. It was distributed in both paper-based and online formats to individuals over the age of 65. The aim was to collect information on the daily habits of this age group, which represents the primary target population of the simulator. To ensure clarity for participants, only simple and easily interpretable parameters were considered, avoiding abstract or potentially ambiguous concepts such as social relations or personal beliefs.

The questionnaire consists of 20 multiple-choice questions, divided into four sections: eating, drinking, social activities and sleeping. These areas were selected because they capture the main activities that influence how a person moves and behaves throughout the day. Each section focuses on different aspects of daily life:

- **The eating section**: provides information about when people usually have their meals, how long they last and where they take place.
- **The drinking section**: is focused on the distribution of drinking during the day, analyzing if there are parts where is more frequent.
- **The social section**: explores how often people interact with others, with whom and in which places, giving insight into social behavior and the importance for them.
- **The sleeping section**: provides information about sleep time, duration and the presence of an afternoon nap.

The data collected through the questionnaire is then used to define a set of representative personas. Instead of modeling each individual separately, personas make it possible to reduce the complexity of the simulation while still maintaining realistic variability.

Methodology for Questionnaire Analysis

After the distribution of the questionnaire, the responses were collected both in paper and digital format. To analyse the questionnaires, different methodological approaches were considered.

The first approach consists of a manual analysis using spreadsheets, where responses are aggregated into groups in order to define personas based on a set of predefined characteristics derived from multiple questions.

A second alternative involves the use of machine learning techniques, enabling a more quantitative and statistical analysis of the data through methods such as Principal Component Analysis (PCA) and classification algorithms. These techniques allow for dimensionality reduction and the identification of hidden structures within the responses, potentially improving the robustness of the resulting personas.

A third approach relies on large language models (LLMs) to automatically extract relevant information from the questionnaire responses in a more flexible manner. Thanks to their ability to understand the context, LLMs enable the extraction of structured insights from semi-structured or qualitative data, reducing the need for extensive manual work. In addition, LLM-based analysis allows for a more natural interpretation of the responses, without requiring rigid preprocessing pipelines.

4 | Implementation

This chapter presents the contributions of this thesis to the NeeMAS simulator. In particular, it describes the architecture and design of the graphical user interface developed as part of this work. The chapter details its main components, underlying design decisions and integration with the existing simulator framework.

4.1. Introduction

The primary objective of this thesis was the development of an intuitive graphical user interface for the NeeMAS simulator. Before this work, NeeMAS could only be used through a command line interface (CLI), requiring users to manually create and edit a large collection of JSON configuration files describing the simulation environment, agents, needs, social relationships and execution parameters. While this kind of approach provides direct access to the simulator's internal configuration, it also introduces a significant usability barrier, particularly for users unfamiliar with the simulator's architecture or the underlying data structures.

Configuring a simulation scenario required users to manually define numerous interdependent parameters, often involving complex calculations and a detailed understanding of the simulator's behaviour. Furthermore, the absence of visual tools made it difficult to inspect the simulation environment, verify configuration correctness, or refining the scenarios. As a result, creating realistic simulations was a time-consuming process that demanded both technical expertise and deep knowledge of the simulator's configuration model.

To address these limitations, a dedicated graphical interface, was designed and implemented. The objective was to simplify the creation and management of simulation projects and to abstract the complexity of the JSON files. Using the interface, users can focus on modelling scenarios rather than on low-level configuration details.

4.2. The Overall Structure

A key design principle in the development of the NeeMAS Interface was the separation between the simulation engine and the graphical user interface. This architectural choice improves modularity, simplifies maintenance and facilitates future extensions by allowing the two components to evolve independently. In fact, the simulator and the graphical interface are maintained as distinct projects hosted in separate repositories.

The original NeeMAS simulator remains fully functional as a standalone command line application and can be executed independently through its CLI. The graphical interface serves as an additional layer that simplifies the creation, configuration and execution of simulation scenarios without modifying the simulator's core functionality. When the graphical interface is used, the CLI simulator acts as the computational backend, while the interface serves as the frontend for the creation, configuration and management of simulation projects. The frontend allows users to easily create and store multiple projects, enabling simulation scenarios to be saved, reopened and modified in a second moment. This functionality simplifies the iterative development of scenarios and improves collaboration, giving the possibility to share the high level, intuitive representations of the simulations with others. On the other hand the backend provides the core functionality required to model agent behavior and execute simulation runs.

Communication between the backend and the frontend is based on the JSON files representing the simulation configuration. The frontend is responsible for collecting user-defined settings and generating the corresponding JSON configuration file. Once the simulation is launched, the simulator backend is executed from within the interface and uses the generated configuration as input. The simulator backend then performs the computation and returns the outputs, which are subsequently processed and displayed by the graphical interface.

Both components are based on *Python 3*. The *NeeMAS Interface GUI* is implemented using *PySide6*, the official Python binding for the Qt Widgets framework. This cross-platform library enables the interface to be executed on different operating systems, including Windows, macOS and Linux. Environment management, dependency installation and project execution are handled through the *UV* package manager, ensuring a consistent and reproducible development environment across both projects.

4.3. The Interface Architecture

This section describes the internal architecture of the NeeMAS interface, focusing on the main data structures and their relationships. The interface is implemented using object-oriented programming and follows the Model–View–Controller (MVC) design pattern, as represented in Figure 4.1. This design pattern separates the application into three interconnected components: the model, which manages the simulation data and logic; the view, which defines the graphical representation of the system; and the controller, which handles user input and coordinates interactions between the model and the view.

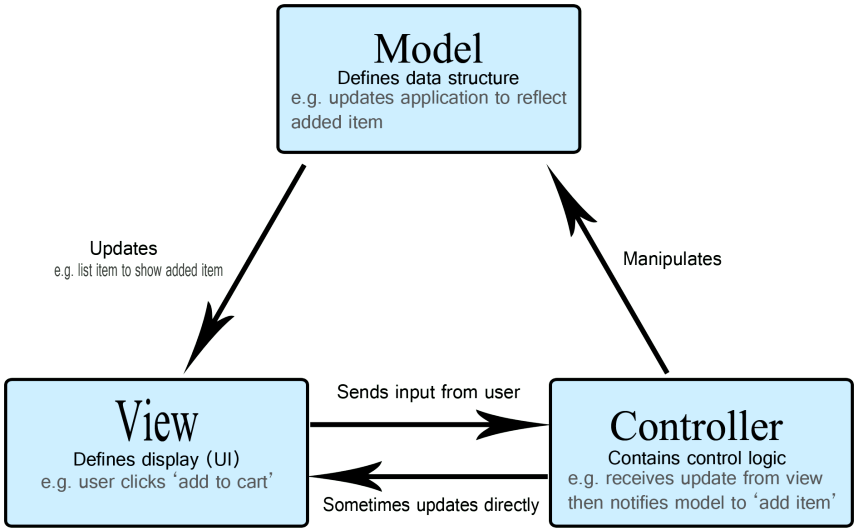

Figure 4.1: Model View Controller design pattern

Model

The *Model* represents the core of the NeeMAS interface and is responsible for storing all the information required to generate the configuration files used to run the simulation. It is implemented in the `/simulation` module and defines the complete set of entities involved in the system, including agents, the environment and social structures.

Figure 4.2 illustrates the main structural components of the model and their relationships. At the highest level, the *Simulation* class, implemented as a *Singleton*, acts as the central coordinator and is associated with a collection of *Person* agents, a *Map* and a *SocialityMatrix*. It provides a shared entry point for managing the simulation and ensures

consistency across the different components of the model.

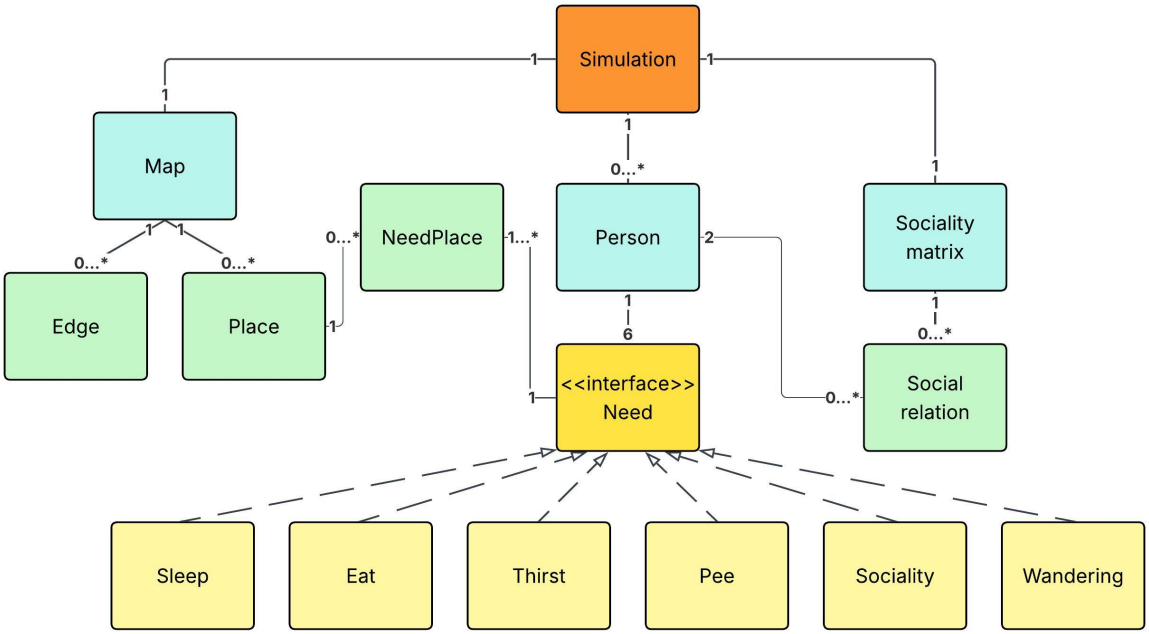

Figure 4.2: Class diagram of the model used by NeeMAS interface

The *Person* class is the representation of the agents in the simulation. Each agent has a set of six needs, represented through the *Need* interface class and implemented by the concrete classes: *Sleep*, *Eat*, *Thirst*, *Pee*, *Sociality* and *Wandering*. This abstraction ensures that all needs share a common lifecycle and decision-making structure while preserving their specific behavioral dynamics. Each of these classes stores high-level information related to the agent's behavioral patterns, such as sleep schedules or meal timing and contains the logic needed to convert them to the JSON files.

The *Map* represents the simulation environment as a graph composed of *Place* nodes connected through *Edge* objects. Each *Place* may be associated with one or more *Need* objects through the use of *NeedPlace* instances.

Finally the *SocialityMatrix* stores the information about agents social relations through *SocialRelation* objects, tracking compatibility and friendship levels. This structure enables the representation of interpersonal relationships and supports the evolution of social interactions over time.

View

The *View* corresponds to the graphical representation of the system and is responsible for displaying the simulation interface to the user. The graphical layouts are designed using the Qt Designer tool and subsequently converted into Python-compatible files. The layout files are stored in the `gui/layout` folder as `.ui` files. The View defines the structure of the graphical interface, including windows, panels and widgets used to visualize and interact with the simulation. It does not contain application logic but provides the visual components that are dynamically updated based on the state of the Model.

Controller

The *Controller* is implemented in the `gui/controller` module and acts as an intermediary between the Model and the View. Each screen is associated with a dedicated controller file, which improves modularity and keeps the code organized. The Controller manages user interactions, processes input events and updates the Model accordingly. At the same time, it ensures that changes in the Model are reflected in the View. In this way, it coordinates the execution flow of the application, including window management and execution control.

4.4. Graphical Interface Overview

This section presents the main functionalities implemented in the NeeMAS graphical interface. The interface was designed to simplify the creation, configuration and execution of simulation scenarios, providing users with a visual environment for managing all aspects of the simulation workflow.

In the following sections will be described the main screens of the application and their role within the overall workflow. For each screen, the corresponding functionality is explained and illustrated through screenshots. Together, these components provide a complete environment for project management, scenario configuration and simulation execution.

4.4.1. Home Screen and Project Management

As shown in Figure 4.3, the home screen is the first interface displayed when the application is launched. This screen serves as the entry point to the system, allowing users either to create a new project or to open an existing one. By providing direct access to previously saved configurations, the home screen supports the management and continuation of simulation projects across multiple sessions. When creating a new project, as shown

in Figure 4.4, the user is required to provide the information necessary to initialize the simulation project. This includes the project name, the directory in which the project will be saved and the weather database to be used during the simulation. Additionally, the user may optionally specify a places file containing a predefined set of locations to be imported into the simulation environment. Once the required information has been provided, the project structure is created and becomes available for further configuration.

