User Modeling and User-Adapted Interaction (2020) 30:895–947
https://doi.org/10.1007/s11257-020-09266-4

Modeling the behavior of persons with mild cognitive impairment or Alzheimer's for intelligent environment simulation

Yannick Francillette, et al. ***[full author details at the end of the article]***

Received: 14 February 2018 / Accepted in revised form: 28 May 2020 / Published online: 20 June 2020

Abstract

Intelligent environments may improve the independence and quality of life of persons with mild cognitive impairment (MCI) or Alzheimer's disease (AD) through their ability to automatically provide assistance or guidance. In order to deploy these systems in this category of the population, it is necessary to be able to carry out validation and experiments to improve efficiency, safety, user experience and reduce installation costs. Unfortunately, this type of experiment can be difficult to perform because of the difficulty in recruiting candidates and accessing adequate intelligent environments. These problems could be partially offset with simulators. These tools can be used to simulate the behavior of an intelligent environment and its occupants in order to generate data, or to observe and evaluate their behavior. However, to design systems for populations suffering from MCI or AD, it is necessary that the simulator be able to emulate the behavior of these persons. In this paper, two approaches to simulate and generate sequences of actions containing errors usually committed by persons with this type of disease are proposed. Those approaches aim to be simple to use and both are based on the use of behavior trees. The first one consists in adding nodes to a behavior tree to simulate errors with their specific probabilities. The second approach consists in defining an interval to bind the number of errors that can be inserted through the error injection algorithm. We also present the results of the experiments carried out to evaluate these approaches. For the first experiment, several simulations were conducted and were recorded in videos. These videos were analyzed by specialists in cognitive disorders who diagnosed the avatar of these videos. The second experiment aimed at comparing the two approaches together. To do so, several action sequences were generated. The results show that our model is able to generate healthy, MCI and Alzheimer's behaviors. The results also show that the second approach facilitates the generation of a desired number of errors.

Keywords Simulation · Simulation of activity of daily life · Error simulation · Intelligent environment · Mild cognitive impairment · Alzheimer

1 Introduction

Advanced technologies in the field of sensors, wireless technologies and ambient intelligence allow us to design autonomous assistance systems integrated in the environment of individuals. These systems are called Intelligent Environments (IEs) and are generally designed to make daily life easier by automating certain chores and controlling various devices such as heating or lighting to increase comfort. They are also useful in resource management because they can minimize energy consumption by managing the operation of electrical appliances, for example (Anvari-Moghaddam et al. 2015; Jiang and Fei 2015). Consequently, IEs can potentially have a great impact on the lives of individuals, particularly by increasing well-being at home, but also on societies in general by allowing a more efficient management of resources (Leitner 2015).

Intelligent environments are particularly promising in the field of healthcare to assist people with disabilities (Bouchard et al. 2014; Lotfi et al. 2012). Indeed, the support that an intelligent environment provides to individuals with disabilities could allow them to gain independence and increase their well-being (Morris et al. 2013). In addition, it could have several positive consequences for societies (Bellagente et al. 2018). By increasing the independence of these individuals, informal caregivers or home support workers are less in demand and could be affected more effectively. Besides, they can contribute to patient monitoring by providing healthcare staff with accurate and objective information about aspects of their patients' daily lives (Aramendi et al. 2018; Do et al. 2018).

In order to be able to offer and install these systems in the environment of this targeted population, it is necessary to work on reducing their cost, maximizing the user experience, minimizing the sensation of presence and increasing reliability and robustness (Brush et al. 2011). IEs can be particularly well-adapted to correct errors and help persons with cognitive impairments. However, it is necessary to carry out experiments in order to personalize them effectively. These points require, in particular, the carrying out of laboratory and field tests, but the setting up of this type of experiment can be very difficult. Indeed, some researchers may not have an intelligent environment laboratory for various reasons (logistics, economics, etc.). Moreover, it may be difficult to access participants from the target population to conduct these experiments.

Researchers can use IE simulators to solve the problem of accessing physical infrastructures (Synnott et al. 2015). These tools can be used to build virtual environments and to simulate the behavior of different sensor types. This allows, for example, to use different scenarios to generate data, to study the evolution of the IE, or to test new approaches to detecting activity. Basically, the scenarios played in a simulator are intended to emulate human activities by generating coherent and plausible interactions between virtual occupants and various objects in the environment.

There are two approaches to emulate an occupant's behavior in a simulator (Synnott et al. 2015). The first method consists of creating an interface to allow a human to interact with the elements of the virtual environment in order to imitate an interaction scenario. However, so-called interactive approach has several disadvantages in our context. The first is the need to recruit a human to play the scenario over a period of time that can be quite long. The second limitation, and perhaps the most significant, is caused by the difficulty for a non-expert human to imitate the behavior of people with a mild cognitive impairment (MCI) or Alzheimer's disease (AD). The second solution is called "model-based approach" and consists in setting up a computer model that is used to simulate interactions. This approach makes it easier to perform and to repeat experiments. Therefore, it is no longer necessary to recruit a human to perform actions to generate data. In this paper, we propose two approaches to inject errors in a simulation to emulate the behavior of persons with MCI or AD. The behavior of this part of the population is characterized by difficulties when performing their activities of daily living (ADLs) (Belchior et al. 2015; Schmitter-Edgecombe and Parsey 2014). Thus, the research question is: How to easily generate plausible action sequences similar to those performed by the targeted populations? The goal is to propose a model that offers a formalism that is easy to read and to use even for non-experts. This paper is a continuation of our work on human activity modeling presented in the papers Francillette et al. (2017) and Bouchard et al. (2018). In these papers, we have introduced a way of representing the human activities of daily life through the formalism of behavior trees. With this model and the use of an IE simulator (Francillette et al. 2017), we can simulate ADLs in order to generate logs for testing. We propose an extension of this work to simulate ADLs performed by people with MCI and AD. In order to allow more errors to be simulated, a memory management system for the avatar was added to this previous model. From this model, we propose two approaches to simulate the creation of errors in the performance of the ADLs. One is based on predefined error probabilities independently associated with each action and with the memory management system. The other is based on error intervals instead of simple probabilities per action. We propose two experiments to evaluate these approaches.

This paper is organized as follows. The next section presents the background and prerequisites for this work. Section 3 presents the literature review and the problem of current methods. Section 4 presents the behavioral model that forms the basis of our error injection approach. Section 5.1 presents our first error injection proposal. Section 5.2 presents our second error injection proposal. Section 6 presents the experiments conducted to evaluate the models. Section 7 presents a general discussion of our approach. The final section presents the conclusion and future work.

2 Background

In this section, we present the three important concepts related to this research. More specifically, the intelligent environments, the human activity modeling and the behavior simulation and the agent behavior modeling are defined.

2.1 Intelligent environment

Essentially, an intelligent environment can be defined as a daily living space, i.e., an apartment, an office, a room, etc., to which are added a set of sensors and actuators that provide information about the state of the environment and to act upon it. These sensors and actuators are, in turn, controlled by an Artificial Intelligence (AI) that uses them to meet certain objectives defined by the IE designers. The objectives may be to minimize energy consumption in a home while maximizing comfort or, in the case of our study, to detect errors and to inform the occupant or address them (Chen et al. 2012).

For example, they maintain an interactive loop that can be represented as a dialogue between the system and the user in which the system uses (Crawford 2012): (1) a set of sensors to "listen" to the user's actions (2) an AI to "think" about a response and (3) actuators to "respond" by acting on the environment. The occupant, on the other hand, uses his/her senses to perceive the system's actions, interpret them and then act accordingly (Hornbæk and Oulasvirta 2017). Figure 1 shows this interactive loop.

