

FMH606 Master's Thesis 2017
Industrial IT and Automation

Discrete Events Modelling of a Person Behaviour at Home

Badreddine Cherradi

Faculty of Technology, Natural Sciences and Maritime Sciences
Campus Porsgrunn

Course: FMH606 Master's Thesis 2017

Title: *Discrete Events Modelling of a Person Behaviour at Home*

Pages: 76

Keywords: *Behaviour Modelling, Elderly, Routine, Agents, Fuzzy Logic*

Student: *Badreddine Cherradi*

Supervisor: *Carlos F. Pfeiffer*

External partner:

Availability: *Open*

Approved for archiving: _____

(Carlos F. Pfeiffer)

Summary:

Modeling human behaviour remains a challenge for both computing sciences and the humanities. In this context, we worked on a routine simulation program and a human behaviour analysis program. The simulation is based on the definition of an activity as the combination of several states (Position, Room, Level of activity, etc.). The addition of the time dimension makes it possible to decide a behaviour. A simulator allows us to fill a database corresponding to one that would be established by means of cameras and sensors. The analysis program then uses an EventHandler to allow a system of Agents to analyze the behaviours that are introduced into the database by the simulator. Once the data are collected by these Agents, fuzzy logic and exponential distribution are used to establish a judgment on the normality of a behaviour. Several routines were generated using the simulator to populate the database in order to test the analysis program. The alarms output were those expected based on the behaviours input. In the long term, our study represents a step towards providing a powerful tool for the assistance of elderly people living alone.

Preface

This master thesis has been written to fulfill the graduation requirements of the Master degree in Industrial Engineering Sciences at the Institut Supérieur Industriel De Bruxelles (ISIB). It was carried out in the ERASMUS program at the University College of South-east Norway (HSN) from February to June 2017.

This thesis follows the path of Carlos F. Pfeiffer and Nils-Olav Skeie's research project on modelling human behaviour for smart houses. The theoretical aspect that mingles with computer codes is essential for me. Through this work, I have tried to maintain a certain balance between "what is ideal" and "what is realistic".

I hope you enjoy your reading.

Porsgrunn, 14th May 2017

Badreddine Cherradi

Contents

Preface	5
Contents	8
List of Figures	10
List of Tables	11
1 Introduction	15
1.1 Project Description	16
1.2 Document structure	19
2 Theoretical background	21
2.1 Related Work	21
2.1.1 Hidden Markov Models	21
2.1.2 Fuzzy Logic	22
2.1.3 Frequent Pattern Mining	23
2.1.4 Recurrent Neural Network	23
2.2 Stochastic methods for discrete events modelling	24
2.2.1 Hidden Markov Models	24
2.2.2 Agent	27
2.2.3 Fuzzy Logic	30
3 Methodology	33
3.1 Introduction	33
3.2 Development Process	33
3.3 States	34
3.4 Simulating a morning routine	35
3.4.1 Activity and Behavior	35
3.4.2 Database	36
3.4.3 Timer	36
3.4.4 Codification	39
3.4.5 Variation in the routine	39
3.4.6 Cache database	41
3.5 Analysis	42
3.5.1 Introduction	42
3.5.2 User Interface	42

Contents

3.5.3	EventHandler	43
3.5.4	Agents	45
3.5.5	Classification	51
4	Results	53
4.1	Testing module	53
4.2	Situation 1 : Normal routine	55
4.3	Situation 2 : Irrelevant behaviour time	56
4.4	Situation 3 : Abnormal behaviour time	57
5	Future work	59
5.1	User-friendly routine adding system	59
5.2	Sequences analysis	60
5.3	Frequency of behaviour occurrence	62
6	Conclusion	63
	References	65
	Appendices	69

List of Figures

1.1	Monitoring system architecture	17
1.2	Example of a situation	17
1.3	Example of a database	18
2.1	Parameters of a HMM	25
2.2	Graphical model of HMM	25
2.3	Trellis diagram for the Viterbi algorithm	26
2.4	Schematic diagram of a simple reactive Agent	27
2.5	Schematic diagram of a BDI Agent.	28
2.6	Agent interaction with the environment	30
2.7	Fuzzy Inference	32
3.1	Comparison of Waterfall and Scrum	33
3.2	States' values	34
3.3	Simple routine	35
3.4	C# code for the routine	35
3.5	Table Activity	36
3.6	UI of the Simulator	36
3.7	Behaviour switch	37
3.8	Day switch	37
3.9	Flowchart of the routine code	38
3.10	Codification of the state <i>Position</i>	39
3.11	Behaviour transition	39
3.12	Modified routine	40
3.13	Database Cache	41
3.14	UI of the analysis program	42
3.15	C# code of the lists implementation	43
3.16	C# of the delegates implementation	44
3.17	Plot of the exponential survival function	48
3.18	C# code to use the survival function	49
3.19	Plot of the membership functions	52
3.20	Fuzzy rules	52
4.1	State indicator of the simulator on the analysis program	53
4.2	C# testing code	54

List of Figures

4.3	UI Testing module	54
4.4	Result of the first test	55
4.5	Rules' Fire Strenght (Agent 1)	55
4.6	Rules' Fire Strenght (Agent 2)	55
4.7	Result of the second test	56
4.8	C# code of the routine variation	57
4.9	Result of the third test	57
5.1	Analysis program architecture	59
5.2	CodeBook	60
5.3	C# code for the HMM implementation	61

List of Tables

- 2.1 Zadeh MIN/MAX set of operators 31
- 3.1 Example 1 of collected values 49
- 3.2 Example 2 of collected values 49
- 4.1 Input and output testing values 54
- 5.1 Likelihood of generated samples 61

Nomenclature

Symbol	Explanation
ADLs	Activities of Daily Living
SVMs	Support vector machines
RNN	Recurrent neural network
BDI	Beliefs, Desire and Intention
MAS	Multi-Agent System
RNG	Random Number Generator
CSP	Cryptographic service provider
DBMS	Database management system
GPL	General Public License
CRUD	Create, read, update, and delete
UI	User interface
GUI	Graphical user interface
AOP	Agent-oriented programming
SOA	Service-oriented architecture
ACL	Agent Communication Language
JADE	Java Agent Development Framework
SOAP	Simple Object Access Protocol
REST	Representational State Transfer
WSDL	Web Services Description Language
FUSION@	Flexible User and Services Oriented multi-agent Architecture
CSV	Comma-separated values

1 Introduction

The study of human behaviour has always been a major concern for scientists and great thinkers since antiquity. The History of Psychoanalysis, examining what lies beneath the surface of human behaviour, begins with the work of Sigmund Freud in the early 19th century. Behaviorism founded by John Broadus Watson in 1913 scientifically study the behaviour of living organisms and their relations with the environment. The fundamental proposition of behaviourism is that any behaviour is the result of learning [1].

At the same time that behaviourists focus their research on learning behaviours, ethologists, scientists who study animals in their natural environments, start to take an interest in *instinct*. They consider that behaviour is instinctive when it is innate [2]. Behaviorists objected this idea and presumed instinct is only prenatal learning. For example, chicks recognizing from their birth the call of their mother is due to the fact that, already in the egg, chicks become familiar with the voice of their mother. A wide array of human behaviour are based on an instinctual nature that are involuntary such as breathing or primitive reflexes in newborn [3]. A modern definition of instinct might be: a largely inheritable and unalterable tendency of an organism to make a complex and specific response to environmental stimuli without involving reason [4].

Psychoanalytic theories have introduced a dynamic conception of mental life. They have made it possible to consider the symptom of an illness as a function of past history considering the successions of different situations and their possible reappearance. The cognitivist current, coming from work on logic and mathematics and the development of computer sciences, aims to develop machines that have the ability to virtually reason like a human brain. It considers the brain as similar as a computer working by processing information and communicating with the environment by manipulating different symbols.

A human collects and stores information from the environment. He analyzes this information and then takes it into account in his decision-making process [5]. Another important part of the consistency between a computer and a human brain is the concept of memory. In 1984, Douglas L. Hintzman developed an episodic memory model called MINERVA 2. This project was an attempt to simulate the process by which memory can generate abstract representations from unmatched and contextualized experiences. The model makes it possible to apply frequency judgment which mean the capability of judging if an item is “old” or “new” based on the frequency of it’s appearance in the memory [6].

1 Introduction

With the recent technological breakthroughs in helping people, there is an increasing amount of research on human behaviour in the field of artificial intelligence. The emergence of artificial intelligence seeking to reconstitute human intelligence within digital systems, enabled the developing of simulation tools that allows the traceability of phenomena, reiteration of actions and key moments. Moreover, the possibility of studying non-existent cases allows us to make the unpredictable aspect of the human behaviour appear within a digital model [7].

Human behaviour is characterized by a large number of internal and external interactions with the environment. Based on these interactions, humans do not always make rational decisions [8]. However, human behaviour can be described by human routines which can be simply defined as a sequence of actions regularly followed [9]. Routines can be seen as different actions performed in particular situations that caused those actions. Capturing routines from a human everyday life would allow us to partially model a system representing human behaviour[10].

In order to build a model of human's routine, a necessary first step is the gathering of all the information needed. Informations such as images, movement data, temperature and humidity are acquired through cameras and sensors. These informations can be analyzed to predict the position, state or activity of a person. The key challenge here is the extracting of a routine pattern from a big dataset of informations. An important aspect in this study is the routine variations that is part of the routine behaviour. Several questions have to be answered such as : *When does the routine begin? When does it stop? Is there a variation in the routine we are analyzing?*

Commercial system as Care Innovations[11] propose a system called QuietCare's that uses multiple sensors located in different places such as doors, refrigerators, etc. The datas collected from these sensors are used in the their QuietCare's algorithm that learns activity patterns and provides alerts to caregivers when an abnormal situation occurs.

1.1 Project Description

Aging people living alone are more vulnerable to accidents and may not be able to ask for assistance. We consider that to be able to live independently at home, a person must complete Activities of Daily Living (ADLs). Monitoring the completion of the ADLs would allow us, with the help of an adequate human behaviour model, to be notified of a change of behaviour or an abnormal situation[12]. The application we will be focusing on is part of a research project led by Carlos F. Pfeiffer, Veralia Gabriela Sánchez and Nils-Olav Skeie[13]. The system they propose is an automatic non-invasive monitoring system of the behaviour of a person living alone. Currently, a prototype system is under construction at the University College of Southeast Norway. Figure 1.1 shows the monitoring system architecture of the main project.

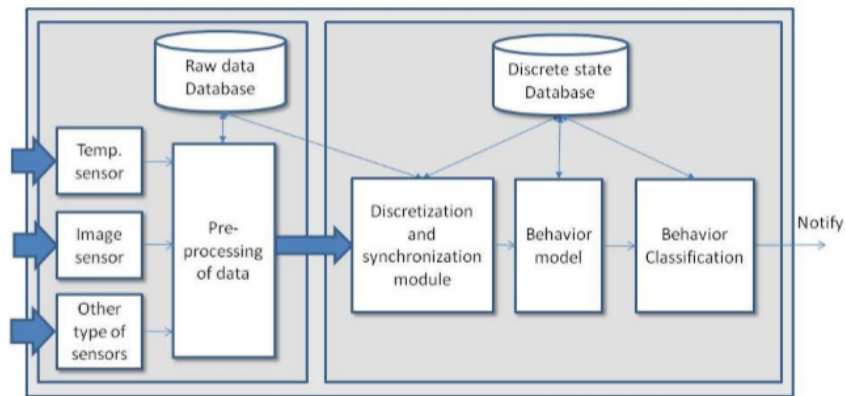


Figure 1.1: Monitoring system architecture

A three room apartment lab is equipped with sensors and cameras that acquire position and calculate activity levels [13]. Figure 1.2 shows an example of a commonplace situation in this three room apartment: A person enters his home at noon and directly goes to the bathroom. To reach the bathroom, he has to pass through the main hall and the bedroom.

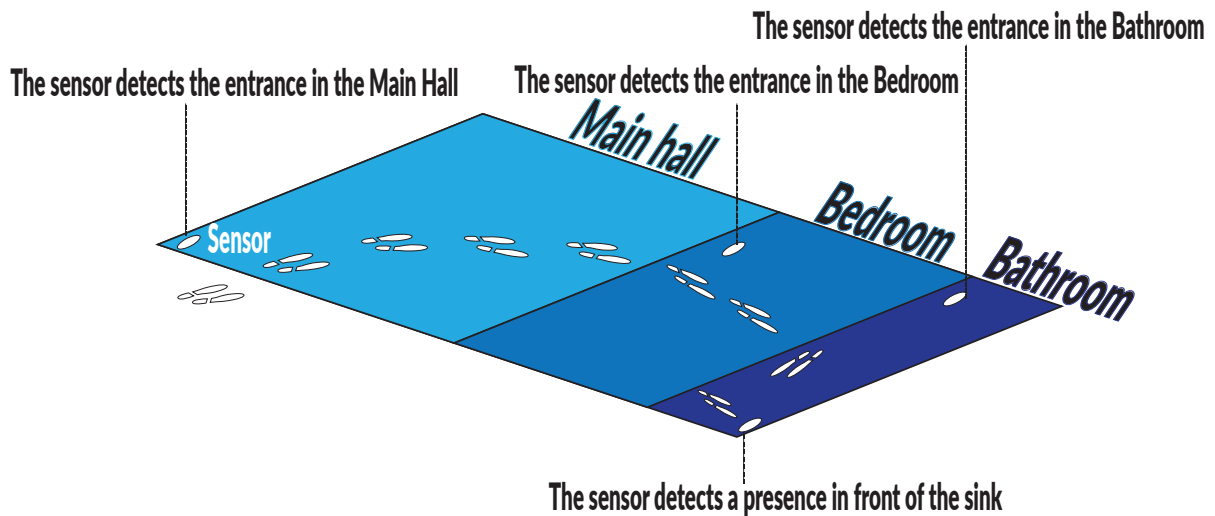


Figure 1.2: Example of a situation

First of all, we must define the *Person Status* which is characterized by a time stamp, the room (Hall, Bedroom, Bathroom and Unknown) where the person is and his/her position (Standing, Lying, Sitting and Unknown). More states are defined such as the *Room Status*, the *Room section state* and the *External Status*. All these states at a given time define the state of the system. Second, we define an *Activity* as a set of system states. The combination of different states are mapped to an activity. Furthermore, we express

1 Introduction

a *Simple behaviour* as a single activity that lasted a certain period of time. Finally, a *Complex behaviour* is used to describe a sequence of simple behaviours.

We will implement a system simulator that populates a database in order to test different behaviour modelling techniques. The database will be populated by data that a sensor could provide. Those data will be then mapped into discrete state variables. Let us consider that we only populate the database with the room where the person is located, the position and the time. Then a possible database would be created as shown in Figure 1.3.

	ROOM	POSITION	TIME	
Activity	Unknown	Unknown	2017-02-11 12 : 26 : 45	Simple behavior
	Hall	Standing	2017-02-11 12 : 27 : 00	
	Hall	Standing	2017-02-11 12 : 27 : 15	
	Bedroom	Standing	2017-02-11 12 : 27 : 30	Complex behavior
	Bathroom	Standing	2017-02-11 12 : 27 : 45	
	Bathroom	Standing	2017-02-11 12 : 28 : 00	
	Bathroom	Standing	2017-02-11 12 : 28 : 15	

Figure 1.3: Example of a database

In order to build a model for behaviour analysis and classification, a preprocessing step is needed. We will then manipulate these data using different statistical tools and stochastic methods and then proceed to the behaviour classifying step and create an alert system.