Figure 4.3: Home screen of Neemas Interface

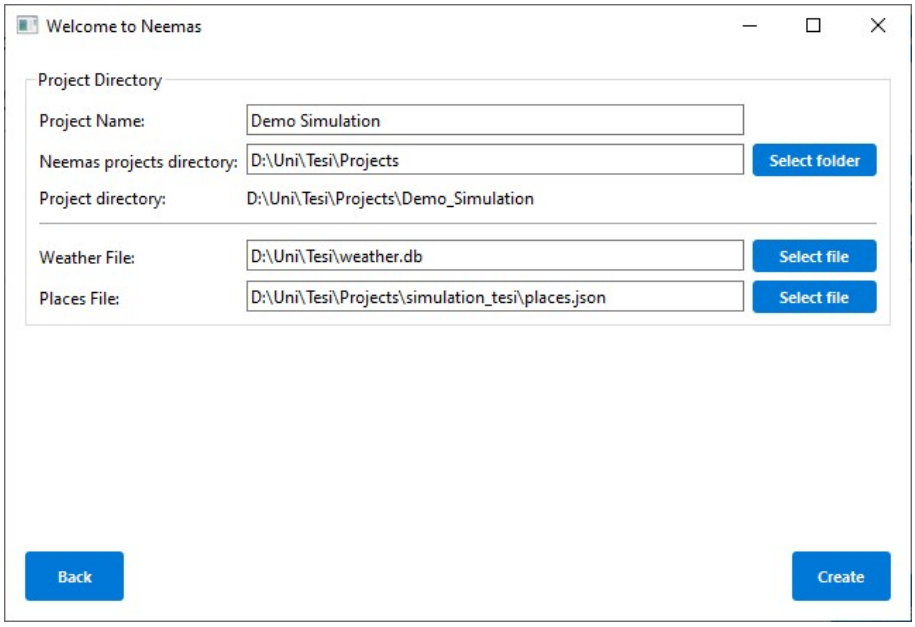

Figure 4.4: Project creation interface

After a simulation project has been created, the main interface window is displayed. This window provides access to all the functionalities required to configure and manage

a simulation scenario, including the definition of agents, places, social relationships and simulation parameters.

As shown in Figure 4.5, the interface is organized as a multi-tab workspace, allowing different aspects of the project to be edited simultaneously. A sliding panel located in the lower part of the window displays simulation logs and execution progress, providing real-time information during simulation runs. On the left side, a tool bar gives access to the main functionalities of the application, enabling users to efficiently switch between the different projects and configuration.

4.4.2. Agents Configuration

In the *People* tab, shown in Figure 4.5, the interface is organized around a table containing the list of agents included in the simulation. For each agent, a summary of the configured needs and behavioral parameters is provided, allowing users to quickly inspect the characteristics of the simulated population.

A set of dedicated buttons located at the top of the window enables users to create, edit, duplicate, or remove agents. When an agent is selected, detailed information is displayed in the panel on the left side of the interface. This panel includes personal attributes such as age and sex, as well as simulation-specific parameters, including the assigned bedroom, default kitchen and initial position within the environment.

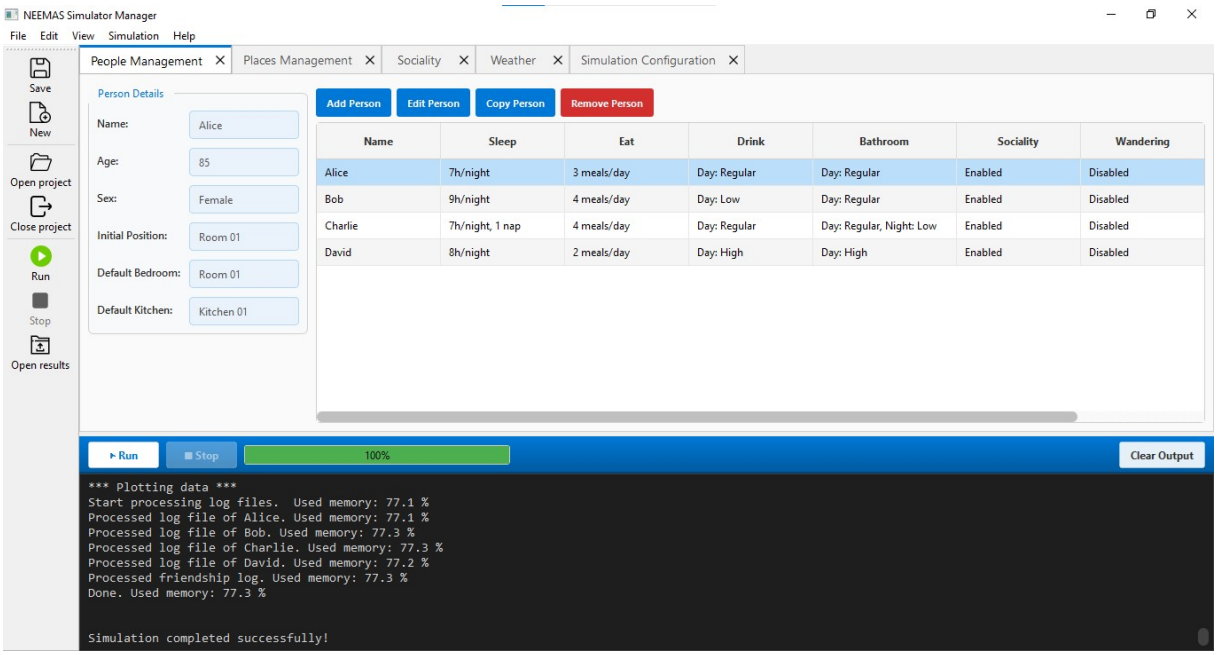

Figure 4.5: People management window

When creating or editing an agent, the *Person Editor* window is displayed. This interface allows users to configure all six needs modeled by the simulator: *Sleep*, *Eat*, *Drink*, *Bathroom*, *Sociality* and *Wandering*. In addition, users can define general agent attributes, such as name, age, sex, default locations and initial position within the environment.

To facilitate configuration, the editor provides a dedicated interface for each need, tailored to its specific characteristics. This approach enables users to define behavioral habits in a precise yet intuitive manner. Figure 4.6 illustrates the configuration interface for the *Eat* need. Users can specify meal schedules, expected meal durations and optional snack events that may occur throughout the day.

The upper-right section of the window allows users to associate multiple places for the execution of the selected need. For each place, a selection probability can be specified, enabling the simulator to take into account preferences among alternative locations, specifying also the time availability. The lower-right section contains the advanced parameters governing the need dynamics. These parameters are initialized with default values, allowing users to create realistic configurations without requiring detailed knowledge of the underlying model, while still providing the flexibility needed for advanced customization.

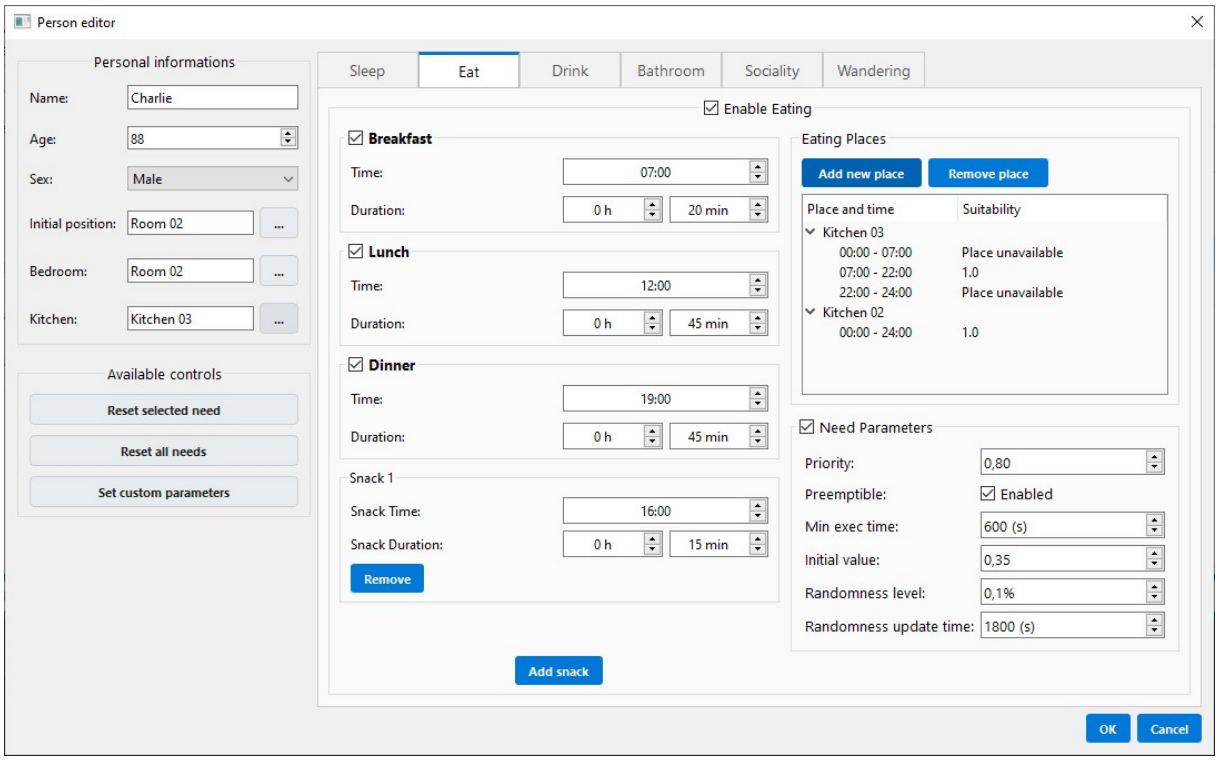

Figure 4.6: Person creation and editor window

4.4.3. Environment Configuration

The *Places Management* tab, shown in Figure 4.7, provides the tools required to define and manage the simulation environment. The interface is organized around a table containing all places currently included in the project. For each place, the table displays its name, position coordinates, outdoor or indoor, social type, availability and connections with other locations, allowing users to easily obtain an overview of the environment structure.

A set of dedicated buttons located at the top of the window enables users to create, edit, or remove places. A search bar is also provided to quickly locate specific locations within large simulation environments. Through this interface, users can build incrementally the graph representing the environment in which agents move and interact.

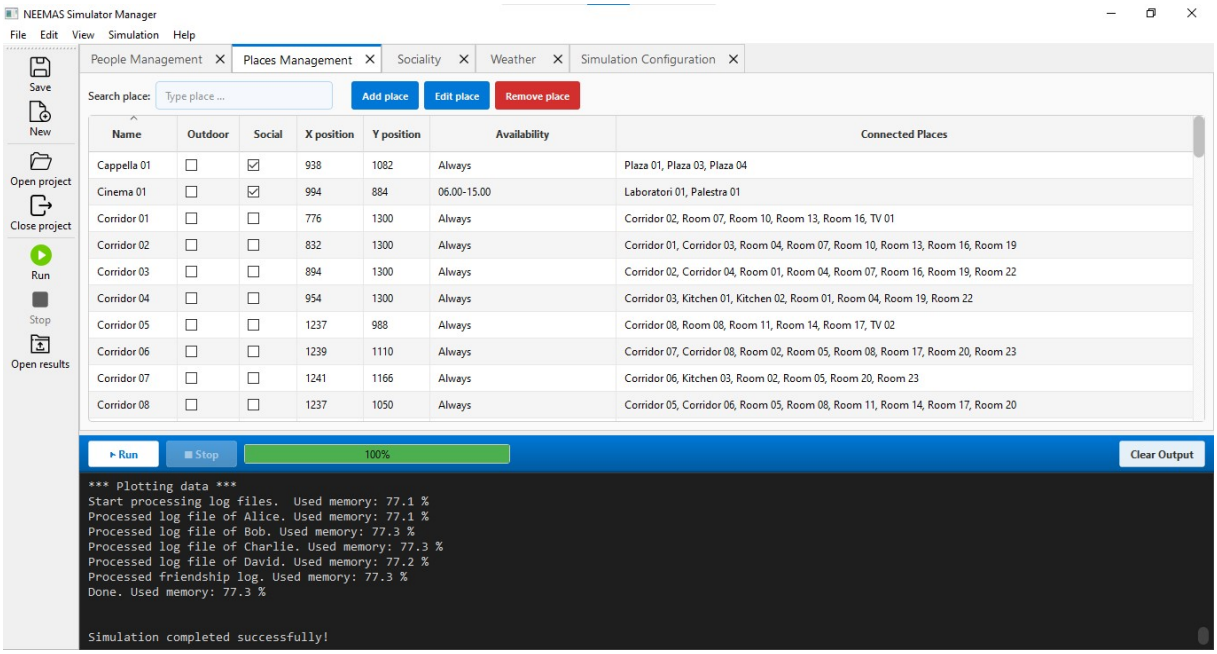

Figure 4.7: Places management window

When creating or editing a place, the *Place Editor* window is displayed, as shown in Figure 4.8. This interface allows users to configure all the properties associated with a location. Basic information such as the place name and spatial coordinates can be specified, together with additional attributes indicating whether the location is social or outdoor.

The editor also provides a mechanism for defining the temporal availability of the place. Users can specify the hours during which the location is accessible by selecting the corresponding time slots, enabling the simulation of environments whose accessibility varies throughout the day.

The lower section of the window is dedicated to the definition of connections between places. Since the environment is modeled as a graph, each place can be connected to one or more neighboring locations. For every connection, the corresponding distance can be specified, allowing the simulator to represent movement costs and navigation paths within the environment. Through these tools, users can construct complex and realistic indoor environments.