The AI is an important component of intelligent environments. In fact, it is the element that distinguishes basic home automation and IEs (Brush et al. 2011). In both cases, sensors and actuators are present in the environment. However, in the context of home automation, the occupant usually has a device that allows him/her to remotely control the actuators and possibly have the information provided by the sensors (Pavithra and Balakrishnan 2015). For example, he/she can use his/her

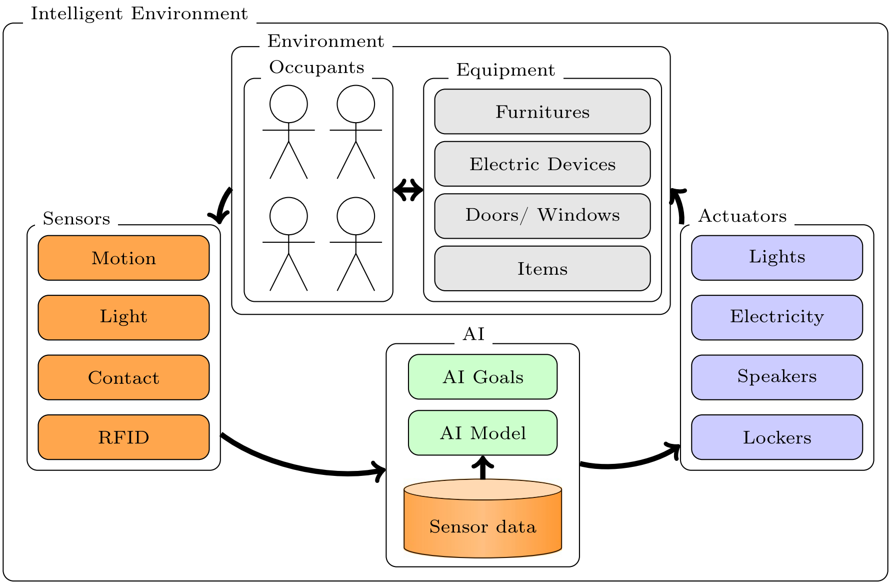


Fig. 1 Diagram of the main components of an IE. The environment group includes occupants and environmental equipment. These two subgroups interact with each other. AI: artificial intelligence; RFID: radio-frequency identification

smartphone to remotely lock the doors and windows of his/her home. He/she can also have information about the internal temperature and humidity level so that he/she can choose to turn on and configure the heater. In the case of an IE, the AI decides whether or not to lock the doors/windows and the heating configuration using the data supplied by the sensors and according to its objectives.

Specifically, in an IE, occupants perform a whole range of activities. These ADLs take the form of actions carried out on the various equipments. The system AI uses data generated by the various sensors to detect changes in equipment status and infer the occupant's current activity (Chen et al. 2011, 2012; Hallé et al. 2016). In addition to ADLs recognition, it must identify possible errors so it can respond with proper assistance or an alert. For example, it would try to detect whether the occupant has the wrong utensil when performing a subtask of a kitchen activity or if the user forgot to put coffee in his coffee machine before triggering it. Figure 2 shows the process flow for these systems.

It can be seen that recognition of activity is an important step in the assistance process. Indeed, in order to be able to provide the right assistance, it is important to correctly identify ongoing activities and errors.

2.2 Models of human activity for recognition

Modeling human activities is a key step for automated assistance. Indeed, these systems are based on activity libraries in order to recognize the activity in progress and determine the need for assistance. Thus, the type of model used to represent activities affects a system's ability to infer information from these libraries.

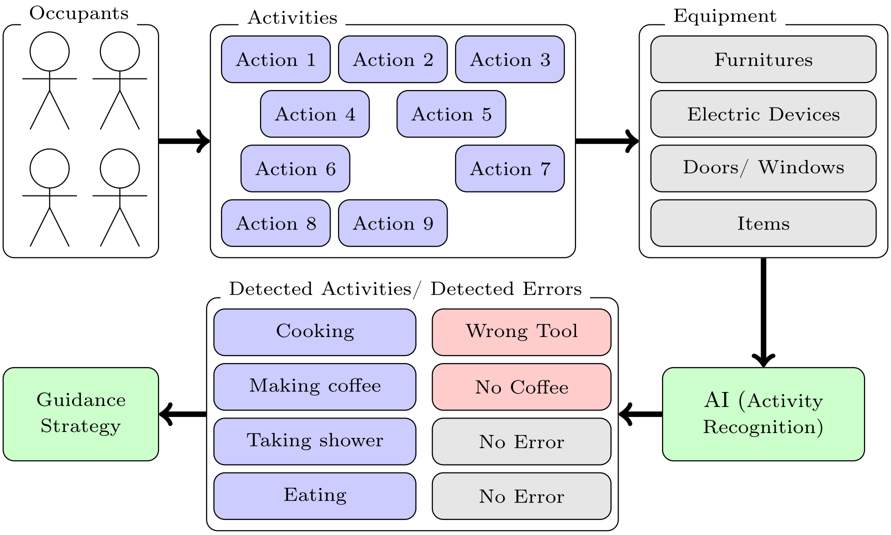


Fig. 2 Process flow diagram in an intelligent environment. Actions on the same line are linked to the same activity and are organized according to a temporal logic. Each line represents actions in parallel

This is one of the reasons why several approaches can be found in this field to model human activities.

These approaches can be classified into three families according to Bouchard et al. (2018) The first family includes activity models that have representations directly based on data from different sensors placed in the environment (Chen et al. 2012). Thus, a model consists in defining an activity according to the way the sensors are triggered. Formally, an activity a is defined by a sequence $\{s_i, \dots, s_j\}$ where $s_i, s_j \in S$ and S is a set of sensors $\{s_1, \dots, s_k\}$ that represents the set of detectable occupant actions. For example, if we have in our environment contact sensors (*cs*), RFID antennas (*rfid*)(which can be used to estimate the position of an object with an RFID tag on it), electricity consumption sensors (*ec*) and pressure plates (*pp*), our S set could be $\{cs_1, cs_2, \dots, rfid_1, rfid_2, \dots, ec_1, ec_2, \dots, pp_1, pp_2\}$. So, a coffee-making activity can be modeled by the sequence $\{cs_3, rfid_2, rfid_4, ec_1\}$. The major problem with this type of modeling is that there is no real abstraction of activity. These are linked to sensor data, therefore, if the sensors in the environment are changed (e.g., by changing their number, type or positions) the modeling may become obsolete or inconsistent. This implies that modeling activities to simulate them in a virtual environment requires knowing its composition.

The second family includes models that are based on logical representations. In this type of model, an activity is represented by basic actions and logical axioms that define the relationships between these actions. Different theories are used in the literature, Kautz (1991) and Camilleri (1999) used first-order logic, and Wobcke (2002) used situation theory. These approaches make it possible to model an activity without having prior knowledge of all the environmental sensors and they offer a high-level expressiveness. However, these approaches are very complex and modeling is often a computational challenge. Thus, they can be difficult to use for neophytes.

The third and final family of approaches includes stochastic modeling. These combine theories from the field of graphs and probability theory to represent compactly observable phenomena in the real-world. Among the approaches in the literature, one can find methods using Bayesian Networks (BN) such as Charniak and Goldman (1993); Patterson et al. (2005), Hidden Markov models (HMM) (Geib and Goldman 2005; Grześ et al. 2014) or Conditional Random Fields (CRF) (Nazerfard et al. 2010). Generally, this family of approaches is based on learning from a dataset. These datasets are generally obtained from the sensors during the interaction between the sensors and the occupant. This is a considerable problem in our context. Indeed, it becomes difficult to add or modify an activity because it implies a relearning of the probabilistic distribution. Thus, the creation of new scenarios is laborious. However, using graph-based tools allows improving models readability and intelligible.

We also find these families of approaches in the field of plan recognition. Plan recognition can be defined as the higher level of abstraction and complexity of activity recognition. It is about identifying the reason that drives an agent to carry out a set of activities/actions. As Schneider points out (Schneider 2010), there are approaches based on logic and those based on probabilities.

2.3 Behavior modeling, agent-oriented approach

The problem of simulating individuals' behavior can be approached from multi-agent systems perspective, where we try to define the behavior of an agent (the virtual occupant). Different definitions of the concept of agent can be found in the literature. However, here we can use the Wooldridge's definition (Wooldridge 2009), "An agent is a computer system that is situated in some environment, and that is capable of autonomous action in this environment in order to meet its design objectives." Figure 3 shows the key components of an agent.

Thus, an agent is an entity that interacts with an environment by receiving information about it through its sensors and uses its effectors to act upon it. The concept of environment in our problem refers to the set of devices that make up the virtual intelligent environment we simulate, more formally, the environment is a space of movement, a set of available resources and a set of laws.