1.2 Document structure

This thesis is organized as follows:

- The Theoretical background chapter presents a literature review in order to evaluate previous research on the human behaviour in a smart house topic. Then, some theory about stochastic method for discrete events modelling is given.
- The Methodology chapter outlines methods we used to develop our simulator and analysis program.
- The Result chapter presents findings of different situations by testing our analysis program.
- The Future work chapter describes possible improvements to the continuity of the project.
- The Conclusion chapter sums up the development of this work and the achievement made.

2 Theoretical background

2.1 Related Work

Researchers have reviewed a wide range of methods to model human behaviour. Their approaches differ by the type of sensors used and which kind of data is collected. They choose which pattern is recognized and propose a method to train a model that will use the recognized pattern in order to predict the behaviour of an individual. We will describe in this section four papers that we selected for their relevance and methodological differences dealing with ADLs recognition in a smart house.

2.1.1 Hidden Markov Models

Detecting Human Behavior Models From Multimodal Observation in a Smart Home [14]

The experiments described in this paper take place in a room equipped with microphone arrays and video cameras. A 3-D real-time robust tracking system is able to detect and track objects in these video images captured from the cameras. Different postures can be detected using support vector machines (SVMs). To optimize the recognition of the posture, the number of classes has been reduced by the author. The basic postures detected are “standing”, “lying down” and “sitting”. The main goal of the SVM is to find the hyperplane that separate the different classes. Classification is simply done by determining on which side of the hyperplane is the vector we are testing. The author uses a “one-against-one” classification meaning that the testing data are compared with two classes and the score of the winning one is incremented. We associate the testing data to the class with the highest score. In addition to postures, the author uses properties as speed and interaction distance to object in the room to determine additional classes. These classes are considered as “individual roles”. From the individual roles, ambient sound and speech different situation are considered: siesta, individual work, introduction, aperitif, presentation and game. Hidden Markov Models are used to learn these different situations. The Baum–Welch algorithm is used to find the unknown parameters of the Hidden Markov Models. The Viterbi algorithm is then used to find the most likely state sequences and their probabilities for a given observation sequence. Several recordings were done involving up to two persons. The offline situation (only involving classification from learned situations) have led to good results. Finally, the author conducted series of

detection of situations within the long recordings. The author highlights the fact that the transitions between situations need to be correctly detected.

2.1.2 Fuzzy Logic

Human Activities of Daily Living Recognition Using Fuzzy Logic For Elderly Home Monitoring [15]

Fuzzy Logic is the decision module of the activities of daily living's recognition system described in this paper. Fuzzy Logic is used to map knowledge onto fuzzy relationships to avoid manipulating complex probabilistic tools. The author motivates the use of fuzzy logic using two main reasons. First, the imprecision and imperfection of the data collected by the different sensors and second, the well-known utility in pattern recognition. The values of the linguistic variables used in this paper for the application of fuzzy logic are adjectives and adverbs of language. Numerical scale of length are replaced by fuzzy labels as "very small", "small", "medium", "large" and "extra large". Fuzzy logic is very useful to describe syntactic data (well-defined format), numerical and contextual data or conceptual data. Fuzzy c-mean algorithm and fuzzy ISODATA[16] algorithm can be used to do clustering. The author mentions the fact that the design of a "discriminator" would be done to produce a fuzzy partition in order to describe the data. Fuzzy logic reflects human reasoning and human language. The first step of the fuzzy logic is called "fuzzification". A numerical input is translated into a fuzzy variable. Using a triangular membership function this fuzzy variable gets a membership degree to a fuzzy set. The inference system build up fuzzy logic rules. A fuzzy rule has the following structure: "IF linguistic variable IS input fuzzy variable (AND/OR) ... then linguistic variable is output fuzzy variable." The last step of the fuzzy logic system is the defuzzification which is the translation of the output fuzzy variable generated by the fuzzy rule into a real value. The author used sound to define fuzzy sets as for example "object sound" composed by "chair", "table" and "step foot". The input activity is described by four fuzzy sets that are "immobile", "rest", "normal" and "agitation". The output activity is described by sets as "Sleeping", "Getting up", etc. The next step is the use of a domain expert knowledge of the activities with the aim of obtaining fuzzy rules. The author and his team developed a software that allows the writing of fuzzy rules and the configuration of the defuzzification method. At the end, several tests were realized and rules were added when a detection was missed.

2.1.3 Frequent Pattern Mining

A Frequent Pattern Mining Approach for ADLs Recognition in Smart Environments [17]

In this paper, an approach based on frequent pattern mining to recognize ADLs is proposed. A pattern mining has as purpose the explanation of how individuals of event sequences behave. In this case, events corresponds to the states of the sensors in the room. A sequence is an ordered list of events that can be written as e_1, e_2, \dots, e_m (where m is the number of events). An event correspond to a sensor state and is associated with a time-stamp. Frequent pattern mining allows us to discover patterns called episodes. An episode is composed of events with almost similar timestamps. The frequency of an episode is defined by how often this episode occurs in an event sequence. A first parsing of all the sequences in the dataset D is done by using the Apriori algorithm. The extracted frequent episodes are then mapped with activity models. Considering that human perform activities in a hierarchically structure, the author decomposed the activities in task, subtasks and elementary tasks that cannot be decomposed. An activity can be described by different episodes composed with the same events but in different order. The author compared his approach with the Hidden Markov Model. The results obtained with the Frequent Pattern Mining Approach were better than those obtained using the Hidden Markov Model. The author pointed to the fact that the activities associated to a great number of sensor are recognized with higher accuracy.

2.1.4 Recurrent Neural Network

Recurrent Neural Network for Human Activity Recognition in Smart Home [18]

In this article, recurrent neural network (RNN) is used for human activities recognition. The research project CASAS smart home project is led at the Washington State University. The first step consisted in providing training data to the algorithm. The sensors collected 10 activities as for example “Bed to toilet”, “Breakfast” or “Dinner”. The data collected are completed with the date, time, sensor ID, sensor value and a label. The recurrent neural network used is made of three layers: an input layer, a hidden layer and an output layer. The recurrent neural network algorithm aims to minimize the error function by modifying weight coefficients. The neurons used in the input layer, hidden layer and output layer were 6, 10 and 10 respectively. Finally, the results obtained by the RNN looked better compared to those obtained by the HMM and the naïve Bayes classifier.

2.2 Stochastic methods for discrete events modelling

In this section we will describe different stochastic methods to model activities in order to be used to model human behaviour. Stochastic process is defined as a collection of random variables [19]. Random variables refers to a number depending on the result of a random experiment. One of the simplest stochastic processes is the Bernoulli process. Bernoulli process is described as a sequence of independent and identically distributed random variables. Each random variable takes the value 1 with probability p . The value 0 is then taken with probability $1 - p$ [20]. The reason we consider a stochastic method is due to the fact that in our case a cause do not always induce the same result. Let us take for example an elderly person who has as a morning routine that consists on getting up at a time $T_1 + \Delta t_1$ and take his breakfast at a time $T_2 + \Delta t_2$. Each day we record those times where Δt_1 and Δt_2 change. These small fluctuations make the results not certain. We decided to only consider discrete events in our application. This decision is fully consistent with the information provided by majority of the sensors. This is part of the preprocessing step in order to transform the sensor data to discrete values.

2.2.1 Hidden Markov Models

A hidden Markov model [21] considers a system which has as parameters N distinct states $S = \{S_1, S_2, \dots, S_N\}$. Those states generate a set of M external observations $O = \{O_1, O_2, \dots, O_M\}$. As time goes by, the system undergoes a change of state. This change is made according to the set of probabilities of each different state. The next state of the system is determined by the set of transition probabilities together with the current state. Let us consider that X_t is the state of the system at a given time t . We can define the transition probabilities $A = \{a_{ij}\}$ as

$$a_{ij} = P(X_{t+1} = S_j | X_t = S_i) \quad (2.1)$$

An important comment to bear in mind is that the transition probabilities remain constant over time. These probabilities do not change according to the parameter t . When the transition is completed, an output is produced based on a set of output probabilities. We can define the output probabilities $B = \{b_{ij}\}$ as

$$b_j(k) = P(O_k | X_t = S_j) k \in [1, M] \quad (2.2)$$

Figure 2.1 illustrates the dependencies between the probabilistic parameters of a HMM. We consider the random variables X_t which is the hidden state and the random variable $Y(t)$ is the observation at a time t . We will consider in our case that $X(t) \in \{X_1, X_2\}$ and $Y(t) \in \{y_1, y_2, y_3\}$. The variables a_{ij} are the state transition probabilities and b_{ij} the output probabilities [22].

2.2 Stochastic methods for discrete events modelling

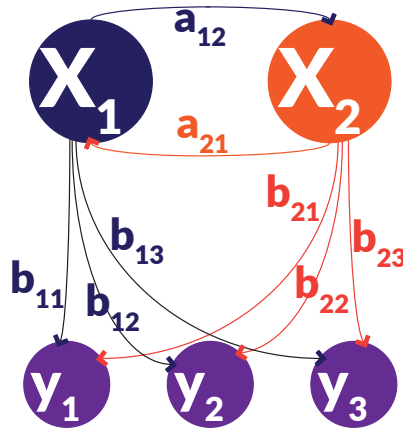


Figure 2.1: Parameters of a HMM

Hidden Markov Models responds generally to 3 problems. The first one is the evaluation problem which corresponds to the determination of the probability that a particular sequence of symbols is produced by a particular model. For this purpose, we can use the forward algorithm or the backwards algorithm. The forward algorithm predicts the state in the future, given current observations. The backwards algorithm updates predictions about states in the past, given more recent observations [23]. Let us take as example a Hidden Markov Model composed of two possible states S_1 and S_2 and three possible observations y_1 , y_2 and y_3 . We can use the forward algorithm to predict what is the posterior probability of X_3 given the observations up to time $t = 3$.

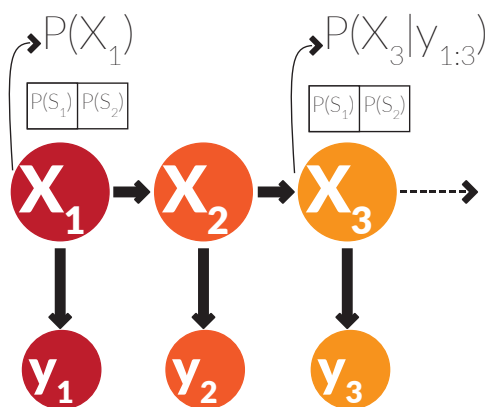


Figure 2.2: Graphical model of HMM

2 Theoretical background

The second problem is the decoding which corresponds to the determination of the most likely sequence of states that produced a given sequence of observations. The Viterbi algorithm is used for this problem. The Viterbi[24] algorithm can be seen as a path-finding problem where the probability of a state is the probability of the most likely path to that state. Figure 2.1 shows an example of a trellis diagram for the Viterbi algorithm is shown below.

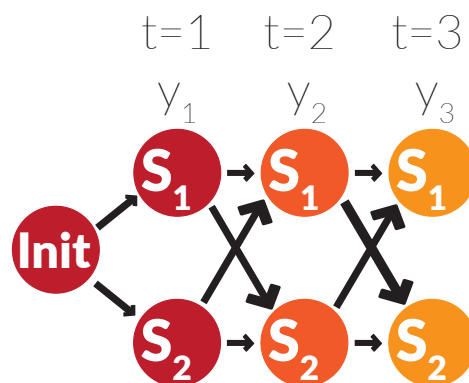


Figure 2.3: Trellis diagram for the Viterbi algorithm

The last problem is the training problem which is finding the model that best fits the data, given a HMM structure and a set of sequences of observations and states. The Baum-Welch[25] algorithm is an iterative algorithm which estimate the parameters (the start probabilities, transition probabilities and emission probabilities) of the model we want to find. One iteration is done by calculating forward probabilities and the backward probabilities with the forward algorithm and the backward algorithm. We then calculate the contributions of the current sequence to the transitions of the model and the contributions of the current sequence to the emission probabilities of the model. Finally we calculate the new model parameters.

2.2.2 Agent

We define an Agent as a physical or virtual entity that is able to perceive in a limited way its environment and act on it. It must be able to perform actions on the environment and vice versa. An Agent possesses his own resources, skills, services and can potentially communicate directly with other Agents. It works without direct intervention. Advanced Agent can be proactive and take initiatives collecting information to improve its future activity. An Agent is characterized by a set of individual objectives or a survival function. Taking into account the resources, competences at its disposal and according to its perception and the inputs it receives, an Agent seeks to optimize his survival function[26].

Reactive Agents

Reactive Agents are defined only from stimulus-response rules and allow to model very precise behaviours. They do not have internal states, historical memory or representation of their environment and other Agents. They are unable to predict or anticipate what will happen. A reactive Agent can be seen as a system accomplishing a given task. This system establishes a relationship between a sensory entity and an output action. This if-then-else logic is common in computer sciences. First, the Agent find a rule whose conditions match the inputs. Then it accomplishes the action associated with that rule[27].

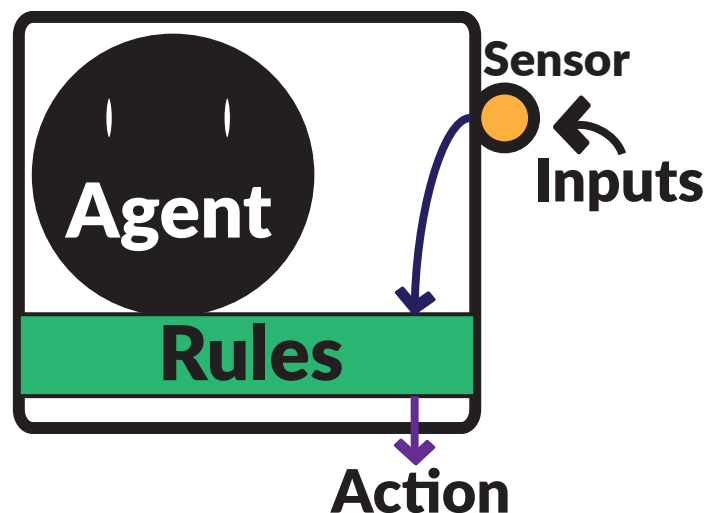


Figure 2.4: Schematic diagram of a simple reactive Agent

Cognitive Agents

Cognitive Agents have internal states and memory to represent the evolution of their environment. These Agents are generally described as BDI (Beliefs, Desire and Intention) Agent.

- Beliefs represent what the Agent knows about his environment.
- Desires represent the objectives the Agent may want to achieve.
- Intentions represent objectives for which the Agent is engaged.

In order to determine which action to perform, the Agent must update its internal states and knowledge based on the information it receives from its environment. The Agent operates in an environment populated by other Agents. It may be in a situation of social dependence on one or other Agents. By responding to events coming from the environment, the Agent updates his state of knowledge and adjust, in a filter function, its intentions according to its current beliefs and desires[28]. Figure 2.5 shows a description of this process.