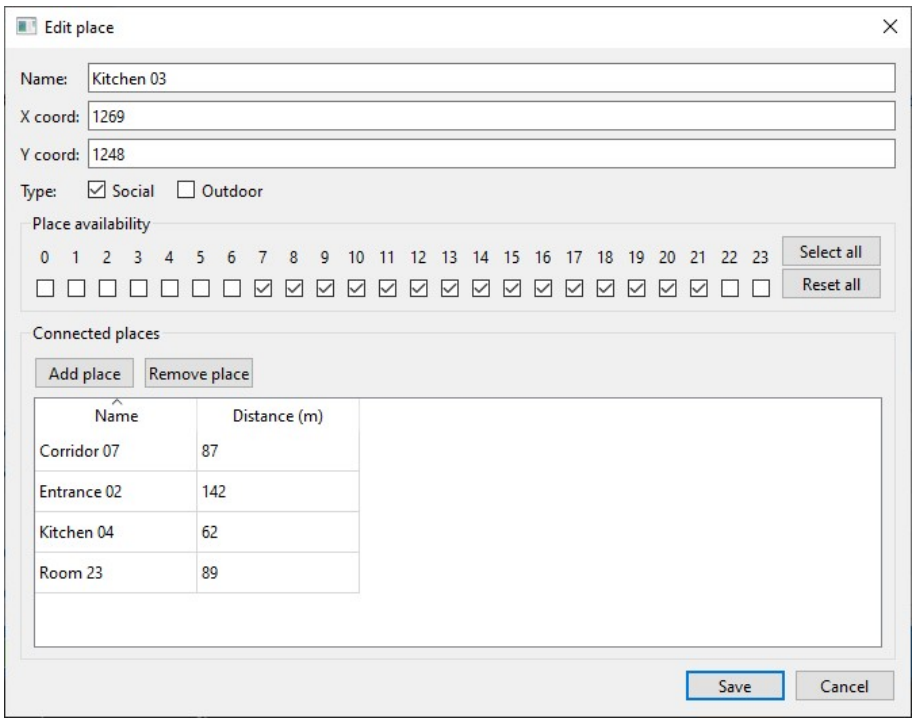

Figure 4.8: Place creation and editor window

4.4.4. Social Configuration

The *Sociality* tab provides the tools required to configure the social relationships between agents. As shown in Figure 4.9, the interface is centered around two symmetric matrix-based representations: the *compatibility matrix* and the *initial friendship matrix*. Together, these structures define the social dynamics that influence interactions during the simulation.

The compatibility matrix represents the affinity between pairs of agents. Since compatibility reflects personal characteristics rather than interaction history, its values remain fixed throughout the simulation. The initial friendship matrix represents the initial strength of social relationships between agents. Unlike compatibility, friendship values are dynamic and evolve over time as agents interact with one another. Positive interactions

may strengthen existing relationships, whereas prolonged periods without interaction can lead to their gradual decay.

The graphical interface allows users to easily check and modify relationships between all pairs of agents in a compact and intuitive manner. This visualization provides an immediate overview of the social structure of the simulated population and facilitates the creation of complex scenarios.

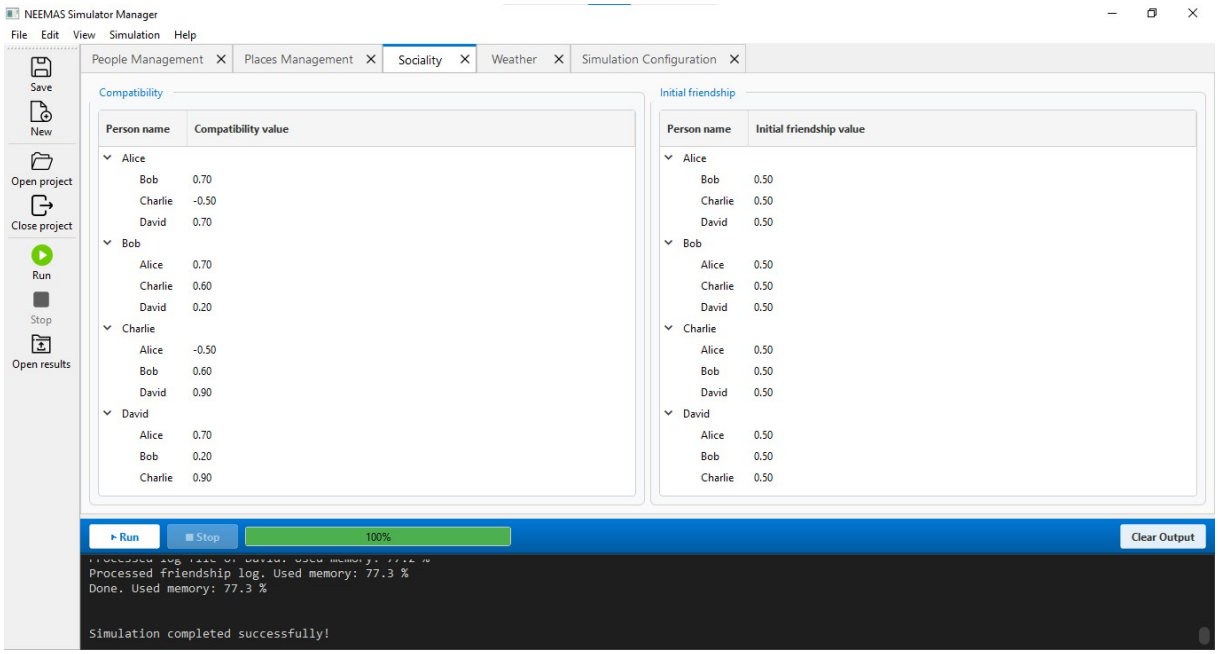

Figure 4.9: Sociality management window

4.4.5. Simulation Management

When managing the simulation, the *Simulation Configuration* and the *Weather* tabs provide information about the project and simulation settings. As shown in Figure 4.10, the *Simulation Configuration* tab serves as the interface for configuring the simulation before execution. From this interface, the user can view the project name, the path of the associated project directory and other relevant configuration details.

This tab enables also the loading of a new simulation map in JSON format, replacing the currently active map, as well as the selection of a different weather data file. In addition, the user can configure the start and end date and time of the simulation through a graphical calendar interface. To ensure consistency between the simulation and the available environmental data, date and time selection is restricted to periods for which weather information is available in the database. For this reason, the simulation can only be executed within the temporal range covered by the selected weather dataset.

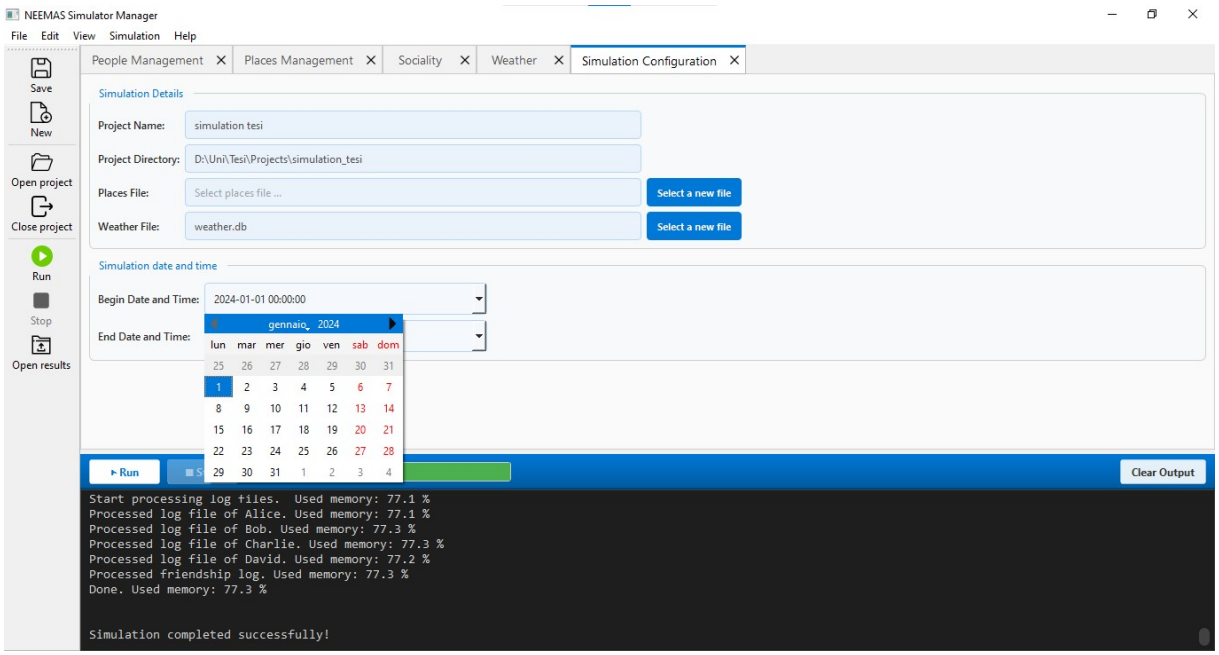

Figure 4.10: Simulation configuration window

The *Weather* tab, shown in Figure 4.11 displays the weather records used by the simulator. The table contains time-stamped environmental observations, including variables such as temperature, humidity and precipitation. These data are imported from the external weather file database and are used to reproduce realistic environmental conditions throughout the simulation.

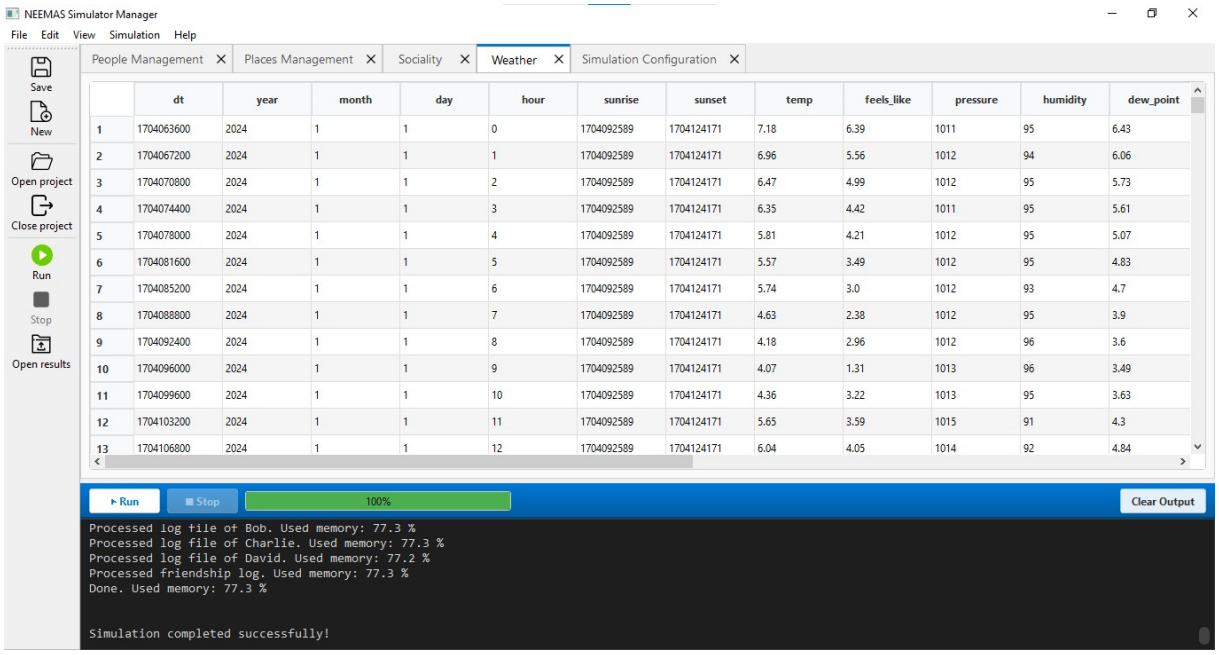

Figure 4.11: Weather window

4.4.6. Simulation Execution

Simulation execution is managed through a sliding panel located at the bottom of the interface, which remains accessible from all sections of the application. The interface is shown in Figure 4.12. This panel serves as the primary control area for monitoring and managing simulation runs.

The simulation can be started using the dedicated *Run* button. During execution, runtime messages and status updates are displayed in the log area, while a progress bar provides a visual indication of the simulation's completion status. The user may also interrupt the execution at any time through the corresponding *Stop* control.

In addition to execution controls, the panel provides other functionalities. A *Clear* button allows the user to remove the contents of the log window, improving readability during subsequent runs. Moreover, an *Output Folder* button provides direct access to the folder containing the simulation outputs once the execution has completed.

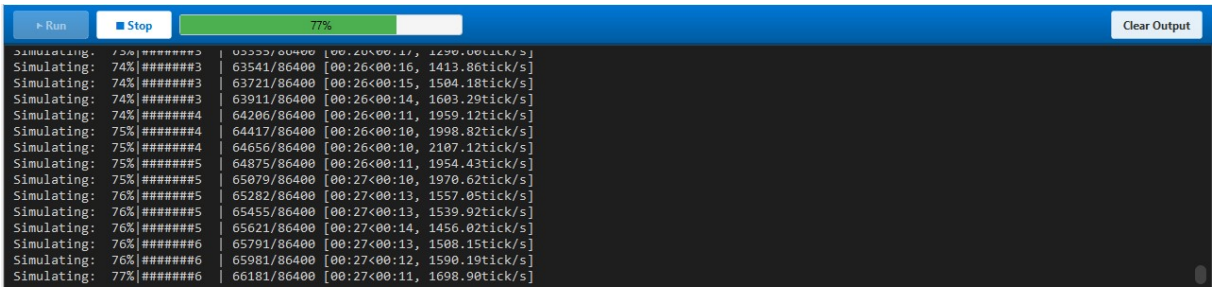

Figure 4.12: Simulation execution interface

5 | Experimental Results

5.1. Questionnaire Analysis

The questionnaire was filled by approximately 70 participants between December 2025 and January 2026 in the cities of Verona and Bolzano. Responses were collected through both online and paper-based questionnaires.

5.1.1. Questionnaire Digitization

While the responses submitted on the online form through Google Forms were immediately available in digital format, the paper questionnaires required an additional digitization step. To address this, a custom Python pipeline was developed to process scanned questionnaire images and extract structured responses. The extraction process relied on the Gemini 3.1 Pro model, accessed through API calls, which converted the scanned forms into a machine-readable format. The resulting data were automatically stored in an Excel spreadsheet to facilitate subsequent analysis and integration with the online data.