In agent-oriented paradigm, the problem is to find a formalism to simply define the behavior that the agent will have to perform. It should be noted that agents can be classified in different families. For example, Ferber defines four families according to the type of behavior and the agent type (Ferber 1997) (Table 1). The behavior is divided into two categories:

- The reflexive behavior in which the action results directly from the stimulus. There is no reasoning on the part of the agent, but rules associated with different stimuli. The overall behavior of the agent, therefore, depends on the stimuli of the environment.
- The teleonomic behavior in which the agent thinks and acts according to the stimuli of the environment, but also the objective it has to achieve.

Moreover, there are two types of agents:

- Reactive agents are those that generally have no memory, no reasoning and a partial vision of their environment (through perception). Reactive agents generally have a type of behavior that is: stimulus → perception → action.
- Cognitive agents are those that have a symbolic representation of their environment and can reason from these symbols.

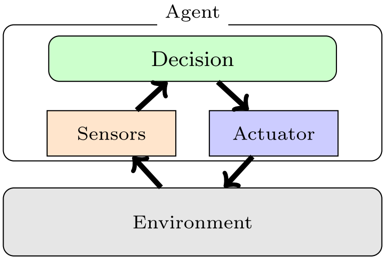


Fig. 3 Diagram showing the key components of an agent

Table 1 Summary of agent categories

	Reactive agents	Cognitive agents
Reflex behavior	Module agents	Tropic agents
Teleonomic behavior	Impulsive agents	Intentional agents

Module agents have a symbolic representation of their environment but a reflexive behavior because they have no objective. They are generally agents that provide services to other agents. Tropic agents have no objectives and only react to changes in the environment. Impulsive agents have a fixed objective and trigger behavior when the environment no longer allows the objective to be met. Intentional agents have objectives that they seek to accomplish. Intentional agents have to reason and plan to achieve these objectives.

In our case, the problem is to identify an approach that simply defines the objectives of intentional agents. There are several models for constructing an agent's deliberation process. The BDI model (for Belief, Desire, Intention) is a model based on Bratman's theory of human reasoning (Bratman 1987). In this model, the agent has beliefs that are the set of knowledge it has about its environment. It builds these beliefs from these perceptions which can be false or expired. Desires are the agent's objectives, i.e., the environmental or personal states that the agent seeks to achieve. Intentions are the actions that the agent will perform to satisfy desires.

In our context of simulating ADLs, the desires would be partly given by the scenario to be simulated. Beliefs would be information about the layout of the environment and the elements necessary to accomplish a task (for example, cooking involves taking utensils, etc.). The agent defines the intentions and they create the interaction with the virtual environment and data generation upon their realization.

Another model is ACT-R (Adaptive Control of Thought-Rational) by Anderson (2014). ACT-R is based on the theory that human actions are the result of atomic operations that emerge from perceptions and cognition. It is also based on the assumption that human knowledge is based on "declarative" and "procedural" representations, so in this model, the agent has at least two memory modules. The declarative memory contains facts and the procedural memory contains procedures, i.e., knowledge on "how to perform operations". The production rules, as well as the perceptions and content of declarative memory, allow the agent's objectives to evolve and interact with the environment. Objectives are usually present in an objective module. Mechanisms are used to filter and select the production rules to be applied according to the current objective. The production rule that is activated is the one with the highest utility value.

The LIDA (Learning Intelligent Distribution Agent) model is an evolution of Franklin et al.'s IDA model (Franklin et al. 1998) on which learning has been added (Franklin and Patterson Jr 2006). The IDA architecture is based on several memories (working, procedural, etc.), and the different cognitive cycles are linked uninterruptedly. A process called awareness is responsible for maintaining the sequential aspect of decisions and the choice of the order of rules to be applied in order to ensure consistent behavior. Dubois et al. have developed this model and proposed CTS

(Conscious Tutoring System) to add the emotional aspect to create training agents (Dubois et al. 2010).

The layered models are another family of approaches, which can be classified into two groups (Weiss 1999):

- horizontal layer models that have *n* layers connected directly to the agent's sensors and effectors and can act directly on the actions.
- vertical layer models that have *n* layers that are connected. Here, one layer deals with the input of perceptions and another layer deals with the processing of actions.

2.4 MCI and AD in the smart home context

Alzheimer's disease is a neurodegenerative disease characterized by a decline in cognitive function and memory loss. MCI represents a state of transition from normal aging to dementia; however, not all patients with MCI fall into AD (Angelucci et al. 2010). These pathologies have an impact on patient behavior and, in particular, on the performance of daily life activities (Van der Mussele et al. 2014; Dillon et al. 2013; Chiaravalloti and Goverover 2016). From an IE perspective, these differences should be reflected in action sequences with patterns different from those of healthy individuals. These differences are errors because they do not complement the activities to which they are related. One of the objectives of an IE is to detect its errors in order to apply a procedure to correct them. There are different types of errors (Chiaravalloti and Goverover 2016) that are summarized in Table 2. Our objective is, therefore, to be able to simulate sequences of actions where these errors will be found.

3 Related works

Our goal is to have an easy-to-learn and flexible computer model that can quickly define interaction scenarios between virtual occupants and a simulated IE. Therefore, to perform the literature review, the following keywords were used: "human activity simulation", "smart home simulation", "error simulation" and "intelligent environment simulation". We have kept the papers dating from after 2013 except for a paper from 2007 that addresses the problem of simulating errors of persons with AD in the field of smart homes. We took the date of 2013 because many 3D simulators with the objective to generate data, similarly to us, appeared around this date. Only those using the model-based approach were kept for the literature review.

3.1 Analysis criteria

The following analytical criteria were considered for our review:

Table 2 Table of error categories adapted from Chiaravalloti and Goverover (2016)

Category	Definition	Example
Omission	A step is not performed	Do not turn off the cooker; do not add sugar to coffee
Substitution	A step or object is replaced by another one	Put salt in the coffee, use the frypan instead of the pan
Sequence	Anticipation of a step; steps done in the wrong order	Put the jam on the bread before toasting it
Perseveration	A step is performed more than once during an excessive period of time	Add butter to a toast several times
Action-addition	Do an action that is not a step of the activity	Turn on the oven during the activity make coffee
Other	An object is used in the wrong way	Scissors are used as a knife

- What is the approach? This criterion consists of analyzing the approach used to simulate human activities.
- Does it generate errors? This criterion consists of analyzing whether the approach used allows errors to be introduced in the simulated activities. If the answer is yes, we study the type of errors simulated.
- Is the model independent of the virtual environment where the avatar evolves? This criterion consists of analyzing whether scenario modeling can be done independently of the configuration of the environment where the scenario will be executed.

3.2 Presentation

Kormanyos et al. propose a 2D simulator that uses a hierarchical model of activity to simulate human activities (Kormányos and Pataki 2013). In this model, an activity is broken down into a set of tasks divided into three levels. The highest level represents the highest level of abstraction such as eating or drinking water. These activities are associated with a priority function that assesses the priority of an activity according to different variables. For example, the priority of the "drinking water" activity is set according to the avatar's level of thirst. The higher level defines the sequence of tasks for the intermediate level, and the latter defines the sequence of tasks for the lower level. The lowest level includes tasks that have a direct effect on the environment, such as taking an object. The idea is that the priority of activities and, consequently, the actions would change according to the interactions with the environment. This would have an influence on the sequence of actions to be executed. However, this is not an error injection, it is simply variability in the sequences of actions that are generated. Any activity in this model is fully performed.

UbikSim is an intelligent environment simulator using the agent paradigm to simulate interactions between several avatars and the environment (Serrano and Botia 2013; Botía et al. 2014). In particular, it can be used to simulate crowd behavior in an intelligent environment. In this model, each avatar is an agent and executes a completely user-defined behavior. UbikSim uses the MASON Java library (Multiagent Simulator of Neighborhoods Luke et al. 2005) to implement the agents but, in absolute terms, the user is free to use the desired approach to model them. Thus, environmental independence depends on the chosen approach.