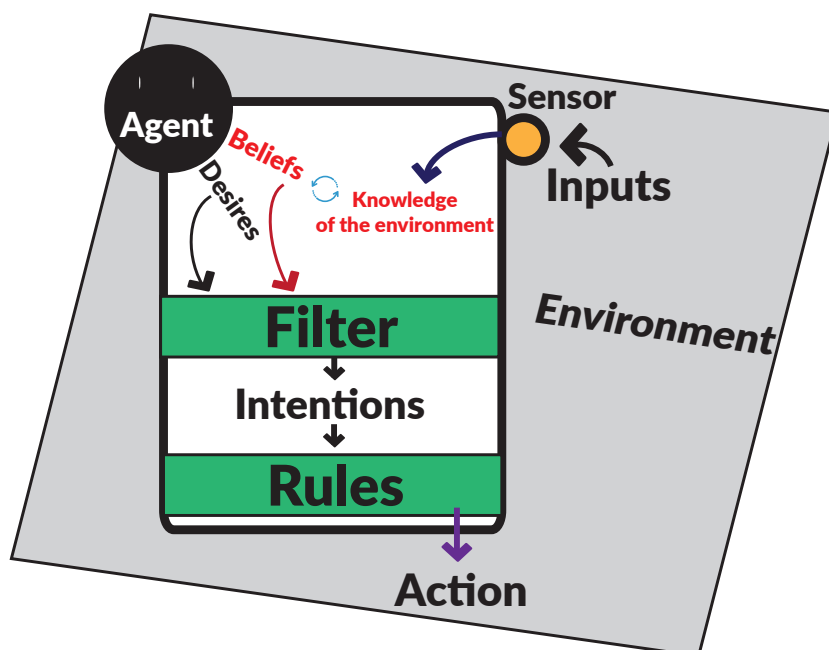


Figure 2.5: Schematic diagram of a BDI Agent.

Multi-Agent System

A Multi-Agent System (MAS) is a system made of a set of objects located in an environment. These passive objects can be perceived, created, destroyed and modified by particular objects representing the active entities of the system (Agents). A set of relations and operations allow Agents to perceive, produce and manipulate passive objects. An Agent is unable to store all information from the environment. However, it can interact with other Agents in its neighborhood to explore the environment. Each Agent has local knowledge, but it remains accessible to inspection by other Agents[26].

The notion of interaction in MAS is very important. It is defined as any form of action within the MAS that has the effect of changing the behaviour of another Agent. An interaction can occur between Agents or between Agents and the environment. A reactive Agent within a MAS will pursue an individual goal. It will not use his resources to interact with others Agents. Conversely, a Cognitive Agent will participate in the satisfaction of the overall goal of the system while pursuing an individual goal. It will spend part of its time cooperating with other Agents [29].

MAS monitoring simple behaviours

In order to implement a MAS in our application, we must first define the system that will be monitored by our Agents. Within our application, we have a series of activities that occur at specific times. We defined this as a simple behaviour. Over time, these behaviours repeat. We are then in a position to ask ourselves several questions :

- Of which activity is the behaviour composed?
- How long does the behaviour usually take to finish ?
- Knowing the history of this behaviour, what is the probability that it is still occurring?
- Which behaviours should precede and follow it?
- Knowing the history of this behaviour, does it often change in duration?
- Does the previous behaviour tend to change in duration that makes this behaviour shifts in time?

The main objective of our Agents is to retrieve as much information from the environment to answer these questions and alert us if there is an abnormal situation. For simple questions, a simple reactive Agent can collect information from the environment and answer. For more complex questions that involve the history of several behaviours at once, the collaboration of several Cognitive Agents is required. Figure 2.6 shows an example of the Agents interaction.

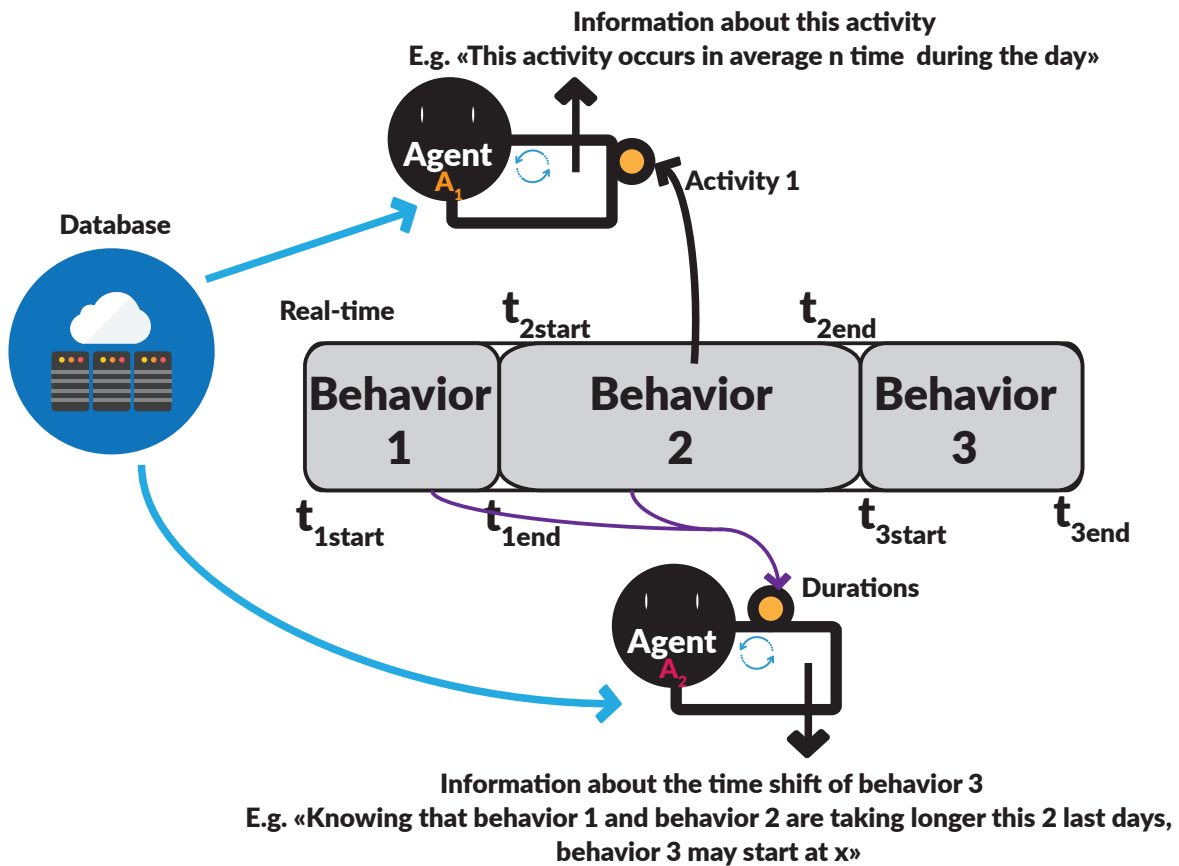


Figure 2.6: Agent interaction with the environment

2.2.3 Fuzzy Logic

Modern fuzzy logic is an extension of Boolean logic. It was developed by Lotfi Zadeh[30] in the mid-1960s. Fuzzy logic is aimed at resolving problems in which imprecise data must be used. Thanks to the Fuzzy Logic, we can consider qualitative values rather than quantitative values. By defining Linguistic Variables instead of numerical variables, it is possible to use Fuzzy Rules simulating the process of human reasoning. However, even if words are less precise than numbers, they are more suitable in several cases[31]:

- The data available is perception-based;
- The problem has a tolerance for imprecision which can lead to a simple, robust and less costly;
- Words are more expressive than numbers in some problems.

Fuzzy set

First, it is important to define the Fuzzy set concept[32]. In a conventional subset, the belonging of a given value to the subset is defined as true (1) or false (0). In classical set theory, an element belongs to a subset or it does not. In the case of Fuzzy subsets, the value can be between 0 and 1. This represents the uncertainty that exists in reality. The fact that an element belongs to a set only on a certain extent represent the *membership degree* to this set. Concepts such as being old or young, whether cold or hot, are good examples of diffuse statements where fuzzy logic can be applied.

Linguistic Variables

Linguistic Variable can be compared to an algebraic variable. Linguistic Variable takes words as values. It possess a term set which correspond to the set of values the Linguistic Variable can take. The values in the term set are fuzzy variable. For instance, a Linguistic Variable with the label “Distance” can have the following term set :

$$T = \{Far, Distant, Nearby, Close\}$$

Fuzzy Rules

Fuzzy rules are used to generate an output. These rules are constructed by simply using an If-Then structure with the Linguistic Variables. A fuzzy rule in the context of a change of speed as a function of distance will, for example, have the following structure:

“IF DistanceIn IS Nearby THEN SpeedOut IS Slow”

FuzzySet Operations

To manipulate FuzzySets easily, the operators of classical set theory were redefined to adapt them to membership functions of fuzzy logic allowing values strictly between 0 and 1. Lets take for example μ_A and μ_B as membership functions for the FuzzySets A and B. We present in Table 2.1 the Zadeh MIN/MAX set of operators :

Union (OR): $\mu_{A \cup B}(x)$	Intersection (AND): $\mu_{A \cap B}(x)$	Complement (NOT): $\mu_{\bar{A}}(x)$
$\max\{\mu_A(x), \mu_B(x)\}$	$\min\{\mu_A(x), \mu_B(x)\}$	$1 - \mu_A(x)$

Table 2.1: Zadeh MIN/MAX set of operators

2 Theoretical background

Fuzzy inference systems (Mamdani)

Fuzzy inference methods can be classified as direct or indirect methods. Mamdani's method[33] is one of the most common used direct method. Let us consider, for example, the LV1 and LV2 Linguistic Variables as inputs and LV3 as outputs. We can create two fuzzy rules. The first one using a “OR” fuzzy operator while the second one uses a “AND” fuzzy operator. Figure 2.7 present a detail description of the inference process.

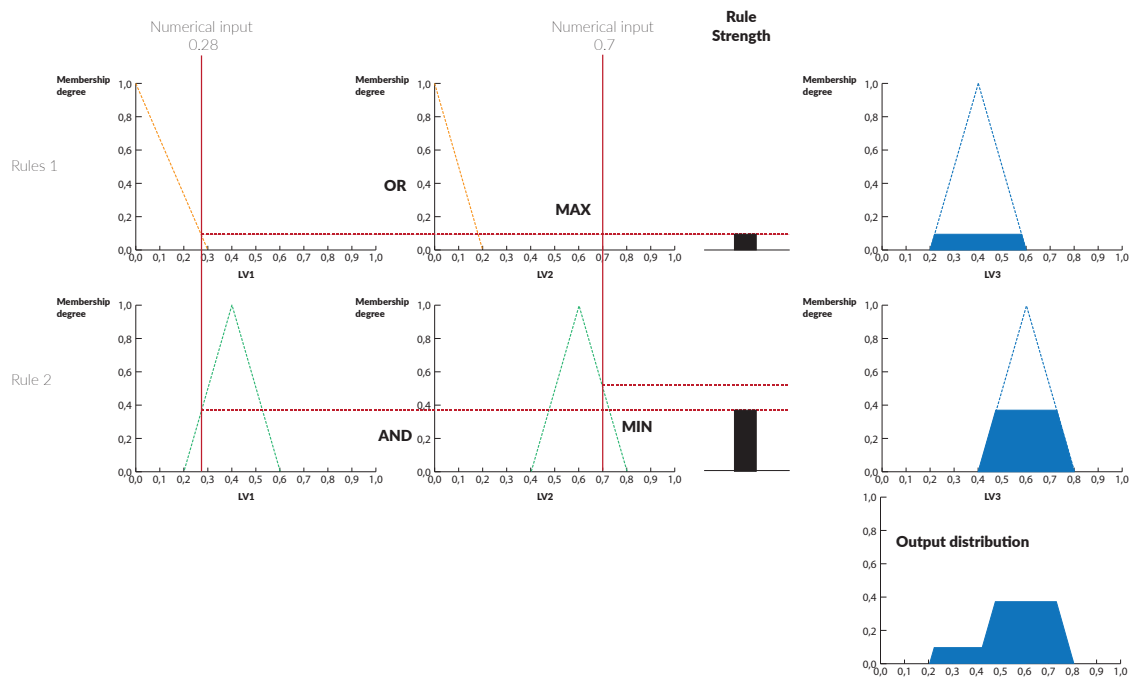


Figure 2.7: Fuzzy Inference

When fuzzifying the first part of the antecedent of Rule 1, we obtain the membership degree of LV1 while we obtain a zero value for the membership degree of LV2. We apply an OR operation, taking the maximum of both memberships degree. The same process is performed for Rule 2 except that the AND operator makes us take the minimum of both memberships degree. Then, we combine the outputs using the most maximum aggregation operator to obtain the output distribution.

Finally, the “defuzzification” step allow us to move from the output distribution to a single final decision. The two main methods of defuzzification are the mean method of maxima and the method of the center of mass.

3 Methodology

3.1 Introduction

This chapter is made of three different sections. The first one corresponds to the *Simulation of a routine*. In this part, we discuss the construction of the main classes of code, database and thread timer used. In the *Analysis* section, we explain the resources used to analyze the behaviours generated by the simulator. Additionally, this second section explains the main paradigms that inspired us to create our own approach. The last section outlines the *results* of three different situations. Then the analysis program will be evaluated in each tested situation. We used C# as the programming language to constitute the simulator and the analysis program.

3.2 Development Process

Scrum[34] is the main management framework we used for the development of our softwares. Scrum has an incremental approach that allows us to implement a software piece by piece. Each part of the developed software is functional and exploitable. Scrum's iterative process means that for each part we go through the same development phases. Figure 3.3 shows the difference between the traditional Waterfall approach and Scrum.

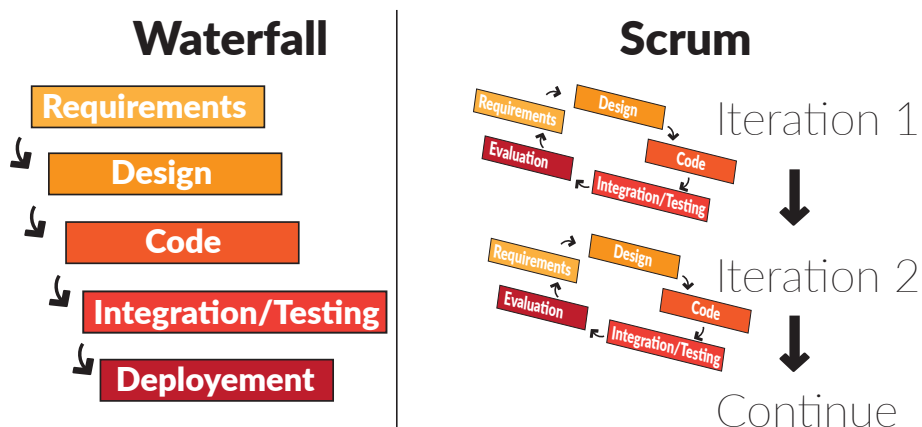


Figure 3.1: Comparison of Waterfall and Scrum

3 Methodology

Being alone, an adaptation of this process was made. To properly implement an effective Scrum methodology, we need a Scrum team. In the context of this work, we had the role of the development team and the Scrum Master (buffer between the team and the exterior). My supervisor Carlos F. Pfeiffer was the Product Owner ensuring that the developer delivers value to the project.

The software development progress is done through a series of "sprints", in our case iterations of 1 to 2 weeks at the most. During a sprint, daily scrum (small planning meeting to allows developers to make a coordination point) are made. In our case, we did not need to coordinate the work.

The requirements were clearly defined. The elements to be implemented are grouped under the name of "Product Backlog". The Product Owner describes the prioritized item to be implemented. In our case, this correspond to the main meetings with the supervisor to determine the main idea to implement.

3.3 States

By using the system definitions of Carlos F. Pfeiffer's[13] research project on human behaviour, we have established a first class named *Human*(see appendix 2 for the Human class code). Instances of this class have attributes named *position*, *room*, *loa* and *room-section* respectively of type *Position*, *Room*, *Loa* and *Roomsection* that are enumerations. An enumeration type is used to define a set of named states that may be assigned to a variable. The value that each enumerator can take is presented in Figure 3.2.