To ensure the reliability of the digitization process, a subset of the automatically extracted responses were verified manually inspecting the questionnaires. This validation step confirmed the correctness of the conversion and allowed potential errors to be identified, increasing the quality of the resulting dataset.

5.1.2. Questionnaires Analysis

After the digitization of the questionnaires, several approaches were considered for the generation of personas from the collected data, as explained in the Methodology part. A direct manual analysis using spreadsheets offered a transparent and highly interpretable workflow, but it was time-consuming and limited in its ability to identify less obvious relationships among the responses. Traditional machine learning techniques were also evaluated; however, their effectiveness is strongly dependent on the availability of sufficiently large datasets, which was not the case for the present study.

For these reasons, a Large Language Model (LLM)-based approach was selected. Recent advances in LLM capabilities make them particularly suitable for extracting structured insights from relatively small datasets while maintaining a high degree of interpretability. In particular, ChatGPT 5.5 and Gemini 3.1 Pro were employed to support the extraction and synthesis of persona characteristics from the questionnaire responses. The outputs generated by the models were subsequently reviewed and compared against manually analyzed samples to ensure consistency and plausibility. This combination of automated analysis and human verification provided a practical balance between efficiency, reliability and interpretability.

5.2. Identified Personas

Based on the analysis of the questionnaire responses, four representative personas were identified. These personas were derived by grouping participants with similar characteristics, habits and daily routines. Rather than representing specific individuals, each persona captures a recurring behavioral pattern observed within the collected data and serves as a reference profile for the configuration of agents within the simulation.

The four identified personas profiles are presented in detail in the following paragraphs.

5.2.1. Maria: Regular Routine and Socially Integrated

The first persona is Maria, a 74-year-old retired teacher living independently in her home. This is the most common behavioral profile identified in the questionnaire data. Individuals belonging to this group follow regular daily routines, maintain an active lifestyle and place high importance on social interactions.

Daily habits: Meals are typically consumed at regular times with breakfast at 08:00, lunch at 13:00 and dinner at 19:30, with meal durations ranging from 20 to 30 minutes. Drinking during the day outside meals occurs frequently, averaging five to seven drinking events per day, with low to no night time drinking.

Sleep: Individuals generally go to bed around 23:00 and sleep for 7-8 hours. Night time awakenings are limited and occasional daytime naps of 20-30 minutes may occur.

Sociality and activities: This persona is characterized by a large social network, usually including more than five people. Social interaction is considered highly important. Common activities include indoor and outdoor activities, gathering with others at home, in bars, restaurant and other social places.

Preferred environments: The preferred environments during the day of this kind of persona are: the kitchen, the living room and outdoor spaces.

5.2.2. Teresa: Domestic and Routine-Oriented

The second persona is Teresa, an 81-year-old widow living independently in a retirement residence. This persona is characterized by a regular and predictable daily routine, with most activities taking place within the residence environment. Sociality is still present, but limited compared to Persona A.

Daily habits: Meals are consumed at regular times with breakfast at 07:00, lunch at 12:00 and dinner at 19:00, typically lasting around 20 minutes. A snack is common in the afternoon. Hydration occurs regularly, about 4 to 5 times throughout the day, while night-time drinking is rare.

Sleep: Individuals generally go to bed around 22:00 and sleep for 7-8 hours, usually with no daytime naps. One or two night-time awakenings may occur, to go to the bathroom or drinking.

Sociality and activities: Social interactions are mainly limited to family members, with three to five regular contacts on average. Sociality remains important, although less central than in Persona A. The activities of this persona are typically conducted at home, like watching television, reading, cooking and household tasks.

Preferred environments: Most of the time is spent in the kitchen and living room, reflecting the domestic nature of this profile.

5.2.3. Edoardo: Shifted Circadian Rhythm

The third persona is Edoardo, a 75-year-old retired craftsman living in his home with his wife, who naturally follows an early daily routine. This persona is characterized by a shifted circadian rhythm, with most daily activities concentrated in the early hours of the day. The questionnaire data also revealed the presence of a complementary profile characterized by a delayed daily routine, with individuals tending to wake up and go to sleep later. However, the early-riser profile was selected as the representative persona because it was more frequently observed in the collected responses.

Daily habits: Breakfast is typically consumed between 06:30 and 07:00, followed by lunch around 12:00 and dinner between 18:00 and 18:30. Morning and afternoon snacks are relatively common. Hydration occurs regularly throughout the day, with little to no

night-time drinking.

Sleep: Individuals generally go to bed between 21:30 and 22:00 and wake up between 05:30 and 06:00, with around 7-8 hours of sleep. Short naps of 15-20 minutes in the afternoon are quite common in this persona.

Sociality and activities: Social interaction is moderately high and is often concentrated during morning hours. Common activities are distributed across both indoor and outdoor environments

Preferred environments: The preferred environments during the day of this kind of persona are: the kitchen, the living room and outdoor spaces, exhibiting a good mobility throughout the day.

5.2.4. Assunta: Fragmented Sleep and Afternoon Nap

The fourth persona is Assunta, an 85-year-old living in a retirement residence, affected by urinary incontinence. This persona is characterized by irregular sleep patterns, frequent nocturnal awakenings and a tendency to compensate the reduced night-time rest with daytime naps. This persona represents a minority that is relevant for the simulator, for this reason was selected in the analysis.

Daily habits: Meals are consumed at regular times with breakfast at 08:00, lunch at 13:00 and dinner between 19:30 and 20:00. Meals have a longer duration of about 30-40 minutes. Snacks are typical, particularly in the afternoon. This persona drinks regularly during the day with night-time drinking episodes occurring 1-2 times. Bathroom usage is relatively frequent, including 2-4 nocturnal awakenings.

Sleep and rest: Night-time sleep is characterized by multiple awakenings, with total sleep often falling below the desired duration. Individuals typically fall asleep between 23:00 and 23:30 and require 15-30 minutes to return to sleep after awakenings. Daytime compensation is common in the form of daily naps lasting between 45 and 90 minutes.

Sociality and activities: Social interaction is limited, with a moderate importance rating. Daily activities are mostly low-intensity and include home activities like watching television or reading.

Preferred environments: Time is predominantly spent in the bedroom, living room and kitchen, reflecting a largely indoor and sedentary routine.

5.3. Simulations Results

After presenting the theoretical foundations of the NeeMAS simulator, describing the development of the graphical interface and identifying the representative personas from the questionnaire data, this section focuses on the practical evaluation of the system. The objective is to verify that the simulator and the developed interface operate correctly and produce the expected results.

A total of six simulation scenarios are considered. Four simulations model the identified personas individually, allowing their behavioral patterns to be analyzed in an isolated scenario. One additional simulations examine a multi-agent environment, with particular attention to the evolution of social relationships. Finally, one simulation evaluates the wandering behavior implemented in the simulator, in which an agent moves through the environment without a specific goal or destination.

Simulation Environment

The four individual personas and the wandering simulations are conducted within a virtual apartment environment as shown in Figure 5.1, which provides a common setting for comparing the behaviors associated with the different personas.

Figure 5.1: Simulated apartment floor plan

The apartment is designed as a two-bedroom apartment representing a simplified shared living environment, similar to those commonly found in retirement residences. Each bedroom includes a private bathroom and is connected to the rest of the apartment through a central hallway leading to the living area and the main entrance. The kitchen and living room are organized as a single open-plan space, providing access to both a balcony and a utility room containing the washing machine and other household appliances. This layout was selected because it includes all the essential spaces required to satisfy the physiological and social needs modeled by the simulator while remaining sufficiently compact to facilitate the analysis of agent behavior.

When loaded into the simulator, the apartment is represented as a graph of connections between locations, as shown in Figure 5.2. In this representation, each node corresponds to a location within the apartment, while edges define the possible connections and movements between them. This structure allows the simulator to model spatial navigation in a simplified and efficient way.

The graph also distinguishes between different types of locations supported by the simulator, including social and private spaces, as well as indoor and outdoor environments. These categories are used to reflect functional differences between areas and to influence agent behavior within the simulation.

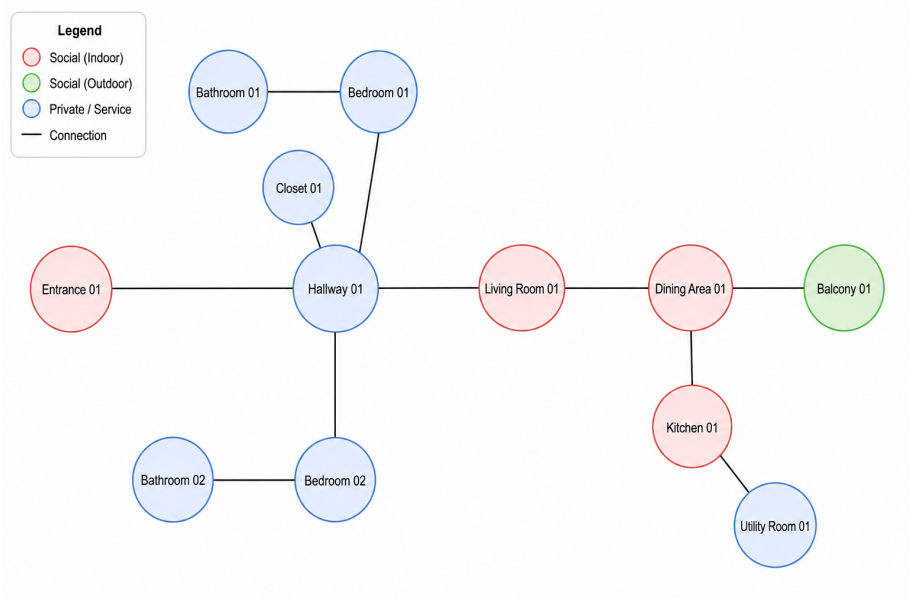

Figure 5.2: Simulated apartment graph visualization

5.3.1. Maria: Active and Socially Integrated

The first simulation involves Maria, representing the active and socially integrated persona. She typically wakes up at 07:00 and has breakfast at around 08:00, with a duration of approximately 20 minutes. Lunch is consumed at 13:00 and dinner at 20:00, with both main meals lasting around 30 minutes.

During the day, Maria maintains regular hydration habits, drinking approximately six times per day and uses the bathroom around six times daily. Her sleep schedule is stable, with bedtime at approximately 23:00 and a total sleep duration of about 8 hours. She usually wakes up once during the night to use the bathroom before returning to sleep.

The simulation is executed through the graphical interface, configured with the specified schedules within the simulated apartment. As shown in Figure 5.3 and Figure 5.4, the defined schedules are correctly followed by the agent. Figure 5.3 illustrates the temporal evolution of the sleep and hunger needs throughout the day. The sleep need increases progressively during the day and reaches its maximum during the night, when Maria goes to sleep. In contrast, the hunger need exhibits a cyclical pattern, decreasing after each meal and gradually increasing again until the next scheduled eating time.

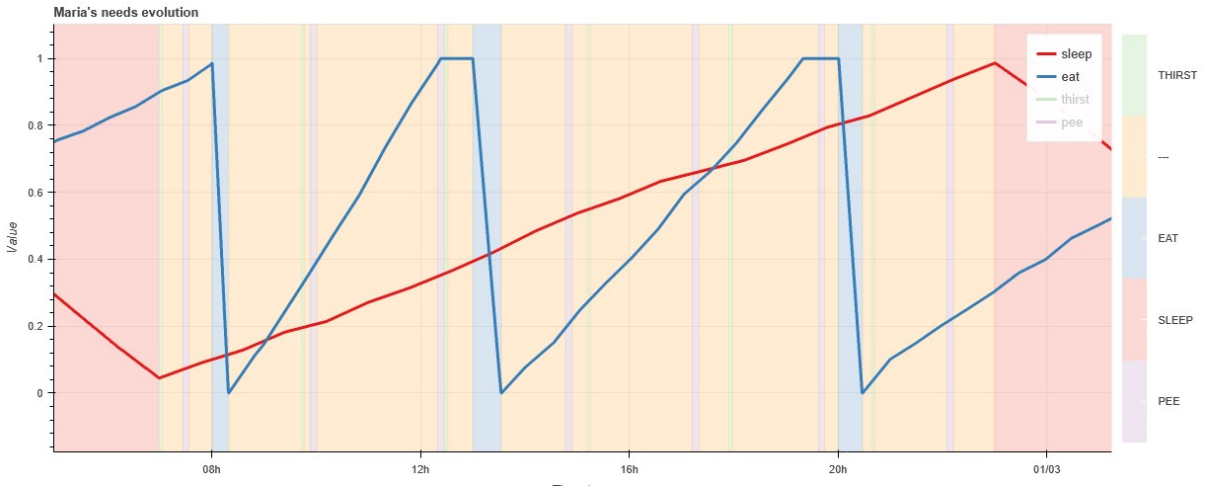

Figure 5.3: Maria's sleep and eat needs

Figure 5.4 shows the evolution of thirst and bathroom-related needs throughout the day. As can be observed, both needs increase over time and are regularly satisfied through the agent's scheduled actions. During sleep periods the rate of increase is reduced, however as scheduled, Maria wakes up once during the night to use the bathroom.