Liu et al. propose in Liu et al. (2015) an approach designed to emulate human activities for visualization and dataset generation purposes. In particular, the authors start from the observation that most of the existing activity models have been created in order to be applied to the recognition of activity and not to the simulation of human activity. The proposal is based on two key elements: a model of human activity and an algorithm interpreting this model in order to define the sequences of actions to be carried out. This model has the particularity of being able to generate activities with flexible action sequences. In fact, when the algorithm must instantiate an activity from its model, it selects from the model a set of actions for each step of the activity (respectively, the beginning, middle and end). For each action, it defines the duration and the number of occurrences according to the information defined in the model. In the model, actions are associated with the steps in which they occur, the number of occurrences and the minimum and maximum duration. The model also proposes a set of actions that can intervene at any stage. The algorithm builds a sequence according to these conditions and generates the sequence which is then executed by the avatar. This approach makes it possible to define scenarios of human activity that are independent of the environment.

The work of Liu et al. (2015) seems to complement Lee et al. (2015, 2014). Indeed, if Liu's proposal is particularly interested in generating sequences of plausible actions for a given scenario, Lee et al.'s proposal makes it possible to manage navigation between activities. The approach consists of defining a scenario as a context graph, in which the avatar navigates. A context can be defined as a container that contains: (i) the set of conditions that allow entering the context; (ii) the set of tasks that is performed when the avatar is in the context. The arrangement of sensors in an environment that affects their activation makes it difficult to make scenarios independent of the environment.

OpenSHS is a 3D simulator of intelligent environments (Alshammari et al. 2017) and (Alshammari et al. 2018) that uses Foundation (2016). First, it proposes an interactive approach where the user directly controls the avatar by using a keyboard and a mouse to interact with the environment. However, it also uses a sensor-based approach to simulate long-term activities. The idea is to use the interactive approach to record for each user-defined context (e.g., Monday morning activities, midday activity, etc.) a set of samples containing the possible activity sequences. The system then proposes a mechanism that can simulate a scenario for a predefined context by selecting from the samples the activities to be repeated. This approach does not propose an explicit model for introducing errors in interactions. In fact, the user can voluntarily make errors during interactions, but these errors are simply repeated during the repetition phase. Moreover, the approach is not totally independent of the environment because if the arrangement of the sensors changes, the recorded scenario is no longer consistent. Ho et al. propose the SESim 3D simulator which also uses the Unity engine. Unlike the previous proposal, the authors use behavior trees to define scenarios and simulate daily life activities (Ho et al. 2019). The common problem with all those previous proposals is that they do not provide an explicit model for injecting errors.

Serna et al. proposed a computational modeling of an individual's cognitive process when performing an ADL (Serna et al. 2007). The objective of this model was to simulate the impact of Alzheimer's disease when performing an ADL. This model uses Anderson et al.'s ACT-R architecture (Anderson 2014), which was introduced in Sect. 2.3. The approach can be summarized as an objective-oriented approach where the agent has a set of production rules. The agent chooses the rule. The production rule that is activated is the one with the highest utility value. This utility value is calculated by a function. The approach generates errors by introducing variables that can affect this utility value and that affect the triggering of the rules. This model makes it possible to define scenarios independently of the experimental environment.

Shima is an intelligent environment simulator presented in Francillette et al. (2017). It proposes to simulate the operation of sensors and actuators in a 3D environment in order to generate datasets. The simulator uses the Unity 3D game engine (Technologies 2016), and avatar behavior can be simulated using behavior trees. The principle is that the user defines these scenarios of interaction between the avatar and the environment using these behavior trees. During the simulation, the trees are interpreted to simulate human activity. This model makes it possible to define the interaction scenario and the intelligent environment completely independently. The user defines the actions to be carried out and the order of these actions. It is possible to use a set of actions already defined or to extend this set by adding new actions. In the first release, Shima did not offer an error simulation. The omission error injection has been added in an enhancement via omission operators to add in the behavior trees (Bouchard et al. 2018).

3.3 Discussion

Table 3 summarizes the different works. We can see from this table that there are several approaches to simulating human activities in a virtual intelligent environment. However, none of the reviewed solutions propose a model that integrates automatic error generation.

It can also be noted that almost all proposals use models of human activity that are independent of the configuration of the virtual IE. In fact, only OpenSHS uses a method based on sensors' activation. The arrangement of the sensors being linked to the virtual intelligent environment explains this dependence. Consequently, it is very difficult or impossible to define scenarios and apply them to different virtual environments configurations. With OpenSHS, samples would have to be redefined in the new environment each time.

Even if the proposed models do not have an explicit method to introduce errors in the simulated activities, it is possible to incorporate these errors in an ad hoc manner in some methods. For example, Liu et al.'s (2015) proposal can be used to define variants for each activity model where the errors could be inserted in the action sets of each step. However, this approach would be costly in time as it would require the definition of these variants for each patient profile and, in addition, the integration of the different omission and substitution scenarios.

This is also true for the behavior trees used by Shima. It would be possible to enrich the action set with actions that correspond to errors and build trees with them. However, this would give a fairly static scenario with errors that would always occur at the same time in the scenario.

The approach of Serna et al. seems to offer a good modeling of the cognitive process involved in the realization of ADLs (Serna et al. 2007) . The experiment presents only the realization of a cooking activity, but it is possible to use this model for other types of ADLs. The main problem in this approach resides in the definition of the production rules. Indeed, the actions contained in the latter include the evolution of the avatar's objective. For example, it may be difficult in some cases to quickly create variants of a scenario or to reuse these production rules as they are in other scenarios. Moreover, the approach focuses on the cognitive aspect of the realization of ADLs and not on the physical aspect such as the duration of actions which can

Table 3 Summary of propositions according to our criteria

Work	Approach	Errors	Independent
Serna et al. (2007)	ACT-R	Omission; substitution; sequence; perseveration; action-addition	Yes
Shima (Francillette et al. 2017)	Behavior tree based	Omission (added in Bouchard et al. 2018)	Yes
OpenSHS (Alshammari et al. 2017, 2018)	Sensors based	No	No
Kormányos and Pataki (2013)	Hierarchical model	No	Yes
UbikSim (Serrano and Botia 2013; Botía et al. 2014)	Multi-agent-based	No	NA
Liu et al. (2015)	Activity model	No	Yes
Ho et al. (2019)	Behavior tree based	No	Yes
Lee et al. (2014)	Context based	No	Yes
Our proposition	Behavior tree based	Omission; substitution; sequence; perseveration; action-addition	Yes

vary from a person to another. In addition to the objectives of flexibility and simplicity introduced in the previous sections, our approach integrates the physical dimension of the realization of ADLs. Our solution aims to help the rapid prototyping of experimental ADL scenarios for dataset generation and pre-experimental testing.

4 Activity model

4.1 Overview of the proposition

Our objective is to be able to easily define scenarios to be performed by an avatar in order to create interactions with an IE simulator to generate datasets. The generated scenarios are to be performed by healthy persons and persons with MCI or AD. An additional constraint is that the formal description must be simple and that a user without any particular knowledge of these diseases can simulate the desired behaviors.

The principle is to use a formal model of the activities that allows us to simulate the sequences of actions involved in carrying out the activities. In this model, it is possible to generate errors but also to detect and correct them automatically during the completion of activities. During simulation, scenarios are executed with probabilities of triggering errors. An automatic error checking mechanism is implemented to allow the virtual occupant to detect and correct errors by itself. Finally, the system uses a memory management system to simulate the avatar's amnesia. Figure 4 shows the interactions between these different components.

4.2 Behavior trees-based activity model

The ADL model we propose is based on the hierarchical job description model presented by Humphreys and Forde (1998). In this model, the actions that intervene in the course of an activity can be grouped into sub-activities that are parts of the overall activity. This hierarchical decomposition represents a definition of tasks as trees which have the advantage of being easily interpretable by humans.