Position	Room	LOA	RoomSection
Undefined	Undefined	Undefined	Undefined
Lying	Bedroom	Low	Door
Sitting	Livingroom	Medium	Chair
Standing	Bathroom	High	Bed
None	Kitchen	None	Sink
	None		Toilet
			None

Figure 3.2: States' values

3.4 Simulating a morning routine

3.4.1 Activity and Behavior

An activity is built by combining different states. Take for example a person sleeping, a possible combination would be *Lying, Bedroom, Low* and *Bed*. By adding the time component to this activity, a behaviour is formed within precise duration. For our simulation, we use a cryptographic Random Number Generator (RNG) using the cryptographic service provider (CSP) to generate a random *Integer* between a minimum value and a maximum value. It should be noted that a person can wake up at different time. For example, the average sleep time of an elderly person is between 7 and 8 hours[35]. For timing constraint, one minute of real life activity will correspond to one second in the simulator which will allow us to quickly have enough data to analyze. The beginning of the routine is shown in Figure 3.3.

	Activity	Time [s]
1.	Lying, Bedroom, Low, Bed	[400 ; 430]
2.	Standing, Bedroom, Medium, Door	[2 ; 4]
3.	Standing, Livingroom, Medium, Door	[2 ; 4]

Figure 3.3: Simple routine

First, we implement a completely fixed routine with no change of behaviour. The only variable component are the durations. Figure 3.3 shows how a behaviour is implemented within the simulation in C#.

```

Instance of the class Human
case 2:
    H position = Human.Position.Standing;
    H.room = Human.Room.Bedroom;
    H.roomsection = Human.RoomSection.Door;
    H.loa = Human.LOA.Medium;
    random = Tools.RandomInt(2, 4);
    break;
Integer between 2 and 4

```

Figure 3.4: C# code for the routine

3.4.2 Database

We use *XAMPP*, a free Apache distribution containing *MariaDB*. *MariaDB* is a database management system (DBMS) published under the General Public License (GPL) license. It is a forked version of the well known MySQL. We use *phpMyAdmin*, a free software tool written in PHP, for the administration of the database over the Web.

To connect C# with our database, we use *MySql Connect/NET*. This also allows us to carry out the basic functions of persistent storage that are create, read, update, and delete (CRUD) as well as more complex queries(see appendix 3 for the Data insertion code).

The “*Activity*” table is the main table in the database. It contains the raw data representing what could be retrieved from sensors. This table is composed of the fields representing the states described earlier. It also includes a field with a special timestamp which correspond to the exact insertion time of the activity. Figure 3.5 shows rows of the table *Activity*.

Id	Position	Room	LOA	RoomSection	Time
14529	Standing	Livingroom	Medium	Door	2017-03-08 15:53:55
14530	Standing	Livingroom	Medium	Door	2017-03-08 15:53:56
14531	Standing	Livingroom	Medium	Door	2017-03-08 15:53:57
14532	Standing	Bedroom	Medium	Door	2017-03-08 15:53:58

Figure 3.5: Table Activity

3.4.3 Timer

Windows.Forms.Timer is a Windows object. It uses Timer tick event that is invoked directly into the user interface (UI) thread. To have a quick simulation and to be able to test pertinent situations, we decided to use a timer that executes a method on a thread pool thread at specified intervals.

The simulator is designed in such a way that it can be started and stopped at any time. The user interface lets you choose how many days to add to the database. Figure 3.5 shows the UI of the program.

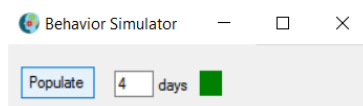


Figure 3.6: UI of the Simulator

3.4 Simulating a morning routine

If the database is not empty, the simulator will match the date of the new entry to the date of the last one in the database incremented by one day. As mentioned previously, a random time is generated for each behaviour. This *random behaviour's time* is different each day. The simulator checks if a behaviour is still running using a *behaviour tick counter* and compares it to the random behaviour's time generated. When these two counters are equal, the simulator switches to the next behaviour in the routine to be inserted in the database. Figure 3.7 shows the behaviour switch in the database.


	Lying	Bedroom	2017-03-03 15:52:43	Low	Bed	
Behavior tick counter	Lying	Bedroom	2017-03-03 15:52:44	Low	Bed	 The simulator switches and populates the database with the next behavior
=						
Random behavior's time	Standing	Bedroom	2017-03-03 15:52:45	Medium	Door	
	Standing	Bedroom	2017-03-03 15:52:46	Medium	Door	

Figure 3.7: Behaviour switch

The morning routine has a defined number of behaviours. It reviews the progress in the routine using a *number of behaviour counter*. When the *number of behaviour counter* reaches the same number of behaviour contained in the routine, the simulator increments by one day the date that will be associated with the behaviours of the routine to follow. When the number of days to be inserted in the database is reached, the simulator disables the timer. Figure 3.8 shows the day switch in the database.


	Sitting	Bathroom	2017-03-03 15:53:07	Low	Toilet	
Number of behavior counter	Sitting	Bathroom	2017-03-03 15:53:08	Low	Toilet	 The simulator switches to the next day
=						
Number of behavior in the routine	Lying	Bedroom	2017-03-04 15:53:09	Low	Bed	
	Lying	Bedroom	2017-03-04 15:53:10	Low	Bed	

Figure 3.8: Day switch

Figure 3.9 shows a flowchart of the implemented algorithm. The startup step represents the interaction of the user with the user interface to start filling the database. The algorithm comes to an end when the database has been filled in for the number of days requested by the user(see appendix 4 for the Timer code).

3 Methodology

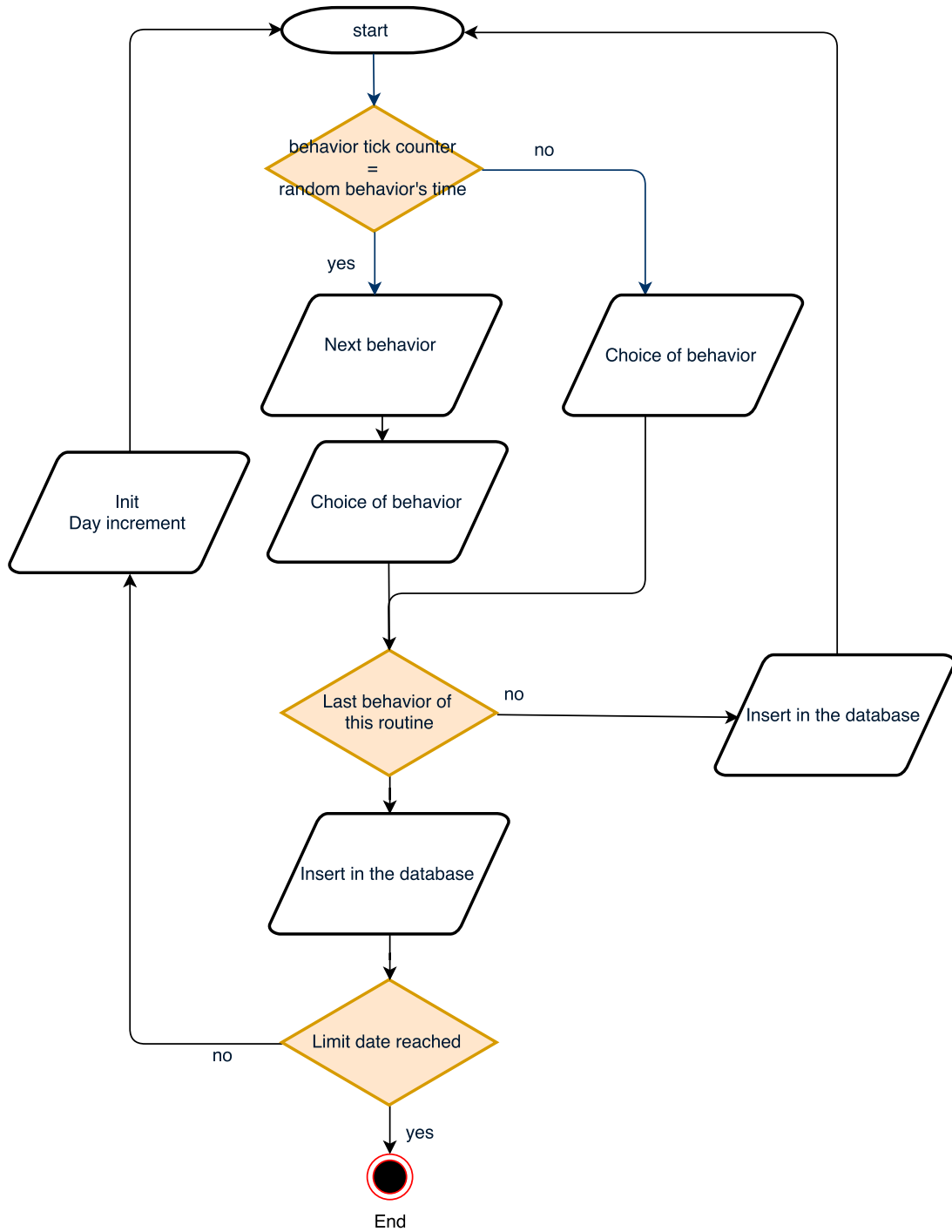


Figure 3.9: Flowchart of the routine code

3.4.4 Codification

To ensure an easy analysis, we decided to translate the behaviours into numerical codes. A very easy way to implement this was to use the enum types previously explained. The first enumerator has by default the value zero. Each successive enumerator is incremented by one. Knowing that we have defined the different states as enum type, it is then very easy to constitute a code from a combination of states. Figure 3.10 shows the codification of the state *Position*.

Position	
Undefined	0
Lying	1
Sitting	2
Standing	3
None	4

Figure 3.10: Codification of the state *Position*

With this codification, the behaviour “*Lying, Bedroom, Low, Bed*” can be transformed for example into “1 1 1 3”. Obviously, if a coding system is set up on the simulator side, then a decoding system must also be implemented (see appendix 5 for the coding and decoding code).

3.4.5 Variation in the routine

The purpose of this part of the simulator is to create behavioural variations in the routine. A possible implementation is to insert new behaviours between fixed and expected ones in the routine to simulate abnormalities. However, physical and spatial constraints must be taken into account. Considering the room where the person is located at a specific time, the next behaviour is constrained spatially according to the configuration of the rooms. Figure 3.11 shows the possible transitions for the configuration showed on Figure 1.2.



Figure 3.11: Behaviour transition

3 Methodology

For example, if a person is in the bathroom, he has to go through the bedroom to reach the kitchen. Moreover, certain transitions and combinations of behaviours are very unlikely, such as moving from the behaviour “Lying Bedroom Low Bed” to “Lying Bathroom High Toilet”.

We implemented this with two List called “*NextPossibleRoom*” and “*NextPossiblePosition*”. Each time a new behaviour is in progress, these lists are updated. The first list containing the possible rooms for the next behaviour is simply updated according to the current room in which the person is located. The second list containing the possible positions for the next behaviour is updated according to the chamber selected in the first list. Indeed, in a situation of a morning routine during the week, it would be unlikely that the person switches from a behaviour “Bedroom Lying Low Bed” to “Livingroom Lying High Couch” behaviour even if the transition of behaviour and state combinations are acceptable.

It is important to understand that the utility of this variation in behaviour within the simulator is essential for the rest of the project. By imagining certain cases of abnormal behaviour to be inserted in our routine, this allows to create material to be analyzed later. We have created two routines. The first one is classic and repeats itself without any real change in behaviour. The second is simply a duplicate except for the addition of abnormal behaviour as described above. Figure 3.12 shows the possible transitions.

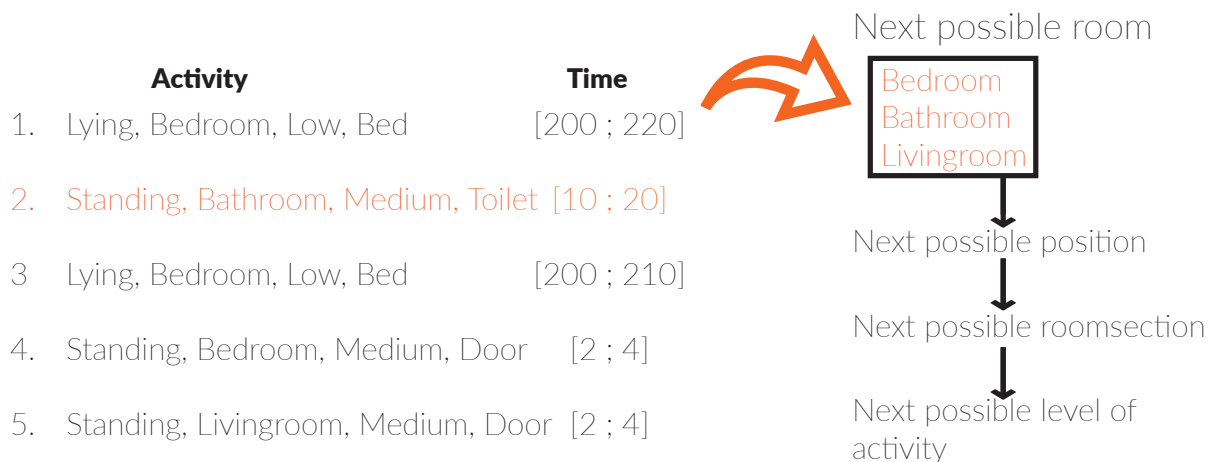


Figure 3.12: Modified routine

3.4.6 Cache database

So far, we have simulated the data that sensors could provide us. In this section, we will process this information to match our definition of the system. It is therefore a matter of detecting the different activities and then delimiting the observable behaviours in the database. This task may seem paradoxical since we have scripted the routine. It is still essential to create a tool that can handle data from a raw data database. We have implemented a method that lists all the different activities and detects the beginning and the end of a behaviour. In addition, if the behaviour has been repeated during the routine, a number corresponding to its position in the routine is assigned to this behaviour. Figure 3.13 shows the cache database.

Id	Behavior	TimeFrom	TimeTo	Appearance
167	1113	2017-03-02 15:52:32	2017-03-02 15:52:41	1
168	3121	2017-03-02 15:52:42	2017-03-02 15:52:44	1
169	3221	2017-03-02 15:52:45	2017-03-02 15:52:48	1
170	3121	2017-03-02 15:52:49	2017-03-02 15:52:52	2

Figure 3.13: Database Cache

3.5 Analysis

3.5.1 Introduction

This section is devoted to the description of the behaviour analysis software. This software must meet several requirements. It must be able to detect a new entry in the database. The program must then retrieve all the information related to this new entry. It must include a part that bridges the gap between data reception and analysis tools.

Our first step in this chapter will be the description of the user interface. The analysis program goes hand in hand with the simulator. It is based on the same definition of the system. This program must be able to analyze specific situations and requires targeting behaviours that interest us. The user interface must therefore be able to meet these expectations. We will then examine the EventHandler system used to manage received inputs in the database. Finally, we will review the Multi-Agent System implemented.

3.5.2 User Interface

Figure 3.14 shows the user interface of the behaviour analysis software. The first element that the user discovers is the “*New Agent*” button. By clicking this button, an Agent is created to follow a specific behaviour. The user is able to know thanks to the “*Simulation state*” at any time whether the simulator is running and therefore if the database is being populated. On the middle part of the program, the user finds 4 listboxes (*System.Windows.Forms.ListBox*) that contain the values of the different states that define the activities and behaviours of a person to be analyzed in the definition of our system.