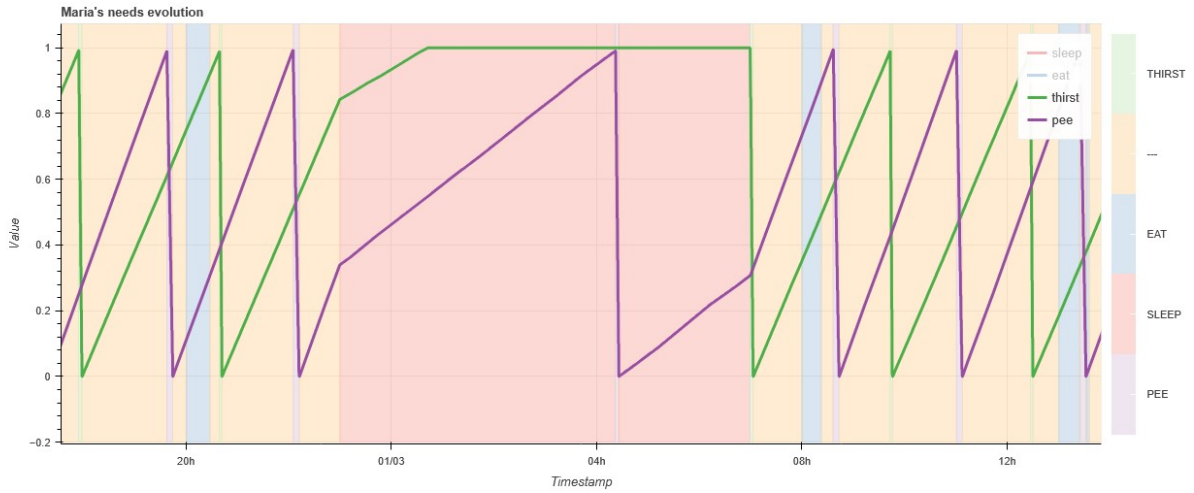

Figure 5.4: Maria's drink and bathroom needs

Figure 5.5 illustrates Maria's movements throughout the simulation. As can be observed, the agent moves between the different locations of the apartment in order to satisfy her physiological and daily needs. The movement patterns are consistent with the scheduled activities and need-driven behaviors defined for this persona. Travel times between locations depend on the distances specified in the environment graph, resulting in realistic transitions between different areas of the apartment.

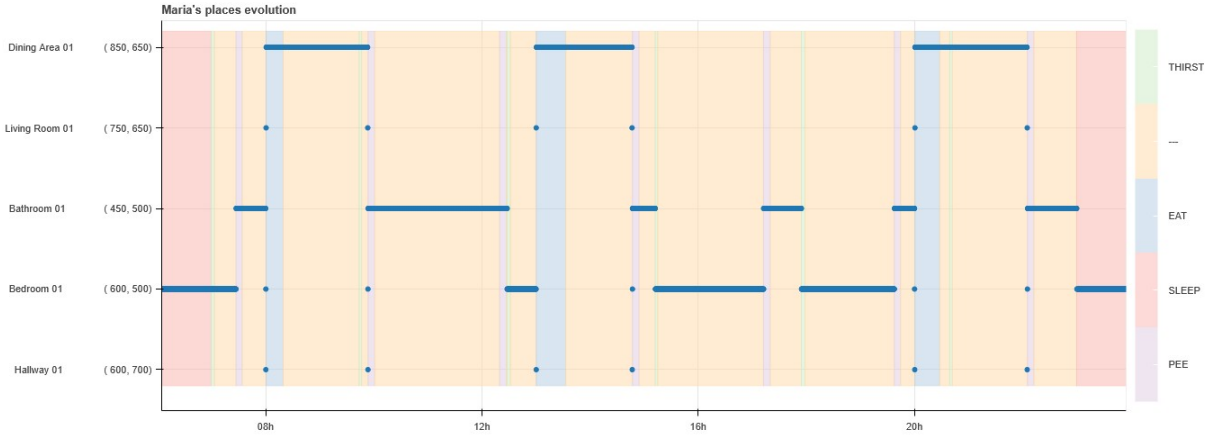

Figure 5.5: Maria's movements during the day

5.3.2. Teresa: Domestic and Routine-Oriented

The second simulation involves Teresa, representing the domestic and routine-oriented persona. Her daily routine is highly regular and predictable, with most activities taking place within the apartment. Teresa wakes up at 06:00 and has breakfast at 07:00, she follows a consistent meal schedule with lunch at 12:00 and dinner at 19:00. In the afternoon

Teresa is used to have a snack at 16:00. Hydration and bathroom usage occur regularly throughout the day and she usually wakes up one or two times during the night to drink or going to the bathroom. Social activities are limited compared to the previous persona.

As shown in Figure 5.6, the sleep and hunger needs follow the expected daily pattern. The sleep need gradually increases during the day and is satisfied during the night, when Teresa follows her sleep schedule. Similarly, the hunger need increases between meals and decreases when breakfast, lunch and dinner are consumed according to the predefined routine. From the graph it can be seen that the snack is eaten in the afternoon.

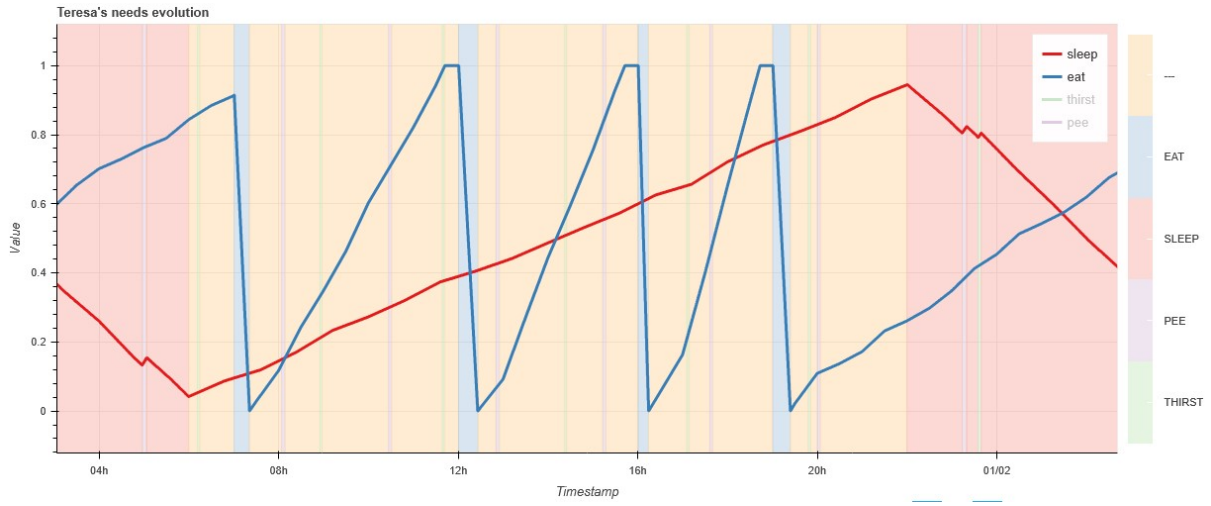

Figure 5.6: Teresa's sleep and eat needs

Figure 5.7 illustrates the evolution of thirst and bathroom needs. Both needs exhibit a regular cyclical behavior, increasing over time and being satisfied through drinking and bathroom visits. As it can be seen in the graph, Teresa wakes up two times during the night, one time for drinking and one time to go to the bathroom, following the scheduled behavior loaded through the graphical user interface.

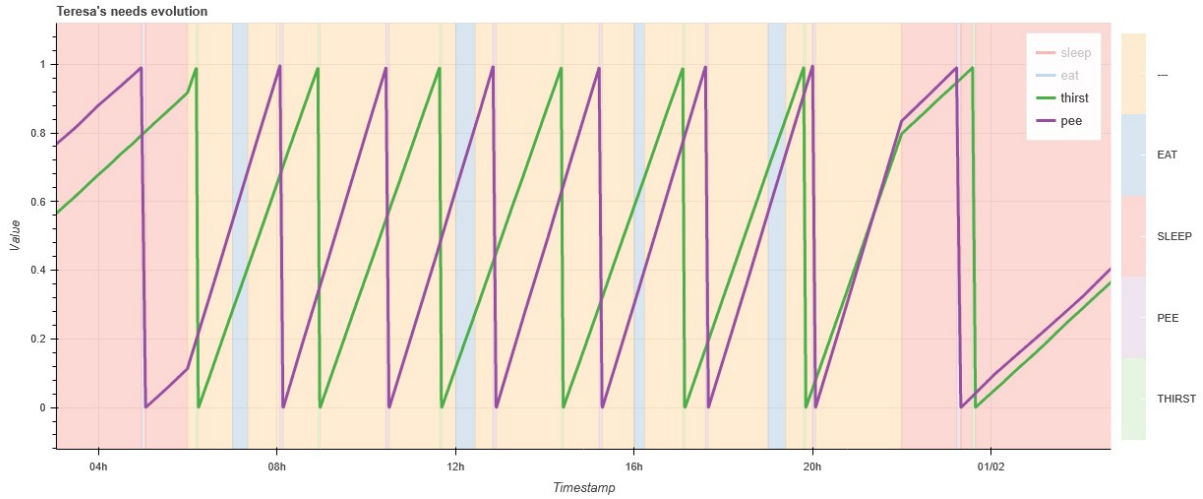

Figure 5.7: Teresa's drink and bathroom needs

Finally, Figure 5.8 shows Teresa's movements within the apartment. Her movement patterns are concentrated around a small set of locations, primarily the dining area, bathroom and living room. The resulting trajectories reflect the domestic and routine-oriented nature of this profile and confirm that the simulator correctly reproduces the expected behavioral characteristics.

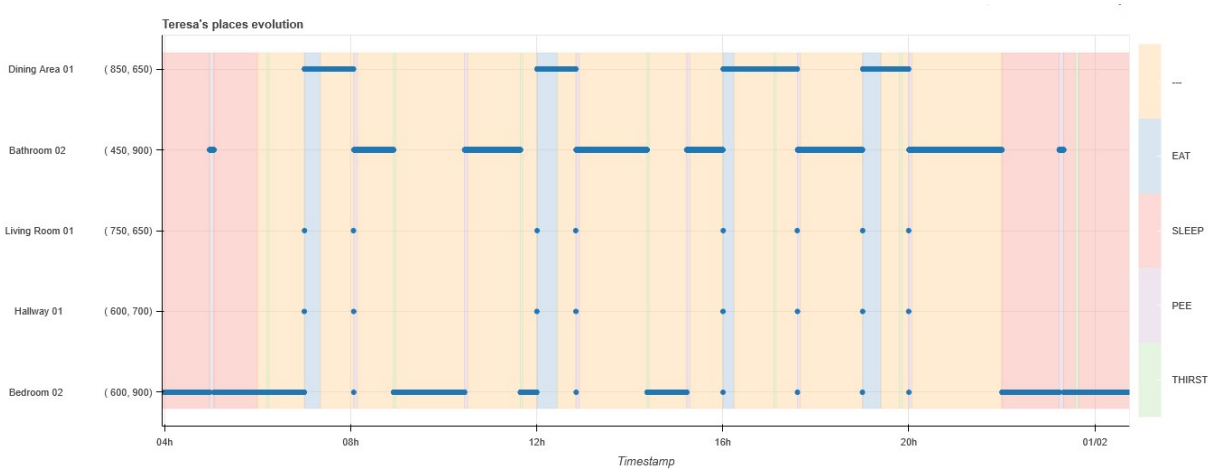

Figure 5.8: Teresa's movements during the day

5.3.3. Edoardo: Shifted Circadian Rhythm

The third simulation involves Edoardo, representing the shifted circadian rhythm persona. Unlike the previous profiles, Edoardo follows an earlier daily schedule, waking up and performing most of his activities during the first part of the day. Breakfast is consumed

early in the morning, followed by lunch at midday and dinner in the early evening. His routine also includes a short afternoon nap.

Figure 5.9 shows the evolution of sleep and hunger needs throughout the simulation. The sleep need increases during the day and is satisfied earlier waking up at 5:00, reflecting Edoardo's advanced circadian rhythm. This kind of persona usually has a nap after lunch, in this case Edoardo sleeps for about 30 minutes at 15:00. The hunger need follows the expected cyclical pattern, with breakfast at 6:00, lunch at 12:00 and dinner at 18:00.

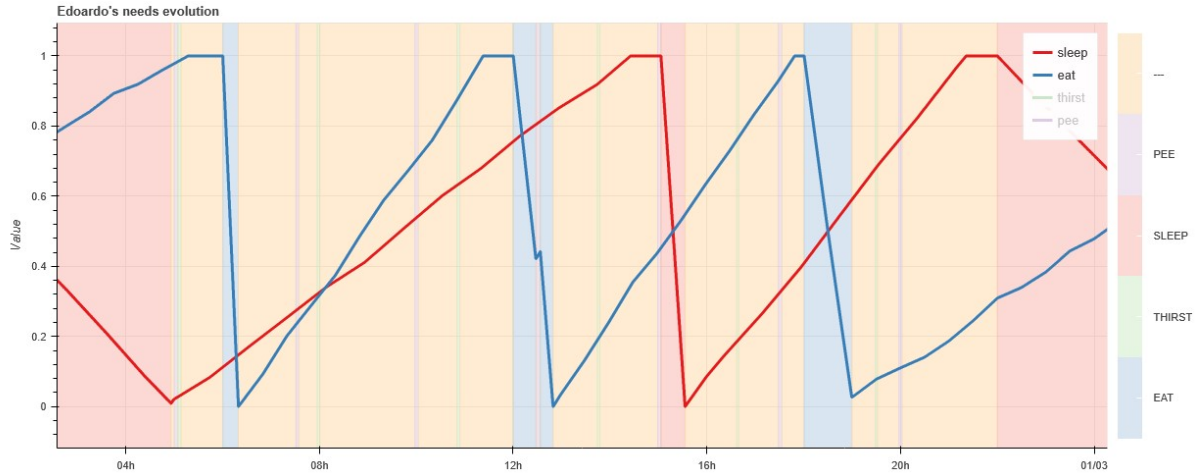

Figure 5.9: Edoardo's sleep and eat needs

Figure 5.10 illustrates the behaviour of thirst and bathroom needs. Both needs exhibit regular fluctuations and are periodically satisfied through drinking and bathroom visits. Due to the absence of significant night activity, the two needs remain stable during the sleeping period.