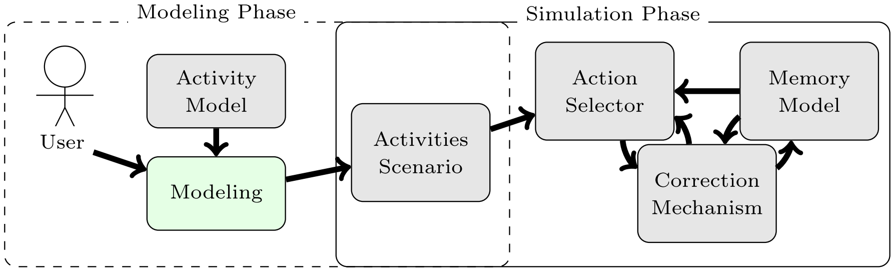


Fig. 4 Diagram showing the interactions between the main components of our approach during the modeling and simulation phases

The formal model of human activity is built on behavior trees (BTs). Behavior trees are a formalism that is regularly used in several fields, including video games, where they are used to define the behavior of non-player characters that are controlled by an AI Marcotte and Hamilton (2017); Marzinotto et al. (2014). In fact, behavior trees make it possible to define and organize under a tree structure the sequence of actions that avatars could then perform. Another similar concept can be found in the field of user interface design. Indeed, Concur Task Tree (CTT) uses tree concepts with intermediate nodes that are control nodes and leaves that represent tasks (in the context of CTTs) (Paternò et al. 1997; Paternò 2004).

Basically, a BT is an ordered tree. The fact that it is ordered means that the tree has the following properties:

- The arcs have an orientation, the starting node is called the parent, the finish node is called a child;
- It has only one parentless node called the root,
- The children of a node share a reading order (for example, from left to right). In the context of BTs, this means that children's order influences the effect of the tree.

A childless node is called a leaf. In BT formalism, a leaf is called an execution node and represents an action or condition, for example, to move or control the state of an entity. A parent is called a flow control node or a composite node. They are declined in many different subtypes, but the main purpose of this type of node is to define how the tree evolves according to the state of its children.

When a BT node is running, it is in one of the following states:

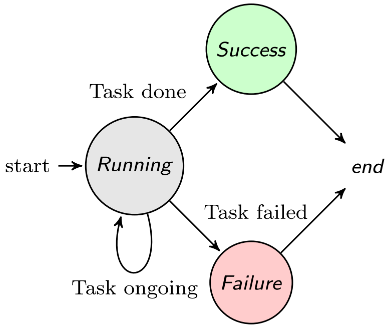
- *Running* means the task is not finished yet.
- *Success* means the task is finished with success.
- *Failure* means the task is finished with failure.

It is important to note that in this model, the failure state does not mean an error. It means that the task could not be accomplished and it is impossible to accomplish it in the future with the same initial prerequisites. For example, if the task consists of completing an action before a certain time, the failure state means that the time is over and the task is not finished. The success and failure states correspond to two end states of the node. Figure 5 shows the state diagram of a node.

A leaf is an action node that defines:

- The procedure for performing an atomic action (the animations to be played, the calculations for movement, etc.)
- Two sets of environmental and agent states for which:
    - The action is considered complete, this is the set S_i (with i as the node identifier);
    - The action is considered failed and can no longer be completed, this is the set F_i (with i as the node identifier); $S_i \cap F_i = \emptyset$

Fig. 5 Diagram of states of a node



- All the other states are part of the set R_i; $S_i \cap F_i = S_i \cap R_i = R_i \cap F_i = \emptyset$.

A parent defines its status based on the status of its children and its internal type. The pattern of a leaf node is given by algorithm 1.

Basically, the execution of a BT is based on discrete updates called ”Ticks” that are performed by the root. A tick is a single depth-first search of the tree recursively from the root. During the search, each node computes its state. If the node is a composite node, it defines how its children are explored. The children are usually aligned horizontally and the first node is the leftmost. Most of the time, the children are explored from left to right until a particular state defined by the nature of the parent is reached. However, some specific search behavior can be defined to change this order (other than from left to right). The process of updating the tree is given by the algorithm 2. The “get next node” function allows you to browse the tree to retrieve the next current node based on the operational semantics of the composite nodes and their children’s states.

The types of parents widely implemented are: *sequence*, *selector* and *decorators repeat* and *inverter*. We have also identified other types of parents that are present below. In addition, Table 4 presents the logic of calculating the status of each parent type:

- *Sequence* (symbol →) when activated, it sequentially activates its first child that is not in an end state as long as the child returns to success. If the node arrives at the end of the child list, it returns a success. If a node returns a failure, the parent returns failure without activating the remaining children. Algorithm 3 gives the functioning of the tick of this node.
- *Selector* (symbol ?) when activated, it activates sequentially its first child that is not in an end state until it has a child return a success. If the node arrives at the end of the child list, it returns a failure. If a node returns a success, the parent returns success without activating the remaining children. Algorithm 4 gives the functioning of the tick of this node.
- *Decorator* when activated, it transforms the end state (succeeded or failed) of its only child. As a decorator, it can implement:

Table 4 Summary of composite nodes and their operational logic

Composite	Symbol	Success	Failure	Running
Sequence	$\rightarrow$	All children S	1 child F	1 child Run
Selector	?	1 child S	All children F	1 child Run
Repeat	$R(\alpha)$	The child's number of $S > \alpha$	none	Child Run
Inverter	I	Child F	Child S	Child Run
At Least	$AtL(\alpha)$	Number of children $S > \alpha$	Number of children $F > \alpha$–Number of children S	Child Run
Random sequence	$\rightsquigarrow$	All children S	1 child F	1 child Run
Random selector	$\sim ?$	1 child S	All children F	1 child Run
Random At Least	$\sim AtL(\alpha)$	Number of children $S > \alpha$	Number of children $F > \alpha -$ Number of children$S$	Child Run
Omission	$\Theta(e)$	Node is activate	None	None
Substitution	$\Sigma(e, \sigma)$	Node is activate and new child S	Node is activate and new child F	Node is activate and new child Run
Parallel	$\Vert\ (\alpha)$	All children S	1 child F	1 child Run
Interrupted	$Itd(ev, \sigma)$	Node is activate and new child S	Node is activate and new child F	Node is activate and new child Run

In this table, we adopt the following notation: S = state of success; F = state of failure; Run = state of running

  - *Repeat* It repeats the processing of the child. It can repeat until a number of succeeded states are reached. Algorithm 5 gives the functioning of the tick of this node.
  - *Inverter* It inverts the ending state of its child (failure is transformed to success for example). Algorithm 6 gives the functioning of the tick of this node.

- *At Least* when activated, it sequentially activates its first child that is not in an end state until it has a number of children in the success state greater than a predefined number. If the preset number is reached, it returns successfully without activating the remaining children. If the number of nodes in the failed state is greater than the number of children minus the desired number of nodes in the successful state, it returns a failure. Algorithm 7 gives the functioning of the tick of this node.
- *Random* In a random composite, the activation of children is no longer necessarily done from left to right but in random order.
  - *Random sequence* (symbol $\rightsquigarrow$) when activated, it sequentially and randomly activates a child that is not in an end state as long as the child returns to success. If the node arrives at the end of the child list, it returns a success. If a node returns a failure, the parent returns failure without activating the remaining children. Algorithm 8 gives the functioning of the tick of this node.
  - *Random selector* (symbol $\sim ?$) when activated, it sequentially and randomly activates a child that is not in an end state until it has a child return a success. If the node arrives at the end of the child list, it returns a failure. If a node returns a success, the parent returns success without activating the remaining children. Algorithm 9 gives the functioning of the tick of this node.
  - *Random at least* when activated, it sequentially and randomly activates a child that is not in an end state until it has a number of children in the success state greater than a predefined number. If the preset number is reached, it returns successfully without activating the remaining children. If the number of nodes in the failed state is greater than the number of children minus the desired number of nodes in the successful state, it returns a failure. Algorithm 10 gives the functioning of the tick of this node.

```
Result: Update the action node
//this method contains the implementation of the action to execute
$state_{current}$ ⇐ agent.update();
if $state_{current} \in S_i$ then
    //$S_i$ is the set of success states
    return $SUCCESS$;
else if $state_{current} \in F_i$ then
    //$F_i$ is the set of failure states
    return $FAILURE$;
else
    return $RUNNING$;
end
```

Algorithm 1: Action.Tick()

```
Data: $root$, root of the behavior tree
Result: The behavior tree $root$ is updated
$state_{current}$ ⇐ $root$.Tick();
if $state_{current} \neq RUNNING$ then
    stop;
end
```