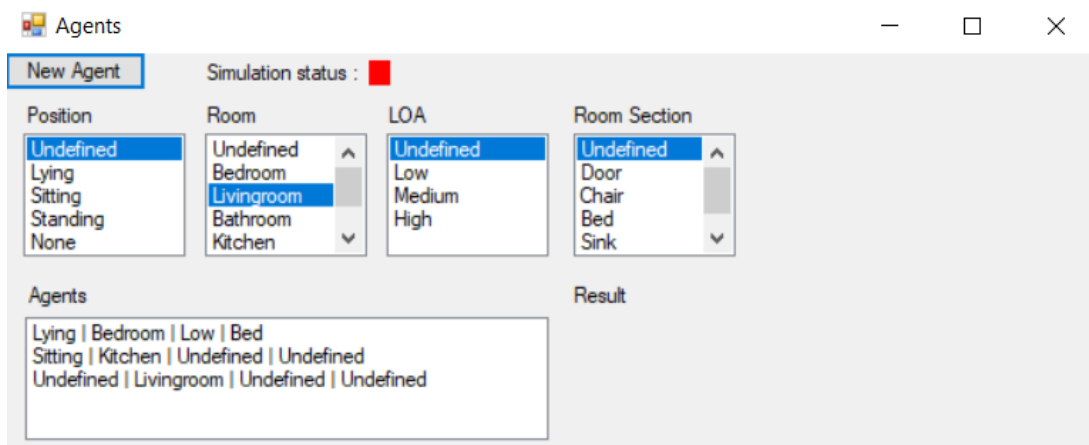


Figure 3.14: UI of the analysis program

These lists are created by directly retrieving the information from the Human class. Figure 3.15 shows how its implemented in C#.

```

Listbox ← lsPosition.DataSource = Enum.GetValues(typeof(Human.Position)); → Enumerator Position
of the Position status lsRoom.DataSource = Enum.GetValues(typeof(Human.Room)); from the class
lsLoa.DataSource = Enum.GetValues(typeof(Human.LOA)); Human
lsRoomSection.DataSource = Enum.GetValues(typeof(Human.RoomSection));

```

Figure 3.15: C# code of the lists implementation

If other states are added within the Human class, it is then very easy to add a corresponding listbox in the analysis program. The selection of a value in each list will compose the state set corresponding to the behaviour to be analyzed by an Agent.

When the “*New Agent*” button is pressed, a new instance of the Human class is created, retrieving the selected state combination from the listboxes. It follows the call to a constructor of the Agent class that accepts as parameter an instance of the Human class and an integer representing an identifier for the Agent. Once the Agent is created, it is placed in a list of Agents. This list is indicated on the GUI (graphical user interface) in a listbox placed at the bottom left. Each line visible in this listbox actually represents an Agent. We have redefined the “*ToString()*” method of the Agent class to display the behaviour that the Agent is supposed to check.

The bottom right part of the program displays the results obtained by the Agents at the end of the simulation.

3.5.3 EventHandler

An *event*[36] in C# is a way to provide notifications to its clients when interesting happenings occur on an object. They provide to the objects a notification of changes of state that can be useful for the clients of these objects. They are an important building block for creating classes that can be reused in many different programs.

EventHandler is often use when dealing with a with a GUI. In our case, we use an Eventhandler to react to a new database entry. When the simulator is running, it adds entries to the database. This addition of input will constitute an *event* for our analysis program. In order to set up an EventHandler, one must understand the concept of delegates.

Delegates are variables that point to a method. A delegate will define a method signature. This way, we can point to any method that respects this signature.

3 Methodology

An EventHandler is a delegate with a special signature. The generic declaration of an EventHandler is shown below:

```
public delegate void MyEventHandler (object sender, MyEventArgs e);
```

It takes as first parameter an “*object sender*” which corresponds to the object that fired the event. The second parameter *MyEventArgs e* contains data that can be used when managing the event. In our case we have created our own specialized EventArgs. This is done by deriving from the EventArgs class :

```
public class ActivityEventArgs : EventArgs  
{  
public Activity Activity { get; set; }   
}
```

We have implemented in the analysis program a first EventHandler named *NewActivityEventHandler* to manage new entries in the database. The second EventHandler *EndActivityEventHandler* manages the end of the simulation and delivers the results obtained by the Agents.

We have build a publisher-subscriber model. The publisher contains the definition of the event and the delegate. Figure 3.16 shows the declaration of the EventHandler and the delegate objects referenced using the key word event.

```
public delegate void NewActivityEventHandler(object source, ActivityEventArgs args);  
public delegate void EndActivityEventHandler(object source, EventArgs args);  
public event NewActivityEventHandler NewActivity;  
public event EndActivityEventHandler EndActivity;
```

Figure 3.16: C# of the delegates implementation

The subscriber accepts the event and provides an event handler. To attach our EventHandler to the event, we use the addition assignment operator. The following example shows an EventHandler method named *OnNewActivity* from the *Agent* class that matches the signature for the EventHandler delegate. The method subscribes to the *NewActivity* event :

```
NewActivity += lsAgent.Last().OnNewActivity;
```

3.5.4 Agents

In order to implement Agents, there is a programming paradigm called Agent-oriented programming (AOP). It is also possible to use a Service-oriented architecture (SOA). In this work, we have chosen an alternative option. We will, however, review the main characteristics of these paradigms and show how we endeavor to draw inspiration from them in the design of our object oriented Agent class.

Agent-oriented programming

AOP was proposed by Yoav Shoham[37] in 1993 as a new paradigm of programming. In this paradigm, Agents are the central elements, in the same way that objects are centered for object-oriented languages. Each Agent is associated with a set of skills and in the AOP, we assume that several Agents will interact. Shoham proposed a programming language called AGENT0 as a demonstration of this paradigm.

The mental notions (Beliefs, Desire and Intention) that characterize the Agents appear in the language itself. Agents can be seen as objects with a state that defines the associated mental notions. Messages between objects are replaced by messages between Agents. These messages are modeled from the theory of speech acts, which focuses on communication actions such as informing, asking, offering, accepting and rejecting. Based on the knowledge they have, Agents have the freedom to decide whether or not they will perform the action specified in the message.

In order to write a program in AGENT0[38], one must first define the initial beliefs and the abilities of the Agents of the program. Then, the implementation of the rules of obligation are necessary to define the behaviour of each Agent. These rules represent the key to this Agent-oriented language and indicate the limiting conditions under which an Agent assumes a certain obligation.

The AGENT0 language interpreter will execute the following control loop for each Agent:

- Read messages received;
- Update beliefs based on these messages;
- Determine obligations by applying the rules of obligation (if the conditions are validated);
- Perform current obligations;
- End.

3 Methodology

The FIPA[39] is a standards-setting organization that promotes the interoperability of applications, services and IT equipment. The most widely used FIPA standard is an ACL (Agent Communication Language) standard . JADE (Java Agent Development Framework) is a software development framework for the developing of multi-Agent systems conforming to FIPA standards which is currently the most used platform for research purpose.

An interesting feature of JADE[40] is that the Agent platform can be split on several hosts. Only one Java Virtual Machine is executed on each host. Agents are implemented as one Java thread and Java events are used for communication between Agents on the same host.

Service-oriented architecture

A service-oriented architecture is based on a set of simple services. We can define a service as a stand-alone component that provides functionality to other applications. These services can be seen as an encapsulated function in a component that can be queried with parameters in order to get responses from it. The services can communicate with each other through buses called Web Service. A Web Service is a web technologies interface protocol allowing the communication and the exchange of data between services. These exchanges can be synchronous or asynchronous. There are two main families of Web Services:

- Simple Object Access Protocol (SOAP)
- Representational State Transfer (REST)

SOAP is an application protocol that consists of two parts. The first part corresponds to an envelope containing information about the message and its routing. The second corresponds to a data model with the specification of its format. A client communicates with a server through different exchanges. First, the server exposes its descriptor named Web Services Description Language (WSDL). The WSDL is an XML-based interface definition language that describes the input parameters of the service, the format and type of the data returned. Then, the client retrieves the descriptor to understand the data that can be sent. Finally, the server receives the request from the client to send a response.

REST is a service-oriented architecture and not a protocol. The client and server work independently. Calls between entities are not dependent on a context held on the server. A call to consume a service with different parameters returns a response in different format (JSON, stream, an image). The queries are written in an URI form with HTTP requests (GET, POST, PUT and DELETE).

An example of using SOA to design a MAS is Flexible User and Services Oriented multi-agent Architecture (FUSION@). The architecture they propose is described as a new and easier method of building distributed multi-Agent systems. The functionalities of the system are not directly integrated into the Agents. They are modeled as services which are invoked by Agents acting as controllers.

Our approach

We have implemented a class named Agent. When instantiating the Agent class, the created object is assigned a state combination. It is possible to consider that a state is not indispensable in the combination to be checked. This state will be “*Undefined*” represented by the symbol “?”. This allows for example the combination “Bedroom ? Low ?” to be checked by an Agent. The “Bedroom Sitting Low Bed” or “Bedroom Lying Low Door” combinations will both be acceptable to this Agent.

As explained previously, the “OnNewActivity” EventHandler informs the Agent of a new database entry. The Agent object then receives the combination of states from the database and compares it to the one it owns. If the combination matches, the Agent object begins its analysis job consisting on retrieving information from the cache database and determining the average time of the behaviour that it analyzes. A key element here is the decoding process. We must consider a state as optional. One way to do this is to use the percentage symbol that represents zero, one, or multiple characters in an SQL query next to a LIKE operator:

```
SELECT * FROM 'tbehaviour' WHERE 'Behavior' LIKE "1%1%"
```

The Agent starts its internal clock and only stops it once the analyzed behaviour is no longer the one the EventHandler notifies. The Agent records the behaviour time in a list. Each time the behaviour reappears in the current routine, a new time is added to the list. When the simulation is complete and the database is no longer populated, the “OnEndActivity” EventHandler notifies the Agent. Since we calculate the mean duration of the behaviour in the course of this routine, we are able to use an exponential distribution, as explained in details in the following section.

Survival analysis - Exponential Distribution

Survival analysis is the estimation of the probability of an event over time, depending on elements influencing the estimate. The expected event is called “death”. Survival analysis will help us to calculate the survival probability at least a certain time “t” from a referential instant.

3 Methodology

Let T be a random variable representing the waiting time until the occurrence of an event. The exponential law is commonly used to represent the lifetime of components for which it is assumed that the failure rate λ is constant over time. The Function of distribution of the exponential law is $F(t)$ and represents the proportion of components that breaks down before time t .

$$F(t) = P(X \leq t) = 1 - e^{-\lambda t} \quad (3.1)$$

We can then define the Survival function $S(t)$. It is the probability that the event of interest “ T ” occurs after a delay greater than “ t ”.

$$S(t) = 1 - F(t) = P(T > t) = e^{-\lambda t} \quad (3.2)$$

This distribution is called the exponential distribution with parameter λ and the mean is equal to $1/\lambda$. Figure 3.18 shows the plot of the exponential survival function with the parameter λ equal to 5.

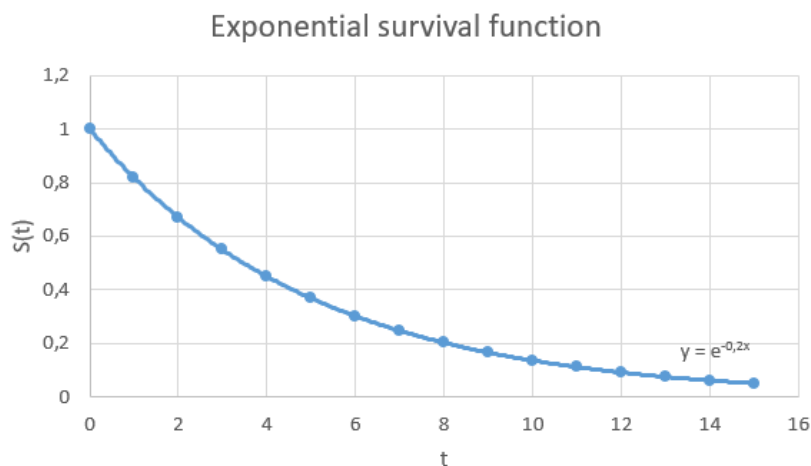


Figure 3.17: Plot of the exponential survival function

We used Accord.NET which is a .NET framework for machine learning, computer vision, statistics and general scientific computing. Figure 3.18 shows an example of how the Survival function is implemented using Accord.NET in C#.

Using this exponential law, we determine for an active behaviour, compared to the average time it has taken in the past, whether it is likely to stop or not. When it stops, this survival value is stored. Finally, with the two probability values obtained, we can already evaluate whether a behaviour is likely to be abnormal or not. The first value is the direct probability that is retrieved as soon as the behaviour comes to an end in the routine. The second value corresponds to the mean of the set of probabilities collected which indicates the predictability of the behaviour.


```

double value = 3;
double meanTime = 5;
double myRate = 1 /meanTime;
var exp = new ExponentialDistribution(rate: myRate);
double survivalValue = exp.ComplementaryDistributionFunction(value);

```

Figure 3.18: C# code to use the survival function

Table 3.1 shows an example of the value that the Agent object collects. The first column refers to the routine, the other columns corresponds to all the parameters collected of an analyzed behaviour during this routine.

Routine	Time t	Survival Function S(t)	Mean Time 1/λ
1	30	1	30
2	28	0,97	29
3	29	1	29
4	31	0,95	29,5
5	29	0,98	29,4
6	30	0,98	29,5
7	15	0,66	27,43

Table 3.1: Example 1 of collected values

It can be seen here that until the seventh routine the time taken by the behaviour is constant. Consequently, the mean time and the survival probability is only slightly modified during the routines. The mean of the probability column is 0.93, which indicates that this behaviour generally takes always the same time to complete.

Routine	Time t	Survival Function S(t)	Mean Time 1/λ
1	12	1	12
2	20	0,88	16
3	10	0,87	14
4	35	0,68	19,25
5	29	0,87	21,2
6	50	0,44	26
7	15	0,73	24,43

Table 3.2: Example 2 of collected values

3 Methodology

Table 3.2 takes on more various time values, implying a more unpredictable ending time for a behaviour. The average of the probability column is 0.78. This clearly indicates a time-related behavioural unpredictability compared to the first example.

We understand from these two examples that we must consider the different survival probabilities over the duration of the existence of a behaviour. Indeed, the simple comparison of the time taken on a given moment with the average time taken by the behaviour analyzed is not relevant enough.

3.5.5 Classification

Once a certain amount of information was collected on the state of behaviour, we had to find a way to evaluate whether the behaviour could be considered normal or not. We considered the use of Fuzzy Logic with the framework AForge.Net. AForge.NET is an open source C# framework in the fields of Artificial Intelligence.

We have implemented a method *getOutputFuzzy* that takes as parameter the survival probability as well as the average survival probability of a behaviour. This method returns an output value allowing to decide whether an alert should be made. First of all, it was important to define the different fuzzy subsets. We defined the sets *lvProbIn* and *AvgProbIn* as Linguistic Variables representing respectively the survival probability and the average survival probability of a behaviour. These sets are composed of the subsets *Bad*, *Normal* and *Good*.

A “fuzzy” database can then be created by adding the Linguistic Variables. To evaluate a response based on Linguistic Variables, rules must be established. These rules are very similar to human language and are composed of causes that produce a consequence. These causes are our linguistic input variables whereas the consequence is the output language variable. Using this database and a set of fuzzy rules, we can create a fuzzy inference system.

First, the system maps numerical values to Linguistic Values with a membership degree to subsets. Then, it calculates the *firing strength* rules using the membership degree of the different Linguistic Values inputs to evaluate their output. The numerical output value is finally obtained by using the *defuzzification* method applied to a fuzzy output language variable.