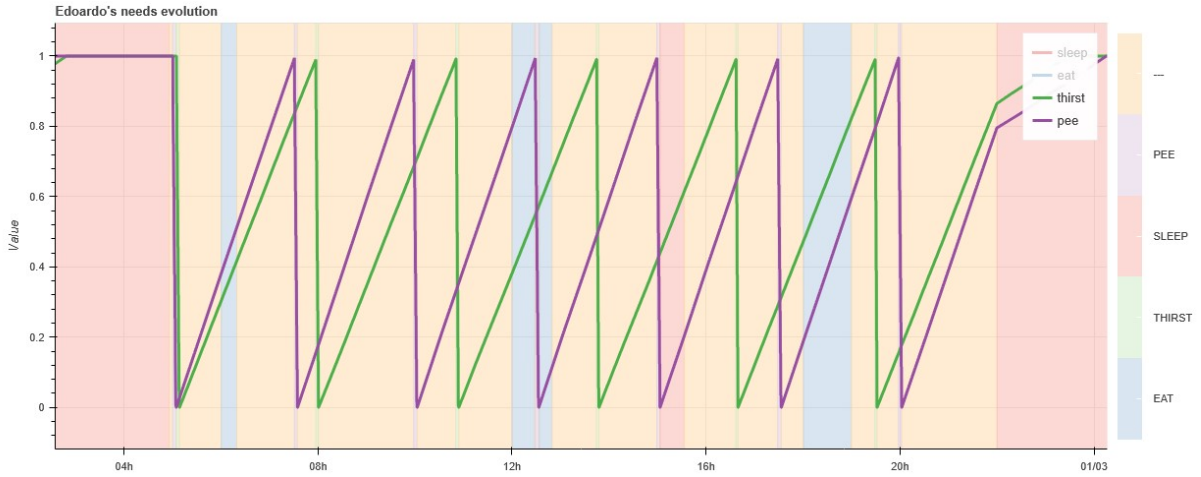

Figure 5.10: Edoardo's drink and bathroom needs

Finally, Figure 5.11 presents Edoardo's movements within the apartment. The majority of his activities occur during the morning and afternoon, resulting in a concentration of movements during these periods. As with the previous simulations, the agent moves between locations according to the need-driven decision process and the spatial constraints defined by the environment graph.

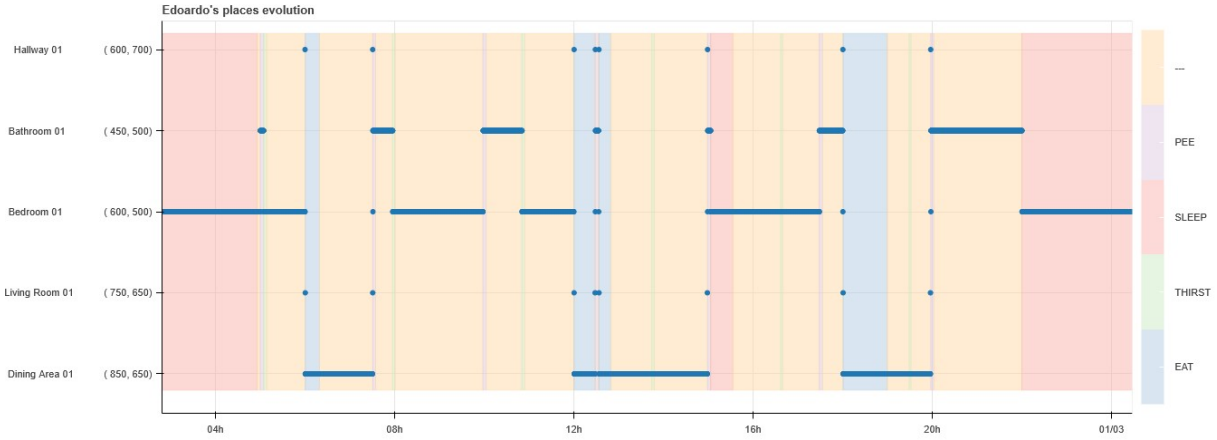

Figure 5.11: Edoardo's movements during the day

5.3.4. Assunta: Fragmented Sleep and Afternoon Nap

The fourth simulation involves Assunta, representing the fragmented sleep and afternoon nap persona. Assunta suffers from urinary incontinence, resulting in more frequent bathroom visits, particularly during the night. This profile is characterized by frequent nocturnal awakenings, reduced sleep quality and the tendency to compensate for nighttime sleep through an afternoon nap. Compared to the previous personas, she exhibits a generally less active lifestyle, with most activities taking place indoors.

Figure 5.12 shows the evolution of sleep and hunger needs throughout the simulation. The sleep need follows a more irregular pattern than in the previous scenarios, reflecting the interruptions occurring during the night. She wakes up at 7:00 in the morning and has an afternoon nap of about 50 minutes at 14:00 to satisfy the accumulated sleep need. The hunger need follows the expected cyclical behaviour, with breakfast at 8:00, lunch at 13:00 and dinner at 20:00. After the nap, at 16:00 Assunta goes to the dining area to have a snack.

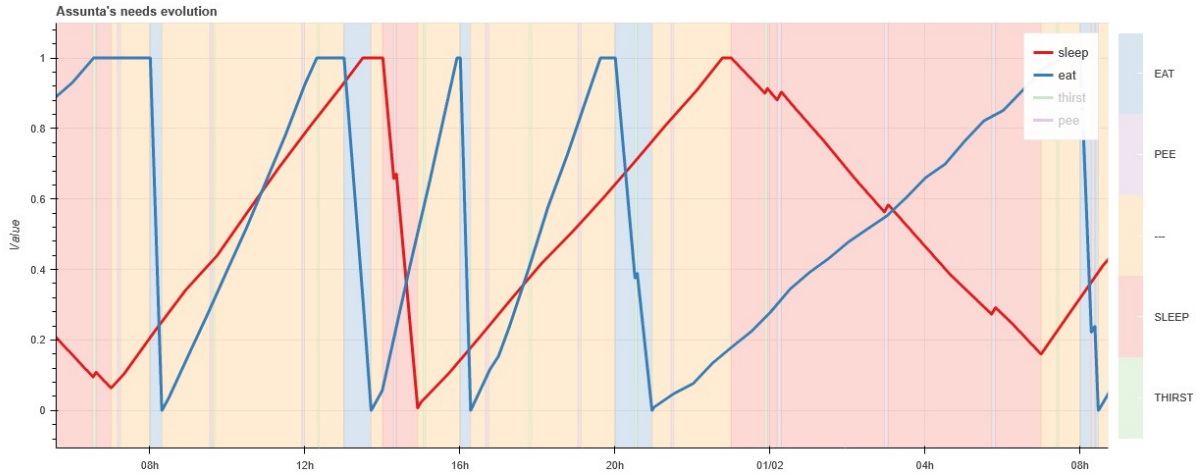

Figure 5.12: Assunta's sleep and eat needs

Figure 5.13 illustrates the evolution of thirst and bathroom needs. Thirst follows a regular daily pattern, with about one drinking event during the night. Bathroom need on the other hand reaches its threshold multiple times during the night due to urinary incontinence. These nocturnal bathroom visits interrupt the sleep cycle and represent one of the defining characteristics of this persona, directly influencing both movement patterns and sleep dynamics.

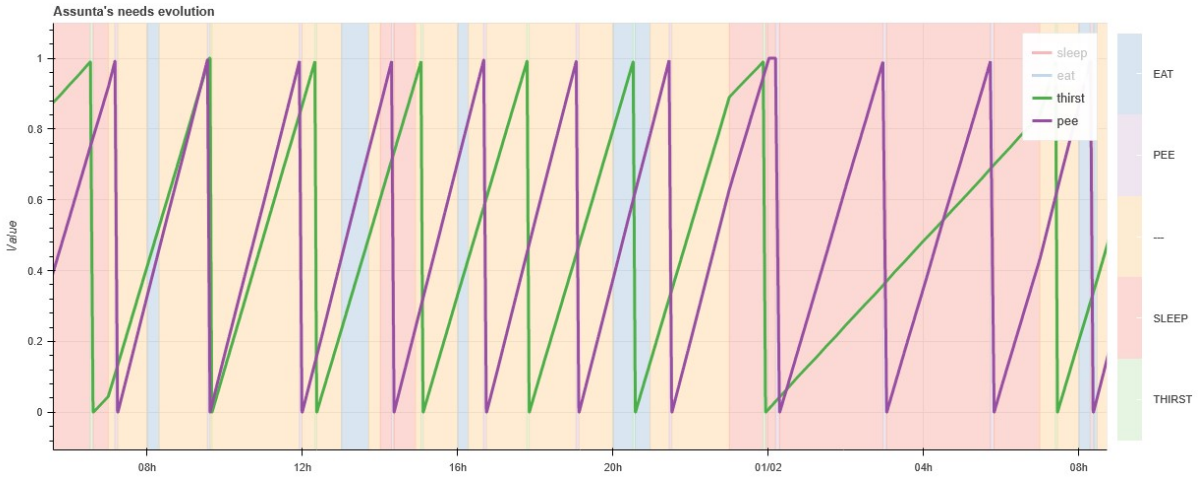

Figure 5.13: Assunta's drink and bathroom needs

Finally, Figure 5.14 presents Assunta's movements within the apartment. Movement is more concentrated around the bedroom, bathroom and dining area. Additional movements can be observed during the night due to frequent bathroom visits, while the afternoon nap introduces an additional period of inactivity during the day. The resulting

behaviour demonstrates the interface and simulator's ability to reproduce more complex daily routines characterized by fragmented sleep patterns and increased nocturnal activity.

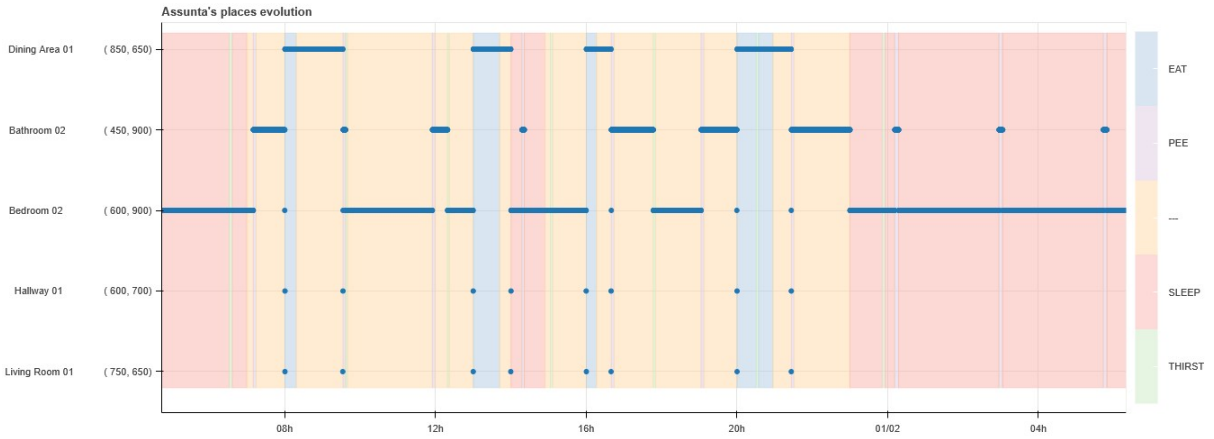

Figure 5.14: Assunta's movements during the day

5.3.5. Friendship Interaction Simulation

The following experiment evaluates the multi-agent and social interaction capabilities of the NeeMAS simulator configured through the graphical interface. In this scenario, the four personas introduced in the previous sections are simulated simultaneously within an extended environment representing a larger retirement residence. The initial friendship values are all set to a neutral value of 0.5. The initial compatibility values between agents are configured through the interface, as shown in Figure 5.15. The compatibility matrix contains both positive and negative values, allowing the simulation of different types of interpersonal relationships. For example, Teresa and Edoardo have the highest compatibility (1.0), while Maria and Assunta are assigned a negative compatibility (-0.7), representing an antagonistic relationship.

Figure 5.15: Compatibility matrix in the interface

During the simulation, the agents move independently throughout the apartment to satisfy their physiological needs while also seeking opportunities for social interaction. As illustrated in Figure 5.16, social interactions occur whenever two compatible agents remain in the same social location for a sufficient amount of time, in this case the social need is satisfied. In this experiment, the living room is the primary meeting point, where prolonged interactions between agents can be observed.