Algorithm 2: Tick()

```
Result: Sequence node is updated
// $children$ is the list of node's children
$state_{current}$ ⇐ $children$.get(0).tick();
if $state_{current} = SUCCESS$ then
    $children$.remove(0);
    if $children$.isEmpty() then
        return $SUCCESS$;
    end
else if $state_{current} = FAILURE$ then
    return $FAILURE$;
else
    return $RUNNING$;
end
```

Algorithm 3: Sequence.Tick()

```
Result: Selector node is updated
// children is the list of node’s children
$state_{current}$ ⇐ children.get(0).tick();
if $state_{current}$ = FAILURE then
    children.remove(0);
    if children.isEmpty() then
        return FAILURE;
    end
else if $state_{current}$ = SUCCESS then
    return SUCCESS;
else
    return RUNNING;
end
```

Algorithm 4: Selector.Tick()

```
Result: Repeat node is updated
// count is an attribute given when this node is created
// children is the list of node’s children
$state_{current}$ ⇐ children.get(0).tick();
if $state_{current}$ = SUCCESS then
    count–;
    if count = 0 then
        return SUCCESS;
    end
else if $state_{current}$ = FAILURE then
    children.get(0).Reset();
    // "Reset" method resets node to initial parameters given when node was
      created
else
    return RUNNING;
end
```

Algorithm 5: Repeat.Tick()

```
Result: Inverter node is updated
// children is the list of node’s children
$state_{current}$ ⇐ children.get(0).tick();
if $state_{current}$ = FAILURE then
    return SUCCESS;
else if $state_{current}$ = SUCCESS then
    return FAILURE;
else
    return RUNNING;
end
```

Algorithm 6: Inverter.Tick()

```
Result: AtLeast node is updated
// count is an attribute given when this node is created
// children is the list of node's children
$state_{current}$ ⇐ children.get(0).tick();
if $state_{current}$ = SUCCESS then
    count–;
    if count = 0 then
        return SUCCESS;
    end
else if $state_{current}$ = FAILURE then
    children.remove(0);
    if children.isEmpty() then
        return FAILURE;
    end
else
    return RUNNING;
end
```

Algorithm 7: AtLeast.Tick()

```
Result: Random Sequence node is updated
// children is the list of node's children
$state_{current}$ ⇐ children.get(0).tick();
if $state_{current}$ = SUCCESS then
    children.remove(0);
    // "shuffle" method changes the position of the remaining nodes in the list
    children.shuffle();
    if children.isEmpty() then
        return SUCCESS;
    end
else if $state_{current}$ = FAILURE then
    return FAILURE;
else
    return RUNNING;
end
```

Algorithm 8: RandomSequence.Tick()

```
Result: Random Selector node is updated
// children is the list of node's children
$state_{current} \Leftarrow$ children.get(0).tick();
if $state_{current} = FAILURE$ then
    children.remove(0);
    // "shuffle" method changes the position of the remaining nodes in the list
     children.shuffle();
    if children.isEmpty() then
        return FAILURE;
    end
else if $state_{current} = SUCCESS$ then
    return SUCCESS;
else
    return RUNNING;
end
```

Algorithm 9: RandomSelector.Tick()

```
Result: Random AtLeast node is updated
// count is an attribute given when this node is created
// children is the list of node's children
$state_{current} \Leftarrow$ children.get(0).tick();
if $state_{current} = SUCCESS$ then
    count–;
    if count = 0 then
        return SUCCESS;
    end
else if $state_{current} = FAILURE$ then
    children.remove(0);
    // "shuffle" method changes the position of the remaining nodes in the list
     children.shuffle();
    if children.isEmpty() then
        return FAILURE;
    end
else
    return RUNNING;
end
```

Algorithm 10: RandomAtLeast.Tick()

Composite nodes can be used to construct complex scenarios, such as making a meal, cleaning a room, etc., from a set of simpler elements. In our context, the leaves represent the actions that an occupant can perform, such as picking up an object, pouring something into a container, activating a switch, etc. With this model, we can define scenarios by simply using composite nodes to organize the actions of the virtual avatar. In addition, it is possible and simple to add new actions to the initial set in order to extend the possibilities. It should also be noted that the leaves can be customized, allowing one to create variations in the same scenario.

Let's take an example to implement this model. In the context of activities of daily life, the following actions can be identified. The word between chevrons indicates the parameter of the action:

- **Take <Item>** if the item exists in the place and is accessible, the avatar goes to the item and takes it then the node returns a success, otherwise the node returns a failure.
- **Put <Item> on < Container>** if the item is in the avatar's hand and the target container exists in the place and is accessible, the avatar goes to the target container and places the item on it and then the node returns a success, otherwise it returns a failure.
- **Turn On/Off <Device>** if the target device exists in the location and is accessible, the avatar goes to the target device and turns it on (or off) then the node returns a success, otherwise the node returns a failure.
- **Pour <Item> into <Container>** if the item is accessible and the container is in the avatar's hand or if the item is in the avatar's hand and the container is accessible, the avatar goes to the accessible and transverse element and then the node returns a success, otherwise the node returns a failure.

There are also the following action nodes that are implicit, i.e., in our implementation, it is not necessary to add them explicitly in the tree because they are induced by other nodes if necessary. Figure 6 shows the sequence of processes that take place to carry out an action. This makes it possible to reduce the initial size of the tree to make it easier to read, but also to generate behaviors that are similar to those of people with MCI or AD.

- **Go to <Room>** if the target part is accessible, the avatar goes there and then the node returns a success, otherwise the node returns a failure.
- **Open/Close <Item>** if the element is accessible, the avatar goes ahead and opens (or closes it) then the node returns a success, otherwise it returns a failure.

From these elements, scenarios such as "making coffee", for instance, can be built. In this scenario, the avatar must go to the place where the coffee is located (this is an implicit action "Go To"), take the coffee, go to the place where the coffee machine is located (this is another implicit "Go To"), put coffee in the coffee

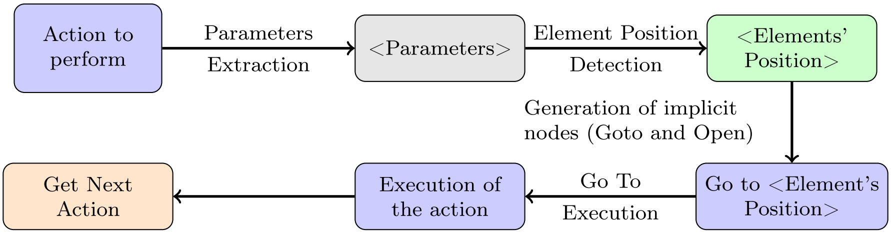


Fig. 6 Diagram of the process of processing action nodes

machine, go to the place where the cup is located (an implicit action "Go To"), put coffee in the coffee machine, go to the place where the cup is located (a implicit "Go To"). The use of the composite "sequence" implies that these actions must necessarily take place in this order.

In the previous scenario, the avatar tries to make coffee and, if one of the actions cannot be completed (for example, there is no coffee), the avatar stops the activity "making coffee" without having had coffee. The complexity of this scenario can be increased so that the avatar tries to make tea if it cannot make coffee. To do this, the composite "selector" is used to define another action sequence to be carried out in case of failure of the first one. This gives the BT of Fig. 7.

The complexity of the activity can be further increased by asking the avatar to prepare a dish in addition to the preparation of coffee or tea. For this scenario, the avatar is free to choose if it prepares the dish before or after the preparation of the drink. In order to model this, the random sequence composite is used to construct the tree of Fig. 8.