The form of the membership function is chosen according to the problem posed. In our case, we chose the Trapezoidal model for the survival probability variables. This way, the judgment will be more human. We use three points to construct the shape known as triangular fuzzy number. For the FuzzySet “Normal” we have initialized a new instance of the TrapezoidalFunction class :

```
public TrapezoidalFunction(float m1,float m2,float m3)
```

The parameters *m1*, *m2* and *m3* are the three points used to construct our membership function. *m1* is the value where the degree of membership starts to raise. *m2* is the value where the degree of membership reaches the maximum value and starts to fall, while *m3* is the value where the degree of membership reaches the minimum value.

For the FuzzySets “Bad” and “Good”, we used a parameter of type *TrapezoidalFunction.EdgeType*. If the value of this parameter is Left, the trapezoid has the right edge open. Conversely, if the value is Right, the trapezoid leaves the left edge open.

3 Methodology

Figure 3.19 graphically shows the shape of the membership functions corresponding to the fuzzy variables defined above. We note that the membership function of the survival probability, average survival probability and alert output are the same. These functions will only be represented graphically once.

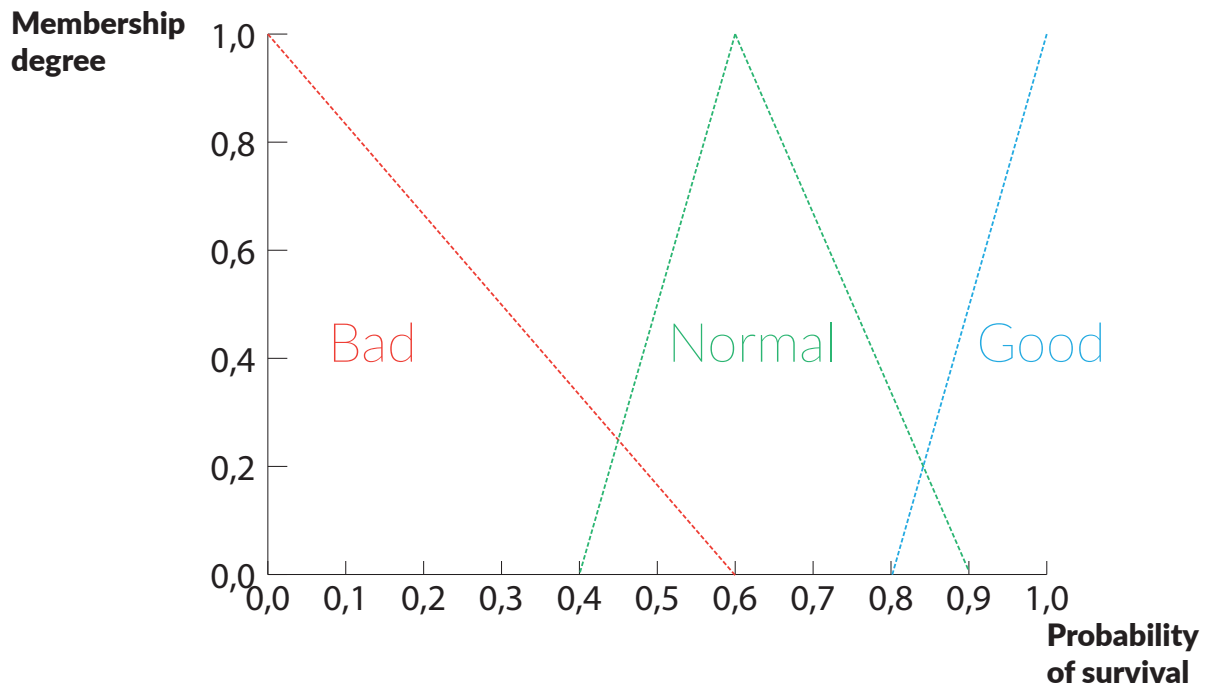


Figure 3.19: Plot of the membership functions

The last step is writing fuzzy rules. Figure 3.20 shows how the rules are implemented in C# using the Aforge.Net framework.

```
IS.NewRule("Rule 7", "IF ProbIn IS Good AND AvgProbIn IS Good THEN alert IS Good");  
IS.NewRule("Rule 8", "IF ProbIn IS Good AND AvgProbIn IS Normal THEN alert IS Normal");  
IS.NewRule("Rule 9", "IF ProbIn IS Good AND AvgProbIn IS Bad THEN alert IS Normal");
```

Figure 3.20: Fuzzy rules

If we consider rule 9, we see that it verifies “AvgProbIn” with a linguistic value corresponding to “bad”, or in other words the mean of the survival probabilities is very low. This suggests that the mean time of the behaviour analyzed is highly variable. Furthermore, even with a low survival probability at a given time, this behaviour is so irregular that creating an alert is not required(see appendix 6 for the classification code).

4 Results

4.1 Testing module

There are several points to test to ensure that the analysis program is functional. The simulator and the database are considered to be functional and unchanged.

The first point to test is the automatic detection of the state of the simulation and hence a new potential input to the database. The input of this test therefore corresponds to the start of the simulator and the output must correspond to the state indicator of the simulator on the analysis program going green. Figure 4.1 shows the state indicator of the simulator on the analysis program.

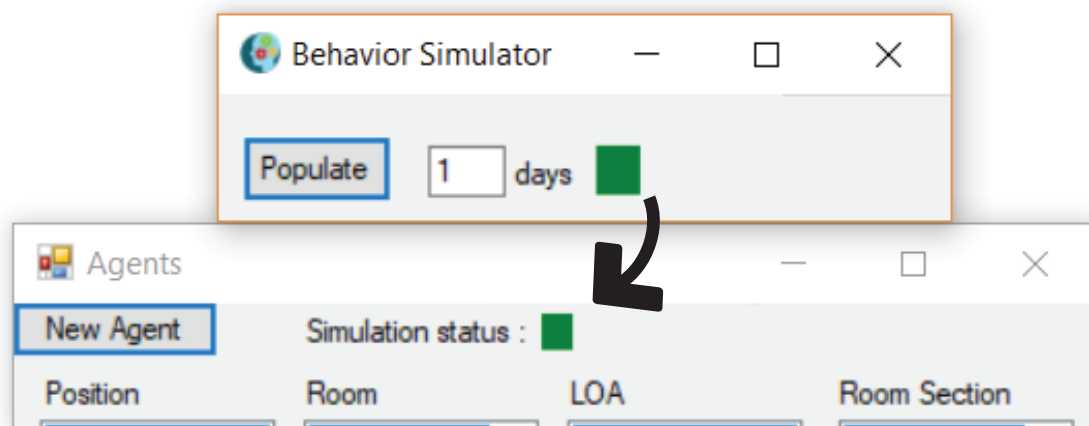


Figure 4.1: State indicator of the simulator on the analysis program

We have implemented a testing module which allows to easily see the results of the classification tool without having to simulate an entire routine. Figure 4.6 shows the C# code for the testing purpose. The testing method allows to directly test the classification tool on input values placed as parameters.

4 Results

```
private void btTest_Click(object sender, EventArgs e)
{
    Human H_test = new Human();
    Agent Ag = new Agent(H_test, 100);
    Ag.Testing(float.Parse(tbProb.Text), float.Parse(tbAvgProb.Text));
    MessageBox.Show("Alert is : " + Ag.PublicInfo);
}
```

Figure 4.2: C# testing code

Using the explanations and considerations of the section “Survival analysis” and “Classification”, we implemented this module to allow us to test the limit values of inputs (survival probabilities and average survival probabilities) of which we know the output (alert level). Table 4.1 shows the pertinent input and output testing values.

Survival probability	Average survival probability	Alert level
1	1	No alert
1	0.1	No alert
0.1	1	Alert
0.5	0.5	Small alert

Table 4.1: Input and output testing values

Figure 4.3 shows the UI testing module and the alert output.

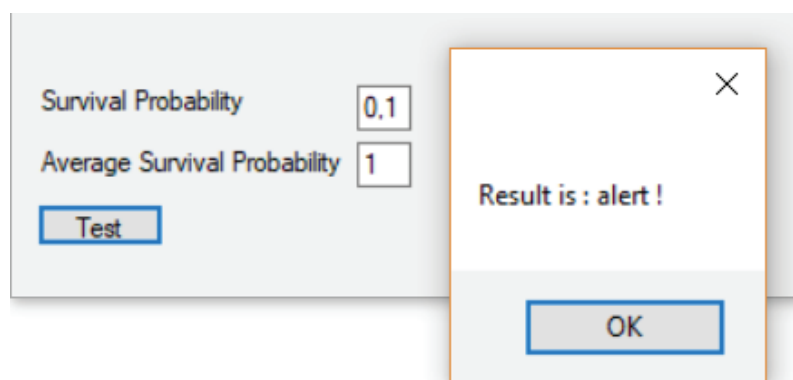


Figure 4.3: UI Testing module

4.2 Situation 1 : Normal routine

Once the simulator and the analysis tool were functional, it was necessary to create interesting situations to carry out different tests. We used for this first test a very simple routine where times of behaviours are not very variable. This ensures that the analysis program does not generate an alert as described in Figure 4.4.

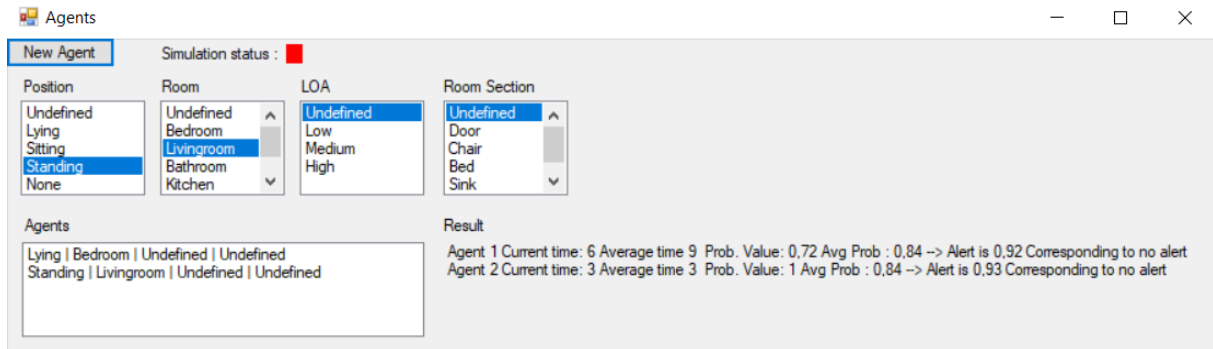


Figure 4.4: Result of the first test

We can see that the Agents collected the different times and survival probabilities of the behaviours. The calculation of the alert value was then calculated in order to assess whether an alert was necessary. Figure 4.5 and Figure 4.6 show the Fire Strenght values of the different fuzzy rules and behaviours analyzed by Agent 1 and 2.

```

Fire Strenght of rule 1 is equal to 0
Fire Strenght of rule 2 is equal to 0
Fire Strenght of rule 3 is equal to 0
Fire Strenght of rule 4 is equal to 0,2132329
Fire Strenght of rule 5 is equal to 0,6330627
Fire Strenght of rule 6 is equal to 0
Fire Strenght of rule 7 is equal to 0
Fire Strenght of rule 8 is equal to 0
Fire Strenght of rule 9 is equal to 0

```

Figure 4.5: Rules' Fire Strenght (Agent 1)

```

Fire Strenght of rule 1 is equal to 0
Fire Strenght of rule 2 is equal to 0
Fire Strenght of rule 3 is equal to 0
Fire Strenght of rule 4 is equal to 0
Fire Strenght of rule 5 is equal to 0
Fire Strenght of rule 6 is equal to 0
Fire Strenght of rule 7 is equal to 0,1900894
Fire Strenght of rule 8 is equal to 0,8760359
Fire Strenght of rule 9 is equal to 0

```

Figure 4.6: Rules' Fire Strenght (Agent 2)

It also appears that Agent 1 has the highest value of fire strength. Rule 5 can be applied as follow:

```

IS.NewRule("Rule 5", "IF ProbIn IS Normal AND AvgProbIn IS Normal THEN alert
IS Good");

```

4 Results

It can be seen that the numerical survival probability (0.72) gave a linguistic value corresponding to “normal” for the Linguistic Variable “ProbIn”.

This test showed that the analysis program manages situations where there is no abnormal behaviour.

4.3 Situation 2 : Irrelevant behaviour time

We use a second situation to test the analysis programs when it faces a behaviour with changing duration. Let us consider, for example, the behaviour “Sitting Bathroom Low Undefined” with a very large time range. Figure 4.7 shows the result from the analysis program.

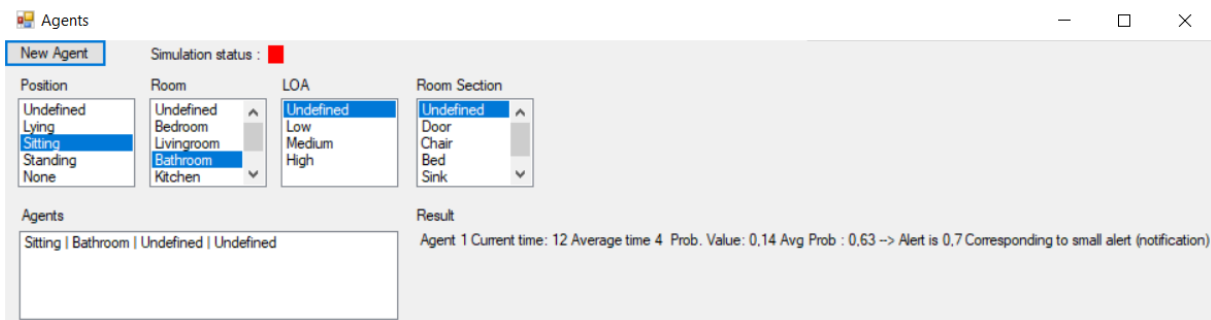


Figure 4.7: Result of the second test

The survival probability is very low given the difference between the time taken on the instant and the mean time. However, as the average survival probability of this behaviour is not very high, there is only a simple notification and not an alert.

4.4 Situation 3 : Abnormal behaviour time

The last situation is divided into two parts. The first part we implemented plays out when a routine has been installed. Then we introduced a particular routine of a day where one of the detected behaviours takes much longer than expected. Figure 4.9 shows the C# code for the two part of this situation.

Two weeks routine	One day
<pre>case 6: H.position = Human.Position.Sitting; H.room = Human.Room.Bathroom; H.roomsection = Human.RoomSection.Toilet; H.loa = Human.LOA.Low; random = Tools.RandomInt(2, 4); break;</pre>	<pre>case 6: H.position = Human.Position.Sitting; H.room = Human.Room.Bathroom; H.roomsection = Human.RoomSection.Toilet; H.loa = Human.LOA.Low; random = Tools.RandomInt(10, 30); break;</pre>

Figure 4.8: C# code of the routine variation

Figure 4.9 shows the result from the analysis program. We observe that the survival probability displayed is round off to zero. Indeed, given that the mean time of this behaviour was 2 minutes and that the average survival probability behaviour is considered “normal” in the fuzzy logic, it is clear that an alert must be made.