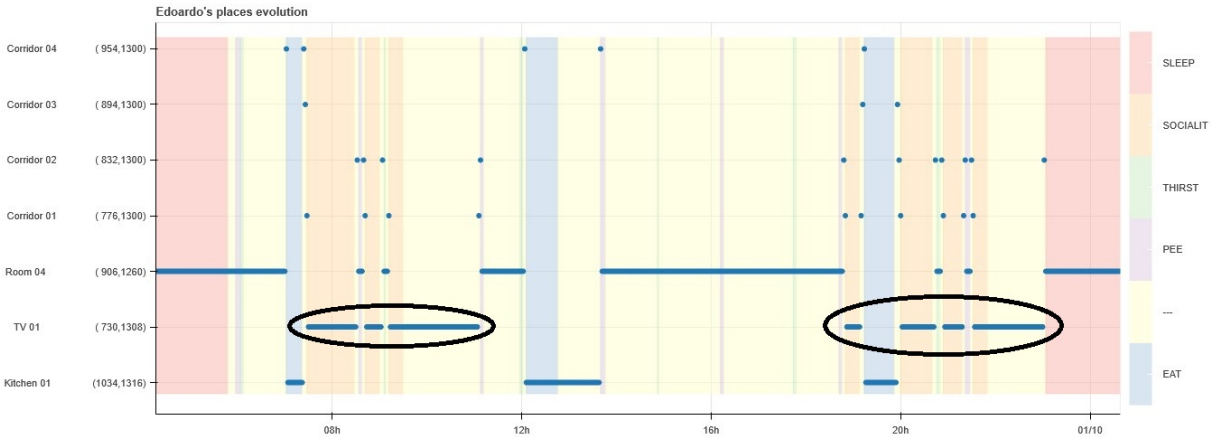

Figure 5.16: Person movements during sociality execution

Lastly, Figure 5.17 shows the evolution of the friendship matrix over a 25 day simulation showing the matrix every 5 days. Initially, all friendship values are neutral (0.5). As interactions accumulate, friendships evolve according to the defined compatibility values. The strongest relationship develops between Teresa and Edoardo, whose friendship increases faster than the other people, reflecting their maximum compatibility. Maria also develops positive relationships with Teresa and Edoardo, although at a slower rate due to their lower compatibility values. In contrast, the friendship between Maria and Assunta progressively decreases, almost reaching zero as a consequence of their negative compatibility. Finally the relationship between Edoardo and Assunta barely change from the neutral value due to their compatibility value close to zero. This experiment validates the ability to configure complex simulations involving social interactions through the interface, showing that both compatibility and repeated interactions contribute to the long-term evolution of social relationships between agents.

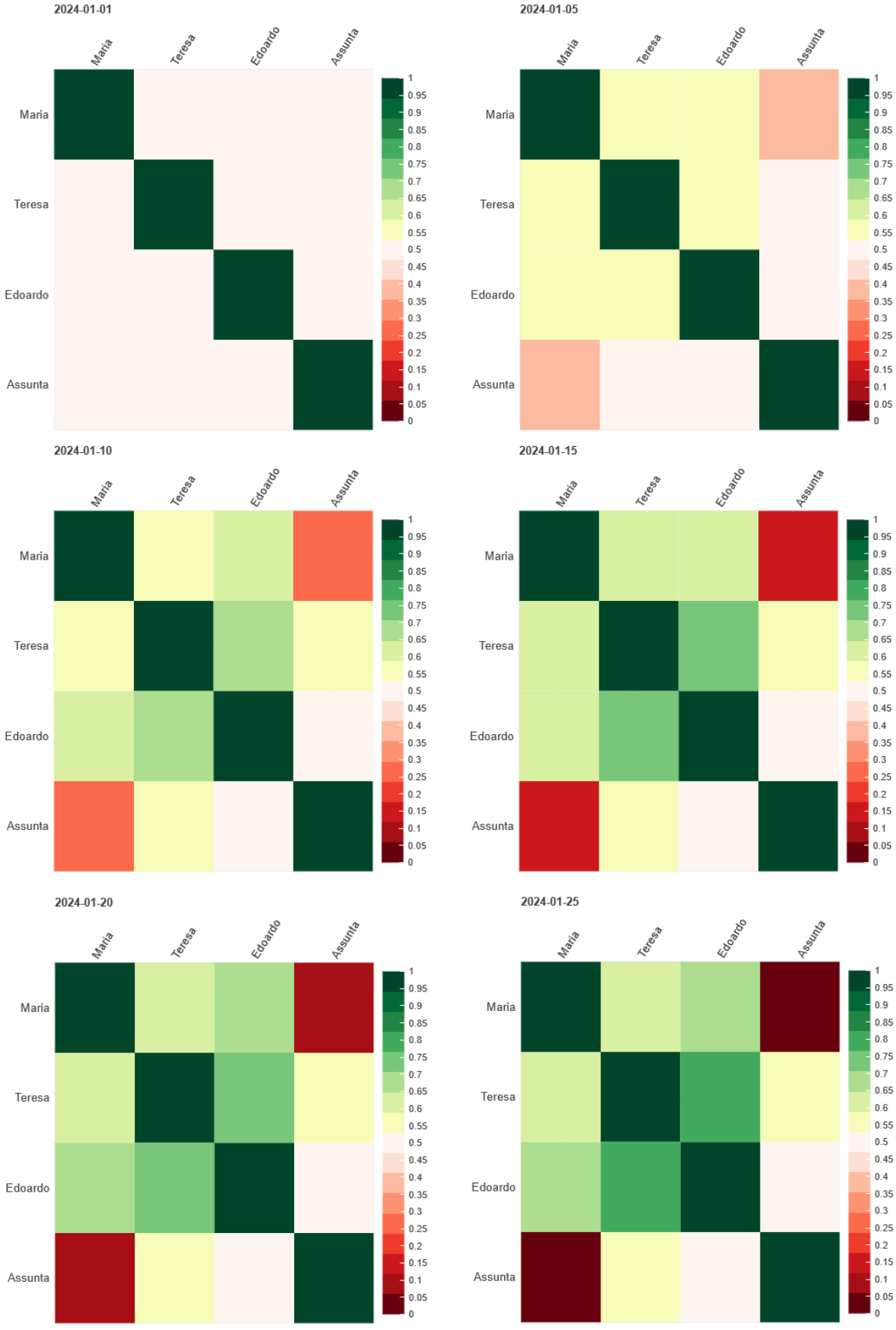

Figure 5.17: Friendship matrix evolution

5.3.6. Wandering Behaviour Simulation

The last experiment evaluates the agent's wandering capabilities within the NeeMAS simulator, configured through the graphical interface. The simulation uses the same apartment map adopted for the individual persona experiments, ensuring a consistent spatial setting across different behavioural scenarios. In this setup, the agent does not have a specific goal, but instead moves back and forth in familiar locations according to the activation of its wandering need. When the need exceeds its activation threshold, the agent starts moving; after reaching a location, it remains there for a predefined minimum amount of time before selecting the next destination.

Figure 5.18 shows the agent's movements during the simulation. The path highlights how the agent alternates between a selected set of locations when the wandering need is executed, producing the typical wandering movement patterns. Rather than following a predefined route, the agent shows non-deterministic trajectories characterized by repeated transitions between a limited set of locations.

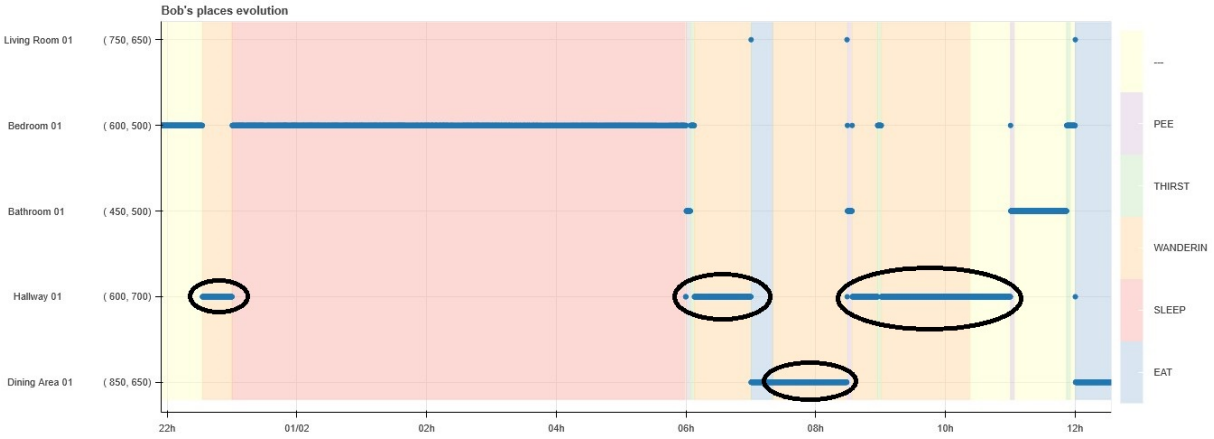

Figure 5.18: Agent's movements during wandering execution

Figure 5.19 illustrates the temporal evolution of the wandering need over the course of the simulation. The curve shows a cyclical pattern, where the need gradually increases during inactivity phases and decreases when the agent is executing wandering behaviour. This alternating trend explains the timing of movements observed in the movements graphs, as the wandering behavior is triggered when the need exceeds its activation threshold.

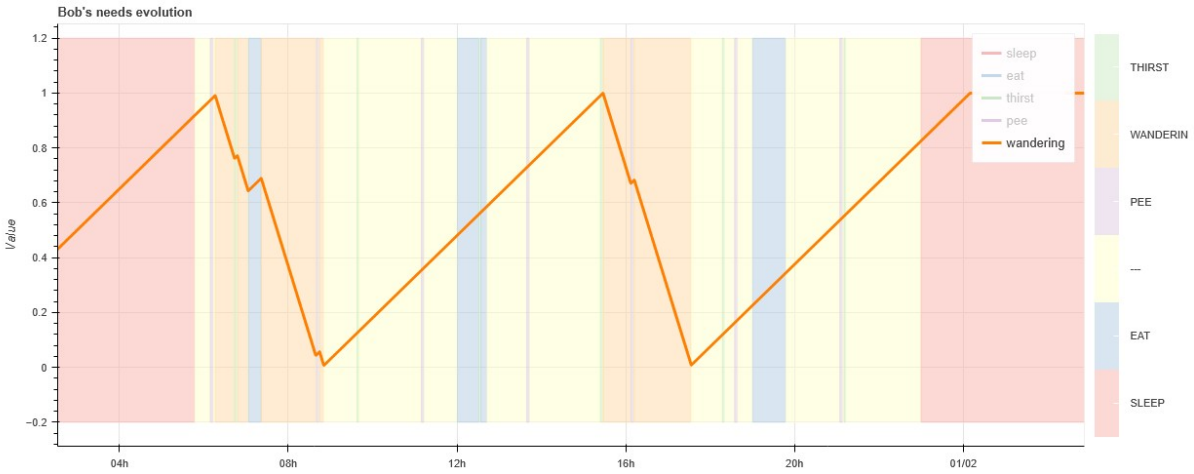

Figure 5.19: Wandering need during the simulation

The experimental results show that the interface and the NeeMAS simulator effectively generates non-deterministic motion patterns, which are particularly relevant for modelling behaviours associated with cognitive impairments such as dementia. Moreover, wandering contributes significantly to increasing the realism and diversity of simulated agent behaviors within the proposed framework.

6 | Conclusions and Future Work

This thesis presented the development of a graphical user interface designed to configure and control in an intuitive way NeeMAS, an agent-based behavioral simulator. After introducing the theoretical background about the simulator and system architecture, a set of four personas was defined from questionnaire data to configure the agents and validate the simulator interface.

The simulator was evaluated through several scenarios, including individual persona simulations, multi-agent social interactions and wandering behavior. The results confirmed the correct functioning of the system and the coherence between the parameters set through the interface and the generated behaviors. The main limitation of the current work lies in the necessary simplification of the real-world human behavior. In particular, human decision-making, emotional states and social dynamics are reduced to a set of variables, which, although effective for simulation purposes, cannot fully capture the complexity and variability of real individuals.

Future work will focus on improving the modeling of the agents and their social interactions, for example by introducing other needs and more complex mechanisms for relationship evolution. Additional work may include adding the visualization of agent movement through advanced 2D or 3D representations, improving interpretability and supporting a more intuitive analysis of simulation results.

In conclusion, NeeMAS and the designed interface provide a flexible framework for behavioral simulation, with potential applications in the training and validation of algorithms for human behavior analysis, assisted living systems and the study of human activity patterns in indoor and outdoor environments.

Bibliography

[1] N. Alshammari, T. Alshammari, M. Sedky, J. Champion, and C. Bauer. OpenSHS: Open Smart Home Simulator. *Sensors*, 17(5):1003, May 2017. ISSN 1424-8220. doi: 10.3390/s17051003. URL `https://www.mdpi.com/1424-8220/17/5/1003`.

[2] D. Bouchabou, S. M. Nguyen, C. Lohr, B. LeDuc, and I. Kanellos. A survey of human activity recognition in smart homes based on iot sensors algorithms: Taxonomies, challenges, and opportunities with deep learning. *Sensors*, 21(18), 2021. ISSN 1424-8220. doi: 10.3390/s21186037. URL `https://www.mdpi.com/1424-8220/21/18/6037`.

[3] K. Bouchard, A. Ajroud, B. Bouchard, and B. A. Simact: a 3d open source smart home simulator for activity recognition with open database and visual editor. *International Journal of Hybrid Information Technology (IJHIT),*, Volume 5:13–32, 07 2012.

[4] L. Cabañero, A. Perez-Vereda, C. Nugent, I. Cleland, R. Hervas, and I. González. A software tool and a metamodel for digital twins of inhabited smart environments. In J. Bravo, S. Ochoa, and J. Favela, editors, *Proceedings of the International Conference on Ubiquitous Computing & Ambient Intelligence (UCAmI 2022)*, pages 747–759, Cham, 2023. Springer International Publishing. ISBN 978-3-031-21333-5.