4.3 Interlaced activities modeling

In everyday life, individuals can perform several activities in parallel, for example, watching television, making coffee and cooking. From an IE perspective, this is characterized by subsequences of actions belonging to several activities that follow each other. These are called interlaced activities. In order to allow this type of behavior, the random sequence operator could be used to divide the activities in order to control the passage from one activity to another. However, instead, the parallel operator can be inserted (symbol || (α)) to allow the switch from one activity to another. This operator improves the reusability of the trees by recovering one or more trees already built and

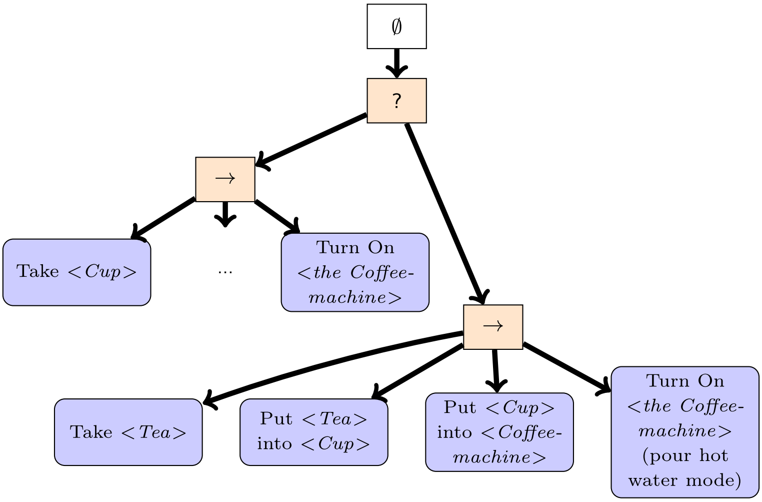

∅
?
→
Take <Cup>
…
Turn On <the Coffee-machine>
→
Take <Tea>
Put <Tea> into <Cup>
Put <Cup> into <Coffee-machine>
Turn On <the Coffee-machine> (pour hot water mode)

Fig. 7 BT to make coffee or tea. Suspension points are the actions *Put <Cup> into <Coffee-machine>*, *Take <Coffee>* and *Put <Coffee> into <Coffee-machine>*

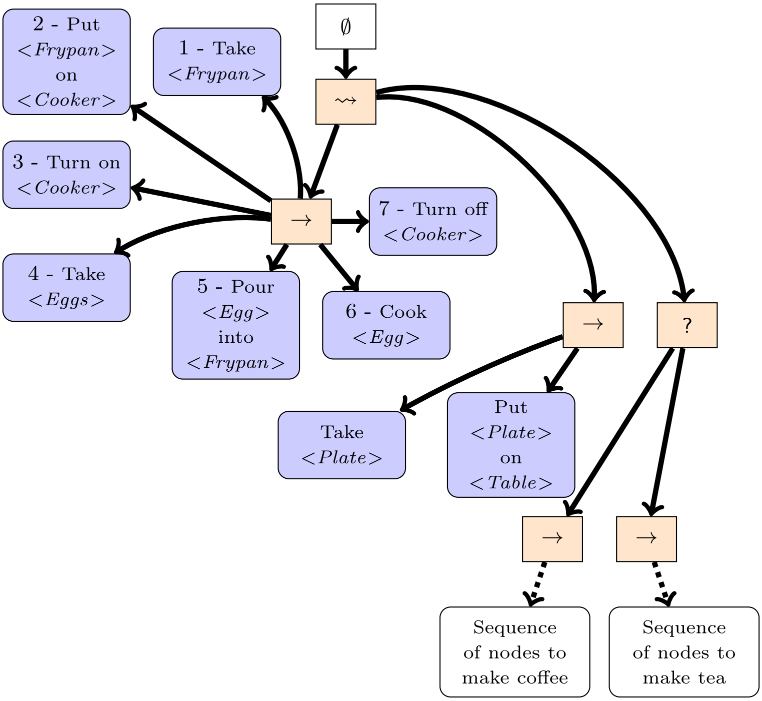


Fig. 8 BT to make a dinner and a cup of coffee or tea. Here, some nodes are not ordered from left to right for reasons of available space. We placed a number in front of the name of these nodes to indicate its position on the list of children

adding the execution in parallel. To do this, the functioning of the tick of the behavior tree is modified. The parallel operator takes several sub-trees as sons. When one of these descendants ends, it does not directly apply the operational logic of the root of the sub-tree having the node that ended. For example, if it is a sequence, it does not directly activate the brother of the node that ended. Instead, the parallel operator randomly activates one of his not ended children.

To be more specific, the avatar performs only one action at the time, but this operation allows a person to build a scenario to integrate actions from other activities during an activity. This allows recreating the interlaced activity. The operational semantics of the parallel operator consists in finishing in success when all the children finish in success. In the opposite case, it ends in failure when all children are in failure.

```
Result: Parallel node is updated
// children is the list of node's children
$state_{current}$ ⇐ children.get(0).tick();
if $state_{current}$ = SUCCESS then
    children.remove(0);
    if children.isEmpty() then
        return SUCCESS;
    end
else if $state_{current}$ = FAILURE then
    return FAILURE;
if children.hasGrandSonChanged() then
    // "hasGrandSonChanged" method indicates whether the next node to execute
      has been changed // "shuffle" method changes the position of the remaining
      nodes in the list children.shuffle();
end
```

Algorithm 11: Parallel.Tick()

4.4 Interrupted activity modeling

The other source of interlaced activities is from activities that are interrupted by events that trigger a new activity. For example, one can imagine interrupting his/ her activity to answer the phone or open the front door after the bell rings. To model these behaviors, we propose the "Interrupt" operator. This operator takes two elements as parameters, the first one is the event that triggers the interruption. This operator waits for this event to occur during the simulation. The second element is the tree that must be executed when the event occurs. When the event is detected, the current node is "paused" and the new tree is executed. The paused node is resumed when the substitution tree is finished (successful or failed).

5 Error injection approaches

Up to this point, the previous model makes possible the reproduction of scenarios of ADLs without any error (the behavior tree is assumed as well constructed and the environment allows the avatar to perform the scenario because all the elements are present). Our objective is to be able to introduce errors that persons with MCI or AD could realistically do while performing these scenarios.

5.1 Approach I: Error probabilities per individual action

5.1.1 Error modeling

Section 2.4 describes 6 types of errors that can be performed by the target populations: omission, substitution, sequence, perseverance, addition and others. To allow these errors to appear in the scenarios, two operators are added to the model. These two operators allow one to configure the appearance of errors. To emulate these characteristics, two composites are added to the set presented above: ”Omission” and ”Substitution”. These composites all take a particular parameter compared to other composites which is the activation probability that defines the probability of error occurrence. The logic of these elements is as follows:

- *Omission* (symbol $\Theta(e)$, where e is the probability of error): when it triggers, it does not activate its child and returns a success.
- *Substitution* (symbol $\Sigma(e, \sigma)$, where e is the probability of error and σ is the set of actions and sequence of actions that can replace the original action): when it triggers, it does not activate its child but replaces it by a node taken at random in the sigma set, then it activates the substitution node.

Figures 9 and 10 present behavior trees with these two composites to illustrate their behaviors. Figure 9 shows a scenario of washing laundry using a washing machine. The avatar must take the laundry, put it in the machine, take the detergent, pour it into the machine and activate it. In this tree, there are three compounds of omissions, one for the party responsible for the laundry, the other for the party responsible for detergent and the last one for starting up the machine. By placing the composite “omission” as the parent of these parts, it means that the avatar can forget these actions when making the scenario. Figure 10 shows the same scenario but with a possible substitution error where the avatar could be washing the dishes with the dishwasher instead of washing clothes.