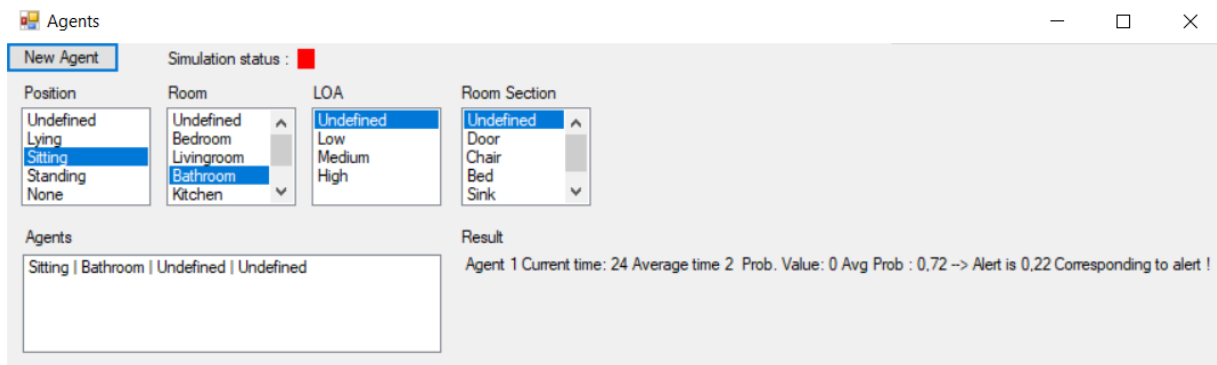


Figure 4.9: Result of the third test

5 Future work

In this part of the thesis, we will discuss possible improvements and potential additions to the simulators and the analysis program. These improvements must be added without causing a redesign of the already established architecture. Figure 5.1 shows the actual architecture of the simulator and analysis program.

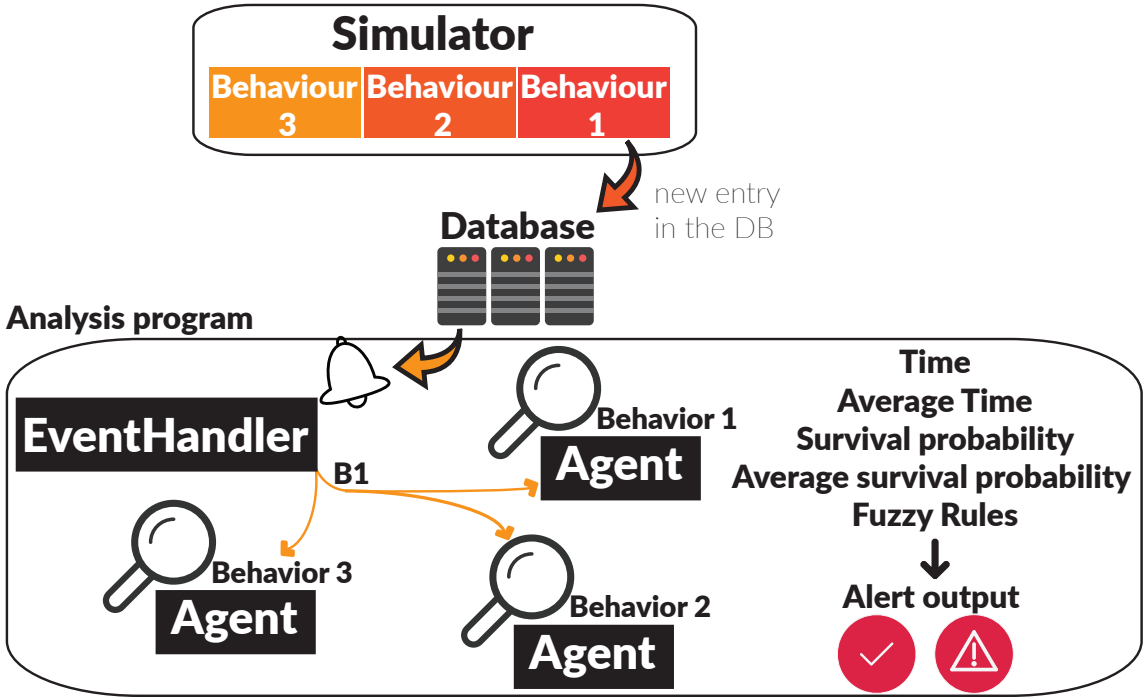


Figure 5.1: Analysis program architecture

5.1 User-friendly routine adding system

The current simulator is not intended for use by anyone. In order to add or modify a routine, one must be able to understand C# code. In addition, a recompilation of the program is required. From a test perspective, this is not a problem. However, if we

5 Future work

start using the simulator more intensively, we must find a more suitable solution for the addition of a customized routine.

A possible solution would be to insert the routine into a CSV (Comma-separated values) file that will then be read by the simulator.

5.2 Sequences analysis

During this work, we have analyzed each behaviour in isolation from the others. However, these behaviours succeed one another after another during a routine. It is this succession that creates complex behaviours. The most widely used tool to analyze these complex behaviours is HMM that we described in the beginning of this thesis.

Due to the architecture of the system, this tool should be included as an add-on to the Agents. This should in no way modify the current behaviour of the Agents or interfered with the other analysis tools. The idea would be to implement an Agent apart from other Agents that analyze simple behaviours. This Agent will intervene when a series of behaviour will have ended in the routine analyzed. It will then collect all the information from other Agents.

Consider, for example, a complex behaviour named *CBH*. *CBH* is composed of behaviours *BH1*, *BH2* and *BH3*. There are several elements to consider when analyzing *CBH*. First, we can ask ourselves the question: is the total time taken by *CBH* “normal” ? This question must be answered in the first place without taking into account if the individual times of the behaviours composing *CBH* are “normal”. The mean times and survival probabilities can be used as in the analysis of simple behaviours. Secondly, it is a question of analyzing the order in the sequence of *CBH*. Indeed, the order “*BH1*, *BH2*, *BH3*” does not have the same probability of appearing in the routine as the order “*BH2*, *BH1*, *BH3*”. It is here that the use of HMM is interesting. We suggest using the Accord.NET Framework for the implementation of HMM. We started to implement it.

```
string[][] complexBehaviour =
    {
        new[] { "1113", "3121", "3221", "3121", "1113" },
        new[] { "1113", "3121", "3221", "3221", "1113" },
        new[] { "3121", "3121", "3221", "3221", "1113" },
        new[] { "1113", "3121", "3221", "3221", "1113" },
    };
```

Figure 5.2: CodeBook

Figure 5.2 shows how we can implement a set of sequences of behaviours in C#. In the future, it would be preferable for these sequences to be directly extracted from the database. The interest here is not to go into details. For this reason, we will limit ourselves to a simple description of the code showed in Figure 5.3.

```
var codebook = new Accord.Statistics.Filters.Codification("Behaviours", complexBehaviours);
int[][] sequence = codebook.Translate("Behaviours", complexBehaviours);
var topology = new Forward(states: 5);
int symbols = codebook["Behaviours"].Symbols;
var hmm = new Accord.Statistics.Models.Markov.HiddenMarkovModel(topology, symbols);
var teacher = new BaumWelchLearning(hmm);
teacher.Learn(sequence);
```

Figure 5.3: C# code for the HMM implementation

Firstly, a codebook is constructed by transforming the set of sequence of behaviours to sequence of integer labels using the codification codebook. Then, we create the training data for the model by specifying a forward topology and the number of states. The model is then created using the topology and the symbols from the codebook. Finally, with the help of the Baum–Welch algorithm, we estimate the model parameters that maximize the probability of observable sequences.

To test the algorithm we have the possibility to generate a sequence from the model and then evaluate its likelihood. Table 5.1 shows the likelihood of three different generated sample. We can observe that the last generated sequence is less likely to appear than the two others.

Generated sample	Likelihood
1113 3121 3221 1113 1113	-1,5
1113 3121 3221 3221 1113	-1,79
3121 3121 3221 1113 3221	-3,99

Table 5.1: Likelihood of generated samples

The idea here would be to add this analysis functionality to the Agent that deals with complex behaviours. This Agent would receive the behaviours order in the sequence from the other Agents. Then, using a model that would be updated with each new routine, the Agent should estimate whether the sequence analyzed was probable or is subject to an alert.

5.3 Frequency of behaviour occurrence

Another line of thought we might pursue is the analysis of the frequency of appearance of behaviours. We had already very briefly discussed this subject in Section 3.3.6. This frequency analysis must be carried out on the entire database. This will allow to add a parameter that will value or not the relevance of a behaviour. Behaviors with a limited number of states will be more frequent. The more states are added to a behaviour, the greater the probability that a combination of states is almost impossible. Imagine, a sensor related to the use of the shower. The behaviour “Lying Bedroom [...] Shower” is very unlikely or that means that there is a problem.

6 Conclusion

The interest in smart houses is gradually increasing . At the same time, the monitoring of elderly population living alone presents a unique challenge for machine learning algorithms. Worldwide, researchers are seeking to develop smart houses technologies to collect pertinent data on a person living alone, which is used for early detection of abnormal behaviours. In this Master Thesis project, we bring an essential algorithmic piece to a data based architecture to build a monitoring system for smart houses currently under development at the University College of South East Norway.

In this study, we presented the complexity of human behaviour analysis in the literature review. By analyzing the different methods of modelling human behaviour such as the use of HMM, SVM, RNN and Fuzzy Logic, we were able to construct our own approach to the problem.

The first major step in this work was the development of a routine simulator. The task of this simulator is to populate a database with entries that simulates data that could be provided by sensors. A preprocessing step then maps these data into discrete state variables. Using the system definitions of Carlos F. Pfeiffer's[13], we built the foundations of our algorithmic logic on a C# class defining the states (*Position*, *Room*, *Loa* and *Roomsection*) of a monitored person. This simulator is able to create behavioural routines relevant to analyze.

The main tool we implemented during this work is the analysis program. We have structured this tool into a part of information retrieval and a part of processing. An effective way of being warned about a new activity is provided by EventHandlers. Once the information was retrieved from the database, we decided to implement a MAS-type architecture to process it. The advantage of using an Agent logic is the flexibility to add analysis methods. We used an exponential distribution to access the survival probabilities of behaviours analyzed on the fly on the database. By coupling these probabilities with Fuzzy Logic as a classification method, we were able to generate an alert depending on the normality of the behaviour.

The simulator and the analysis program are promising because of their complementarity in the construction and analysis of relevant situations. The architecture of the analysis program is designed for the addition of analysis tools such as those mentioned in the *Future Work* chapter (HMM and frequency analysis).

6 Conclusion

In conclusion, our work contributes significantly to the understanding of mechanisms associated with behaviour analysis. Our study should lead to the construction of a powerful tool for the assistance of elderly people living alone. To this end, and as part of the work at the University College of South East Norway, this project represents a short-term hope for an effective and realistic solution.

References

- [1] Histoire de la psychologie. Fort-de-France, Martinique. URL: http://www.univ-ag.fr/modules/module_documents/get-document/default/UFR_Medecine/PACES_cours_UE17/Histoire_de_la_psychologie1.pdf.
- [2] Richard W Burkhardt. Patterns of behavior: Konrad Lorenz, Niko Tinbergen, and the founding of ethology. University of Chicago Press, 2005.
- [3] Lygia Olhweiler, Alexandre Rodrigues da Silva and Newra Tellechea Rotta. “Primitive reflex in premature healthy newborns during the first year”. In: *Arquivos de neuro-psiquiatria* 63.2A (2005), pp. 294–297.
- [4] ”Instinct.” Merriam-Webster.com. dictionary on the Internet. 2017. URL: <https://www.merriam-webster.com/dictionary/instinct..>
- [5] Psychologie générale: Les grands courants de la psychologie. webpage on the Internet. Paris, 2017. URL: <http://ass1.forum-actif.eu/t121-psychologie-generale-les-grands-courants-de-la-psychologie>.
- [6] Douglas L. Hintzman. “MINERVA 2: A simulation model of human memory”. In: *Behavior Research Methods, Instruments, & Computers* 16.2 (1984), pp. 96–101. ISSN: 1532-5970. DOI: 10.3758/BF03202365. URL: <http://dx.doi.org/10.3758/BF03202365>.
- [7] Stephane Daviet. “Etude du comportement humain grâce à la simulation multi-agents et aux méthodes de fouille de données temporelles”. Theses. Université de Nantes, Mar. 2009. URL: <https://tel.archives-ouvertes.fr/tel-00482642>.
- [8] Felix Putze. “Cognitive Modeling: Human Behavior Modeling”. 2012.
- [9] Oxford Dictionaries. dictionary on the Internet. 2017. URL: <https://en.oxforddictionaries.com/definition/us/routine>.
- [10] Nikola Banovic et al. “Modeling and Understanding Human Routine Behavior”. In: *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. CHI ’16. Santa Clara, California, USA: ACM, 2016, pp. 248–260. ISBN: 978-1-4503-3362-7. DOI: 10.1145/2858036.2858557. URL: <http://doi.acm.org/10.1145/2858036.2858557>.
- [11] Care Innovationns. webpage on the Internet. 2017. URL: www.careinnovations.com.

References

- [12] Diane J. Cook, Narayanan C. Krishnan and Parisa Rashidi. “Activity Discovery and Activity Recognition: A New Partnership”. In: *IEEE Trans. Cybern.* 43 (2013), pp. 820–823.
- [13] Carlos F Pfeiffer, Veralia Gabriela Sánchez and Nils-Olav Skeie. “A Discrete Event Oriented Framework for a Smart House Behavior Monitor System”. In: *Intelligent Environments (IE)*, 2016 12th International Conference on. IEEE. 2016, pp. 119–123.
- [14] Oliver Brdiczka et al. “Detecting Human Behavior Models From Multimodal Observation in a Smart Home”. In: *IEEE Trans. Automation Science and Engineering* 6 (2009), pp. 588–597.
- [15] Hamid Medjahed et al. “Human activities of daily living recognition using fuzzy logic for elderly home monitoring”. In: *Fuzzy Systems, 2009. FUZZ-IEEE 2009. IEEE International Conference on.* IEEE. 2009, pp. 2001–2006.
- [16] James C Bezdek. “A convergence theorem for the fuzzy ISODATA clustering algorithms”. In: *IEEE transactions on pattern analysis and machine intelligence* 1 (1980), pp. 1–8.
- [17] Belkacem Chikhaoui, Shengrui Wang and Hélène Pigot. “A frequent pattern mining approach for ADLs recognition in smart environments”. In: *Advanced Information Networking and Applications (AINA)*, 2011 IEEE International Conference on. IEEE. 2011, pp. 248–255.
- [18] Hongqing Fang, Hao Si and Long Chen. “Recurrent neural network for human activity recognition in smart home”. In: *Proceedings of 2013 Chinese Intelligent Automation Conference.* Springer. 2013, pp. 341–348.
- [19] Joseph K Blitzstein and Jessica Hwang. *Introduction to probability.* CRC Press, 2014.
- [20] I. Florescu. *Probability and Stochastic Processes.* Wiley, 2014. ISBN: 9781118593134. URL: <https://books.google.no/books?id=Kdq6BQAAQBAJ>.
- [21] Phil Blunsom. “Hidden markov models”. In: *Lecture notes*, August 15 (2004), pp. 18–19.
- [22] Lawrence R Rabiner. “A tutorial on hidden Markov models and selected applications in speech recognition”. In: *Proceedings of the IEEE* 77.2 (1989), pp. 257–286.
- [23] Brandon Malone. “Hidden Markov Models and Gene Prediction”. In: (2014).
- [24] H-L Lou. “Implementing the Viterbi algorithm”. In: *IEEE signal processing magazine* 12.5 (1995), pp. 42–52.
- [25] Paul M Baggenstoss. “A modified Baum-Welch algorithm for hidden Markov models with multiple observation spaces”. In: *IEEE Transactions on speech and audio processing* 9.4 (2001), pp. 411–416.