[5] S. Comai, A. Masciadri, D. Zuccarello, and F. Salice. Neemas: A need-based multi-agent simulator of human behavior for long-term drifts in smart environments. In J. Bravo and G. Urzáiz, editors, *Proceedings of the 15th International Conference on Ubiquitous Computing & Ambient Intelligence (UCAmI 2023)*, pages 88–99, Cham, 2023. Springer Nature Switzerland. ISBN 978-3-031-48642-5.

[6] D. J. Cook, A. S. Crandall, B. L. Thomas, and N. C. Krishnan. CASAS: A Smart Home in a Box. *Computer*, 46(7), July 2013. ISSN 0018-9162. doi: 10.1109/MC.2012.328.

[7] A. Fuller, Z. Fan, C. Day, and C. Barlow. Digital twin: Enabling technologies,

challenges and open research. *IEEE Access*, 8:108952–108971, 2020. doi: 10.1109/ACCESS.2020.2998358.

[8] M. Grammatikopoulou, I. Lazarou, V. Alepopoulos, L. Mpaltadoros, V. P. Oikonomou, T. G. Stavropoulos, S. Nikolopoulos, I. Kompatsiaris, and M. Tsolaki. Assessing the cognitive decline of people in the spectrum of ad by monitoring their activities of daily living in an iot-enabled smart home environment: a cross-sectional pilot study. *Frontiers in Aging Neuroscience*, Volume 16 - 2024, 2024. ISSN 1663-4365. doi: Katz1963. URL https://www.frontiersin.org/journals/aging-neuroscience/articles/10.3389/fnagi.2024.1375131.

[9] B. Ho, D. Vogts, and J. Wesson. A Smart Home Simulation Tool to Support the Recognition of Activities of Daily Living. In *Proceedings of the South African Institute of Computer Scientists and Information Technologists 2019*, SAICSIT '19, pages 1–10, New York, NY, USA, Sept. 2019. Association for Computing Machinery. ISBN 978-1-4503-7265-7. doi: 10.1145/3351108.3351132. URL https://dl.acm.org/doi/10.1145/3351108.3351132.

[10] C. Jiang and A. Mita. Automatic Daily Activity Schedule Planning for Simulating Smart House with Elderly People Living Alone. *The Impact of Digital Technologies on Public Health in Developed and Developing Countries*, 12157:171–183, May 2020. doi: 10.1007/978-3-030-51517-1_14. URL https://pmc.ncbi.nlm.nih.gov/articles/PMC7313293/.

[11] D. Jones, C. Snider, A. Nassehi, J. Yon, and B. Hicks. Characterising the digital twin: A systematic literature review. *CIRP Journal of Manufacturing Science and Technology*, 29:36–52, 2020. ISSN 1755-5817. doi: https://doi.org/10.1016/j.cirpj.2020.02.002. URL https://www.sciencedirect.com/science/article/pii/S1755581720300110.

[12] S. Katz. Studies of illness in the aged: The index of adl: A standardized measure of biological and psychosocial function. *JAMA*, 185(12):914, Sept. 1963. ISSN 0098-7484. doi: 10.1001/jama.1963.03060120024016. URL http://dx.doi.org/10.1001/jama.1963.03060120024016.

[13] S. Khan, H. Ali, and Z. Shah. Systematic analysis of smart homes: Current trends and future recommendations. *Cogent Engineering*, 11(1):2344452, 2024. doi: 10.1080/23311916.2024.2344452. URL https://doi.org/10.1080/23311916.2024.2344452.

[14] M. P. Lawton and E. M. Brody. Assessment of older people: Self-maintaining and

instrumental activities of daily living. *The Gerontologist*, 9(3 Part 1):179–186, Sept. 1969. ISSN 1758-5341. doi: 10.1093/geront/9.3_part_1.179. URL http://dx.doi.org/10.1093/geront/9.3_Part_1.179.

[15] J. W. Lee, S. Cho, S. Liu, K. Cho, and S. Helal. Persim 3D: Context-Driven Simulation and Modeling of Human Activities in Smart Spaces. *IEEE Transactions on Automation Science and Engineering*, 12(4):1243–1256, Oct. 2015. ISSN 1558-3783. doi: 10.1109/TASE.2015.2467353. URL https://ieeexplore.ieee.org/document/7254253.

[16] W. Lee, S. Cho, P. Chu, H. Vu, S. Helal, W. Song, Y.-S. Jeong, and K. Cho. Automatic agent generation for IoT-based smart house simulator. *Neurocomputing*, 209:14–24, Oct. 2016. ISSN 0925-2312. doi: 10.1016/j.neucom.2015.04.130. URL https://www.sciencedirect.com/science/article/pii/S092523121630580X.

[17] Y. Liu, L. Zhang, Y. Yang, L. Zhou, L. Ren, F. Wang, R. Liu, Z. Pang, and M. Deen. A novel cloud-based framework for the elderly healthcare services using digital twin. *IEEE Access*, 7:49088–49101, 2019. doi: 10.1109/ACCESS.2019.2909828.

[18] B. M. Mîndrut and C. A. Oprea. Experiential learning through controlling and monitoring a real-time 3d house using labview in a virtual laboratory. In *International Conference on Innovative Technologies and Learning*, 2020. URL https://api.semanticscholar.org/CorpusID:227129920.

[19] J. Park, M. Moon, S. Hwang, and K. Yeom. CASS: A Context-Aware Simulation System for Smart Home. In *5th ACIS International Conference on Software Engineering Research, Management & Applications (SERA 2007)*, pages 461–467, Aug. 2007. doi: 10.1109/SERA.2007.60. URL https://ieeexplore.ieee.org/abstract/document/4296972.

[20] V. C. Pezoulas, D. I. Zaridis, E. Mylona, C. Androutsos, K. Apostolidis, N. S. Tachos, and D. I. Fotiadis. Synthetic data generation methods in healthcare: A review on open-source tools and methods. *Computational and Structural Biotechnology Journal*, 23:2892–2910, 2024. ISSN 2001-0370. doi: https://doi.org/10.1016/j.csbj.2024.07.005. URL https://www.sciencedirect.com/science/article/pii/S2001037024002393.

[21] M. Skubic, G. Alexander, M. Popescu, M. Rantz, and J. Keller. A smart home application to eldercare: current status and lessons learned. *Technology and Health Care: Official Journal of the European Society for Engineering and Medicine*, 17(3):183–201, 2009. ISSN 0928-7329. doi: 10.3233/THC-2009-0551.

[22] B. K. Sovacool and D. D. Furszyfer Del Rio. Smart home technologies in europe: A critical review of concepts, benefits, risks and policies. *Renewable and Sustainable Energy Reviews*, 120:109663, 2020. ISSN 1364-0321. doi: https://doi.org/10.1016/j.rser.2019.109663. URL https://www.sciencedirect.com/science/article/pii/S1364032119308688.

[23] J. Synnott, L. Chen, C. Nugent, and G. Moore. The creation of simulated activity datasets using a graphical intelligent environment simulation tool. In *2014 36th Annual International Conference of the IEEE Engineering in Medicine and Biology Society*, pages 4143–4146, Aug. 2014. doi: 10.1109/EMBC.2014.6944536. URL https://ieeexplore.ieee.org/document/6944536. ISSN: 1558-4615.

[24] J. Synnott, C. Nugent, and P. Jeffers. Simulation of Smart Home Activity Datasets. *Sensors*, 15(6):14162–14179, June 2015. ISSN 1424-8220. doi: 10.3390/s150614162. URL https://www.mdpi.com/1424-8220/15/6/14162.

[25] United Nations, Department of Economic and Social Affairs, Population Division. World population ageing 2020: Highlights – living arrangements of older persons, 2020. URL https://www.un.org/development/desa/pd/content/world-population-ageing-2020-highlights.

[26] World Health Organization. Ageing and health, 2025. URL https://www.who.int/news-room/fact-sheets/detail/ageing-and-health. WHO Fact Sheet.

[27] World Health Organization. Ageing: Global population, 2025. URL https://www.who.int/news-room/questions-and-answers/item/population-ageing.

[28] Y. Zhan and H. Haddadi. MoSen: Activity Modelling in Multiple-Occupancy Smart Homes, Jan. 2021. URL http://arxiv.org/abs/2101.00235. arXiv:2101.00235 [cs.HC].

A | Appendix A

A.1. NeeMAS Questionnaire

Buongiorno, le chiediamo gentilmente 5 minuti circa per compilare questo questionario anonimo parte del progetto NeeMAS (Need-based Multi-Agent Simulator) del Politecnico di Milano e contribuire allo sviluppo di un simulatore di comportamento basato sui bisogni delle persone over 65.

Il presente questionario è rivolto a persone di età superiore ai 65 anni, ma può essere compilato anche da un familiare, amico o conoscente. Il modulo è composto da 20 semplici domande a scelta multipla divise in 4 sezioni, corrispondenti ai seguenti bisogni: mangiare, bere, socializzare e dormire.

Parte 1: Mangiare

1. Quali di questi pasti consuma abitualmente? (una o più scelte)

   ☐ Colazione ☐ Spuntino mattutino ☐ Pranzo ☐ Merenda ☐ Cena

2. A che ora fa colazione solitamente? (una scelta)

   ○ Mai ○ 6:00 ○ 6:30 ○ 7:00 ○ 7:30 ○ 8:00 ○ 8:30 ○ 9:00 ○ 9:30 ○ 10:00

3. A che ora pranza solitamente? (una scelta)

   ○ Mai ○ 11:00 ○ 11:30 ○ 12:00 ○ 12:30 ○ 13:00 ○ 13:30 ○ 14:00 ○ 14:30

4. A che ora cena solitamente? (una scelta)

   ○ Mai ○ 17:30 ○ 18:00 ○ 18:30 ○ 19:00 ○ 19:30 ○ 20:00 ○ 20:30

5. Qual è il luogo da lei preferito per i suoi pasti? (una scelta)

   ○ Cucina ○ Camera ○ Salotto

6. Qual è la durata media di un pasto? (una scelta)

   ○ 15–30 min ○ 30–45 min ○ 45–60 min ○ Più di 60 min

Parte 2: Bere

7. Durante la giornata, quante volte beve al di fuori dei pasti? (una scelta)

   ○ Mai ○ Poche (1 o 2) ○ Alcune (da 3 a 5) ○ Molte (più di 5)

8. In quali momenti della giornata beve fuori pasto? (una o più scelte)

   ☐ Mattina ☐ Pomeriggio ☐ Sera ☐ Non ci sono differenze

Parte 3: Socializzare

9. Con quante persone mantiene rapporti sociali quotidianamente? (una scelta)

   ○ Nessuna ○ Poche (1 o 2) ○ Alcune (da 3 a 5) ○ Molte (più di 5)

10. Con chi socializza principalmente? (una o più scelte)

    ☐ Familiari ☐ Amici ☐ Vicini di casa ☐ Altro: _____

11. Quali sono i luoghi dove socializza maggiormente? (una o più scelte)

    ☐ Al chiuso (bar, centro attività, . . .)
    ☐ All'aperto (parco, piazza, . . .)
    ☐ A casa

12. Che orari preferisce per socializzare? (una o più scelte)

    ☐ Mattina ☐ Pomeriggio ☐ Sera

13. In una scala da 1 a 10, quanto considera importante la socialità? (una scelta)

    ○ 1 ○ 2 ○ 3 ○ 4 ○ 5 ○ 6 ○ 7 ○ 8 ○ 9 ○ 10

Parte 4: Dormire

14. Quante ore dorme normalmente di notte? (una scelta)

    ○ 4 o meno ore ○ 5–6 ○ 6–7 ○ 7–8 ○ 9 o più ore

15. A che ora si addormenta solitamente? (una scelta)

    ○ 19:00 ○ 20:00 ○ 21:00 ○ 22:00 ○ 23:00 ○ 00:00 ○ 01:00

16. Quante volte si alza durante la notte? (una scelta)

    ○ Mai ○ 1–2 volte ○ 2–3 volte ○ 3–4 volte ○ Più di 4 volte

17. Quante volte si sveglia, anche senza alzarsi, durante la notte? (una scelta)

    ○ Mai ○ 1–2 volte ○ 2–3 volte ○ 3–4 volte ○ Più di 4 volte

18. Quando si sveglia la notte, quanti minuti ci vogliono per riaddormentarsi? (una scelta)

    ○ Meno di 15 min ○ 15–30 min ○ 30–45 min ○ 45–60 min ○ Più di 60 min

19. Con che frequenza fa un riposino pomeridiano? (una scelta)

    ○ Mai ○ Raramente (1–2 giorni a settimana) ○ A volte (3–4 giorni a settimana) ○ Sempre

20. Quanti minuti dura mediamente il riposino pomeridiano? (una scelta)

    ○ Non lo faccio ○ 15–30 min ○ 30–45 min ○ 45–60 min ○ Più di 60 min

List of Figures

List of Tables

Acknowledgements

Ringrazio i Professori Sara Comai e Fabio Salice per la loro disponibilità, il costante supporto e la guida durante la stesura di questa tesi.

Un ringraziamento particolare va alla mia famiglia che mi ha sempre supportato e aiutato durante questi anni impegnativi. Ringrazio inoltre mia nonna Lilli e mia zia Grazia che hanno contribuito direttamente a questa tesi con il loro prezioso aiuto, distribuendo decine di questionari ai loro amici.

Ringrazio i miei amici e le persone conosciute nel mio percorso universitario per le esperienze e gli indimenticabili momenti passati insieme.

Ringrazio infine tutte le persone che hanno dedicato il loro tempo per compilare il questionario, il cui contributo è stato fondamentale per la realizzazione di questa tesi.