- [26] Mehdi Dastani and Leendert van der Torre. “A classification of cognitive agents”. In: Procs. of Cogsci’02 (2002), pp. 256–261.
- [27] Igor Agbossou. “Modélisation et simulation multi-agents de la dynamique urbaine : application à la mobilité résidentielle”. Theses. Université de Franche-Comté, Nov. 2007. URL: <https://tel.archives-ouvertes.fr/tel-00924741>.
- [28] Stuart Russell, Peter Norvig and Artificial Intelligence. “A modern approach”. In: Artificial Intelligence. Prentice-Hall, Egnlewood Cliffs 25 (1995), p. 27.
- [29] Alexandra Degeest. Intelligence Artificielle : Systèmes Multi-Agents. Isib, 2017.
- [30] Lotfi A Zadeh. “Fuzzy logic= computing with words”. In: IEEE transactions on fuzzy systems 4.2 (1996), pp. 103–111.
- [31] Lotfi A Zadeh. “Fuzzy logic systems: Origin, concepts, and trends”. In: Computer Science Division Department of EECS UC Berkeley (2004).
- [32] Raul Rojas. “Fuzzy Logic”. In: Neural networks. Springer, 1996, pp. 289–310.
- [33] Ebrahim H Mamdani and Sedrak Assilian. “An experiment in linguistic synthesis with a fuzzy logic controller”. In: International journal of man-machine studies 7.1 (1975), pp. 1–13.
- [34] L. Rising and N. S. Janoff. “The Scrum software development process for small teams”. In: IEEE Software 17.4 (July 2000), pp. 26–32. ISSN: 0740-7459. DOI: 10.1109/52.854065.
- [35] National Sleep Foundation. webpage on the Internet. 2017. URL: <https://sleepfoundation.org/>.
- [36] Events Tutorial (C#). webpage on the Internet. 2017. URL: [https://msdn.microsoft.com/en-us/library/aa645739\(v=vs.71\).aspx](https://msdn.microsoft.com/en-us/library/aa645739(v=vs.71).aspx).
- [37] Yoav Shoham. “Agent-oriented programming”. In: Artificial intelligence 60.1 (1993), pp. 51–92.
- [38] Yoav Shoham. “AGENT0: A Simple Agent Language and Its Interpreter.” In: AAI. Vol. 91. 1991, p. 704.
- [39] Foundation for Intelligent Physical Agents. Specifications. webpage on the Internet. 2017. URL: <http://www.fipa.org>.
- [40] Fabio Bellifemine, Agostino Poggi and Giovanni Rimassa. “JADE—A FIPA-compliant agent framework”. In: Proceedings of PAAM. Vol. 99. 97-108. London. 1999, p. 33.
- [41] K. Viard et al. “An Event-Based Approach for Discovering Activities of Daily Living by Hidden Markov Models”. In: 2016 15th International Conference on Ubiquitous Computing and Communications and 2016 International Symposium on Cyberspace and Security (IUCC-CSS). Dec. 2016, pp. 85–92. DOI: 10.1109/IUCC-CSS.2016.020.

References

- [42] Enric Junqué de Fortuny. “Detecting patterns in human behaviour and operationalization of predictive models”. Theses. Universiteit Antwerpen, Faculteit Toegepaste Economische Wetenschappen, 2014.
- [43] S.V. Vaseghi. *Advanced Digital Signal Processing and Noise Reduction*. Wiley, 2008. ISBN: 9780470740163. URL: <https://books.google.no/books?id=vVgLv0ed3cgC>.

Appendices

Appendix 1: Project task description

Appendix 2: Class Human

Appendix 3: Data insertion code

Appendix 4: Timer code

Appendix 5: Coding and decoding code

Appendix 6: Classification code

Appendix 1: Project task description



FMH606 Master's Thesis

Title: Discrete Events Modelling of a Person Behaviour at Home

HSN supervisor: Carlos F. Pfeiffer

Co-supervisor: Nils-Olav Skeie

Task background:

Society faces demanding care challenges in the coming decades, as there will be a limited supply of health care personnel and volunteer caregivers, while the number of elderly people will increase. In the case of Norway, forecasts for the national context of population statistics from Statistics Norway suggests that there will be more than a million people over 67 years in Norway in 2030, and many of these people will live alone.

According with the same report, the most common challenges and problems of aging people living alone are situations of having an accident and not being able to ask for assistance (with falls being the most common accident), feelings of loneliness, and having episodes of cognitive impairment with no one noticing.

There is also a growing population of senior people who may experience mid cognitive impairment or dysfunction episodes, manifested as odd behaviors (skipping meals during the day, walking out at late night, long periods of immobility during the day, etc.), or subtle changes of normal routine. People with cognitive disabilities (such as Alzheimer's disease or other forms of dementia) have difficulty completing activities of daily living (ADLs) e.g. eating, bathing, and dressing. As long as these episodes are short and infrequent, many older people still manage to be self-sufficient and live alone in their own homes, but only if they are closely monitored.

At HSN we are working to develop a system to assist elderly people living by themselves, focusing on the detection of accidents (falling, fires, dangerous house temperatures, etc.), the detection of cognitive impairment or dysfunction situations, and the immediate reporting of these events to family members or to a caregiver service.

An important task of this system is to monitor the position (lying, sitting, standing up) and level of activity of a person (not moving, moving slowly, moving fast). This information is used to model the behaviour of the person and detect possible abnormal or dangerous situations.

Task description:

- Carry on a literature review on different methods for modelling stochastic discrete event processes, applied to modelling human behavior.
- Describe different stochastic methods for discrete events modelling, indicating advantages and disadvantages on their use to model human behaviour.

Address: Kjølnes ring 56, NO-3918 Porsgrunn, Norway. **Phone:** 35 57 50 00. **Fax:** 35 55 75 47.

- Test different methods for modelling human behaviour using simulated events information from a database.
- Write a master thesis report.

Student category: IIA

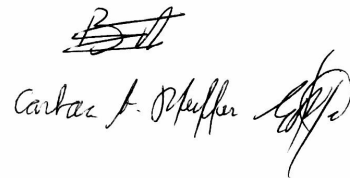
Practical arrangements:

HSN will provide office space, software and hardware required for the project. Most of the work should be carried on at the Porsgrunn campus.

Signatures:

Student (date and signature): Badreddine Cherradi

Supervisor (date and signature): Carlos F. Pfeiffer



The image shows two handwritten signatures. The first signature is a stylized, cursive 'B' followed by 'C', representing Badreddine Cherradi. The second signature is 'Carlos F. Pfeiffer' written in a cursive script, followed by a large, circular flourish.

Appendix 2: Class Human

```
public class Human
{
    public enum Position { Undefined, Lying, Sitting, Standing, None };
    public enum Room { Undefined, Bedroom, Livingroom, Bathroom, Kitchen, Outside, None };
    public enum LOA { Undefined, Low, Medium, High }; // LevelOfActivity // Medium = Walking for example
    public enum RoomSection { Undefined, Door, Chair, Bed, Sink, Toilet }

    public Position position;
    public Room room;
    public LOA loa;
    public RoomSection roomsection;
}
```

Appendix 3: Data insertion code

```
private void InsertRaw(string position, string room, string loa, string roomsection)
{
    string query = "INSERT INTO tbactivity (Position, Room, Time, LOA, RoomSection) " +
        "VALUES(@paramPosition, @paramRoom,@paramDay,@paramLOA,@paramRoomSection)";

    if (this.OpenConnection() == true)
    {
        MySqlCommand cmd = new MySqlCommand(query, connection);
        cmd.Parameters.AddWithValue("@paramPosition", position);
        cmd.Parameters.AddWithValue("@paramRoom", room);
        cmd.Parameters.AddWithValue("@paramLOA", loa);
        cmd.Parameters.AddWithValue("@paramRoomSection", roomsection);

        DateTime today = DateTime.Parse(list[3].Last());
        second++; //Allows us to have a new entry every second.
        DateTime tmrw = today.AddDays(day + 1).AddSeconds(second); //if day is incremented, new day routine

        cmd.Parameters.AddWithValue("@paramDay", tmrw.ToString("yyyy-MM-dd H:mm:ss"));
        lastAddedAcTime = tmrw.ToString("yyyy-MM-dd H:mm:ss");
        cmd.ExecuteNonQuery();
        this.CloseConnection();
    }
}
```


Appendix 4: Timer code

```
private void timer__Tick(object sender, EventArgs e)
{
    OldActivity = H.position.ToString() + " " + H.room.ToString() +
        " " + H.loa.ToString() + " " + H.roomsection.ToString();

    if (count > random) //Check if it's still the same behaviour
    {
        action++; //Increment for the next behaviour
        count = 0;
    }
    RecordAction();//Select the next behaviour
    if (action == 7) // Number depend on how many behaviour are in the routine (To change in a futur dev)
    {
        Console.WriteLine("day = " + day + " action = " + action);
        if (day == Int32.Parse(tbDay.Text)-1)
        {
            start = false;
            timer_.Enabled = false;
            pbState.BackColor = Color.Red; //Simulator state indicator
            timer_.Dispose();
        }
        else
        {
            Init();
            day++; // Increment day
        }
    }
    else {
        count++;
        string currentAct = H.position.ToString() + " " + H.room.ToString() +
            " " + H.loa.ToString() + " " + H.roomsection.ToString();

        InsertRaw(H.position.ToString(), H.room.ToString(), H.loa.ToString(), // Insert in the database
            H.roomsection.ToString());
    }
}
```

Appendix 5: Coding and decoding code

```
private string ConvertToCode(int i)
{
    /*Convert an activity into a code. For example : "Lying Bedroom Low Bed" into 0 0 0 3
    * where those numbers correspond to the index in the enumType (defined in the Human Class) */
    string code = "";
    Human.Position position = (Human.Position)Enum.Parse(typeof(Human.Position), list[1][i]);
    int p = (int)position;
    Human.Room room = (Human.Room)Enum.Parse(typeof(Human.Room), list[2][i]);
    int r = (int)room;
    Human.LOA loa = (Human.LOA)Enum.Parse(typeof(Human.LOA), list[4][i]);
    int l = (int)loa;
    Human.RoomSection roomsection = (Human.RoomSection)Enum.Parse(typeof(Human.RoomSection), list[5][i]);
    int rs = (int)roomsection;

    code += p.ToString() + r.ToString() + l.ToString() + rs.ToString();
    return code;
}

private string CodeToText(string code)
{
    Human.Position position = (Human.Position)Int32.Parse(code.Substring(0, 1));
    Human.Room room = (Human.Room)Int32.Parse(code.Substring(1, 1));
    Human.LOA loa = (Human.LOA)Int32.Parse(code.Substring(2, 1));
    Human.RoomSection roomsection = (Human.RoomSection)Int32.Parse(code.Substring(3, 1));

    return position.ToString()+ "/" + room.ToString() + "/" + loa.ToString() + "/" + roomsection.ToString();
}
}
```

Appendix 6: Classification code

```
public static float getOutputFuzzy(float prob, float avgprob)
{
    //Input
    LinguisticVariable lvProbIn = new LinguisticVariable("ProbIn", 0.0f, 1f); // Linguistic Variable

    FuzzySet fsBadIn = new FuzzySet("Bad",
        new TrapezoidalFunction(0.0f, 0.6f, TrapezoidalFunction.EdgeType.Right)); // FuzzySet
    FuzzySet fsNormalIn = new FuzzySet("Normal",
        new TrapezoidalFunction(0.4f, 0.9f, 0.6f)); // FuzzySet
    FuzzySet fsGoodIn = new FuzzySet("Good",
        new TrapezoidalFunction(0.8f, 1f, TrapezoidalFunction.EdgeType.Left)); // FuzzySet
    //Lv - Fs
    lvProbIn.AddLabel(fsBadIn);
    lvProbIn.AddLabel(fsNormalIn);
    lvProbIn.AddLabel(fsGoodIn);

    LinguisticVariable lvAvgProbIn = new LinguisticVariable("AvgProbIn", 0.0f, 1f);
    FuzzySet fsAvgBadIn = new FuzzySet("Bad",
        new TrapezoidalFunction(0.0f, 0.6f, TrapezoidalFunction.EdgeType.Right));
    FuzzySet fsAvgNormalIn = new FuzzySet("Normal",
        new TrapezoidalFunction(0.4f, 0.9f, 0.6f));
    FuzzySet fsAvgGoodIn = new FuzzySet("Good",
        new TrapezoidalFunction(0.8f, 1f, TrapezoidalFunction.EdgeType.Left));
    // Linguistic labels (fuzzy sets) that compose the alert
    lvAvgProbIn.AddLabel(fsAvgBadIn);
    lvAvgProbIn.AddLabel(fsAvgNormalIn);
    lvAvgProbIn.AddLabel(fsAvgGoodIn);

    // Ouput
    LinguisticVariable lvAlertOut = new LinguisticVariable("alert", 0.0f, 1f);

    FuzzySet fsBadOut = new FuzzySet("Bad",
        new TrapezoidalFunction(0.0f, 0.6f, TrapezoidalFunction.EdgeType.Right));
    FuzzySet fsNormalOut = new FuzzySet("Normal",
        new TrapezoidalFunction(0.4f, 0.9f, 0.6f));
    FuzzySet fsGoodOut = new FuzzySet("Good",
        new TrapezoidalFunction(0.8f, 1f, TrapezoidalFunction.EdgeType.Left));

    lvAlertOut.AddLabel(fsBadOut);
    lvAlertOut.AddLabel(fsNormalOut);
    lvAlertOut.AddLabel(fsGoodOut);

    // Database
    Database fuzzyDB = new Database();
    fuzzyDB.AddVariable(lvProbIn);
    fuzzyDB.AddVariable(lvAlertOut);
    fuzzyDB.AddVariable(lvAvgProbIn);
}
```

```

// creating the inference system
InferenceSystem IS = new InferenceSystem(fuzzyDB, new CentroidDefuzzifier(100));

// Rules

IS.NewRule("Rule 1", "IF ProbIn IS Bad AND AvgProbIn IS Good THEN alert IS Bad");
IS.NewRule("Rule 2", "IF ProbIn IS Bad AND AvgProbIn IS Normal THEN alert IS Bad");
IS.NewRule("Rule 3", "IF ProbIn IS Bad AND AvgProbIn IS Bad THEN alert IS Normal");

IS.NewRule("Rule 4", "IF ProbIn IS Normal AND AvgProbIn IS Good THEN alert IS Good");
IS.NewRule("Rule 5", "IF ProbIn IS Normal AND AvgProbIn IS Normal THEN alert IS Good");
IS.NewRule("Rule 6", "IF ProbIn IS Normal AND AvgProbIn IS Bad THEN alert IS Normal");

IS.NewRule("Rule 7", "IF ProbIn IS Good AND AvgProbIn IS Good THEN alert IS Good");
IS.NewRule("Rule 8", "IF ProbIn IS Good AND AvgProbIn IS Normal THEN alert IS Good");
IS.NewRule("Rule 9", "IF ProbIn IS Good AND AvgProbIn IS Bad THEN alert IS Good");

// Setting inputs
IS.SetInput("ProbIn", prob);
IS.SetInput("AvgProbIn", avgprob);

// Setting inputs
IS.SetInput("ProbIn", prob);
IS.SetInput("AvgProbIn", avgprob);

// Getting outputs
float alert = -1;
try
{
    alert = IS.Evaluate("alert");

    for (int i = 1; i < 10; i++)
    {
        Console.WriteLine("Fire strength of rule {0} is equal to {1} \n",
            i, IS.GetRule("Rule " + i.ToString()).EvaluateFiringStrength());
    }
}
catch (Exception)
{
    //nothing
}

return alert;
}

```