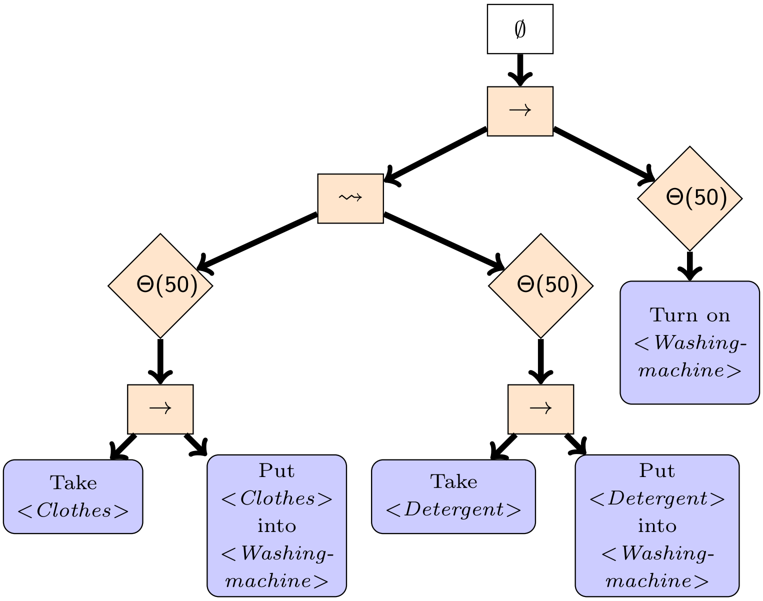

∅
→
⇝
Θ(50)
Θ(50)
Θ(50)
→
→
Turn on <Washing-machine>
Take <Clothes>
Put <Clothes> into <Washing-machine>
Take <Detergent>
Put <Detergent> into <Washing-machine>

Fig. 9 BT for washing laundry with the possibility of omission. The parameter numbers indicate a percentage

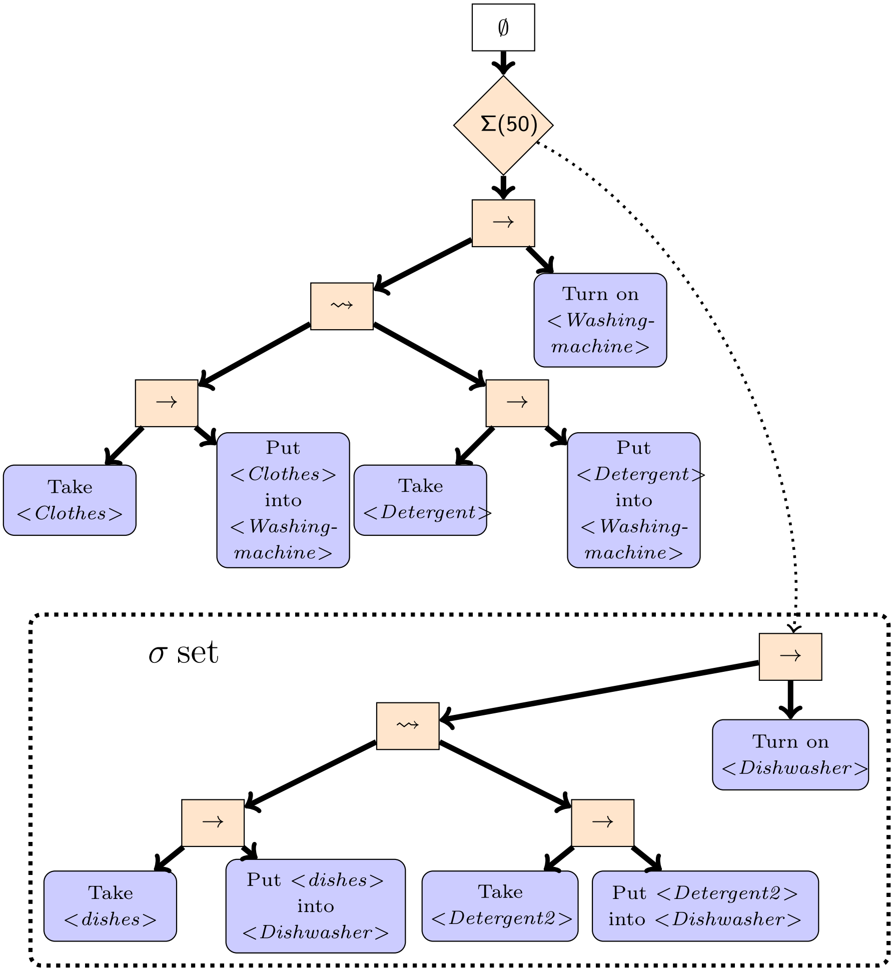

∅
Σ(50)
→
⇝
Turn on <Washing-machine>
→
→
Take <Clothes>
Put <Clothes> into <Washing-machine>
Take <Detergent>
Put <Detergent> into <Washing-machine>
σ set
→
⇝
Turn on <Dishwasher>
→
→
Take <dishes>
Put <dishes> into <Dishwasher>
Take <Detergent2>
Put <Detergent2> into <Dishwasher>

Fig. 10 BT for washing laundry with the possibility of substitution. The parameter numbers indicate a percentage. The σ set contains only one element, the dishwasher use scenario. The children of the composite substitution that are connected by dots represent the σ set

5.1.2 Memory model

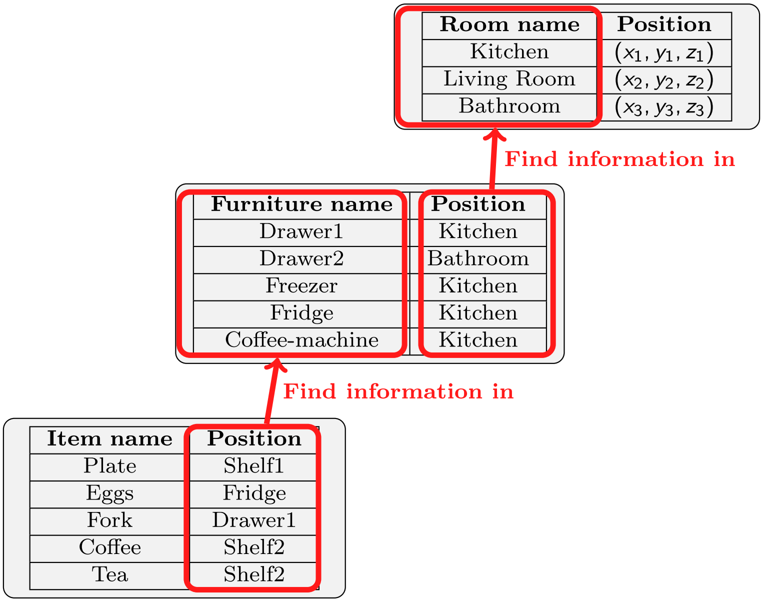
The addition of omissions and substitution composites make it possible to introduce these types of error in the reproduction of daily life activity. However, the model is not complete because it does not make it possible to emulate the behaviors associated with memorizing the location of objects or rooms in a place. This is due to the fact that the avatar can have a perfect knowledge of the locations of the objects and thus get there directly. We can observe, not only in patients but also in healthy individuals, a behavior that consists in looking for the location of objects necessary to accomplish a task. This is especially noticeable if the person is in an unfamiliar environment (Giovannetti et al. 2008). For example, the person discovers the environment by visiting the rooms, opening cupboards or drawers before or during the scenario in order to construct a mental map of the environment.

It is possible to model a BT that would emulate this search and discovery behavior of the environment and then integrate it into the activity scenarios of ADLs. However, this approach has the disadvantage of complicating the BT of the scenario. Moreover, the user chooses the moment when it would like to trigger this search behavior, so it removes the spontaneity during the simulation phase.

Our approach consists in modeling a mental map of the environment and making the avatar pass through this map in order to retrieve the position of the elements of the environment. If the avatar does not find this information in his mental map, it triggers a behavior that consists in visiting the environment until it gets the information and stores it in his mental map.

In addition, a customizable mechanism that can act on this mental map and alter it in order to simulate the impact of AD on the memory of individuals is introduced in the model. This mechanism deletes or falsifies information. This results in forcing the avatar to perform its search behavior. Figure 11 summarizes this approach.

The figure shows that there are three levels of abstraction for the memorization of the environment. The first level is linked to the layout of the rooms in the environment. It allows the avatar, if it has memorized the locations, to go directly to the desired room if necessary. If it does not have the information, it will go through the unknown rooms until it finds the one it wants. The second level represents the arrangement of furniture, appliances, switches in the environment; the avatar associates each piece of furniture to the room where it is located. Thus, when an action



Room name	Position
Kitchen	(x_1, y_1, z_1)
Living Room	(x_2, y_2, z_2)
Bathroom	(x_3, y_3, z_3)

Furniture name	Position
Drawer1	Kitchen
Drawer2	Bathroom
Freezer	Kitchen
Fridge	Kitchen
Coffee-machine	Kitchen

Item name	Position
Plate	Shelf1
Eggs	Fridge
Fork	Drawer1
Coffee	Shelf2
Tea	Shelf2

Fig. 11 Diagram of the different levels of our mental map

requires the use of a device or a piece of furniture if the piece of furniture is not visible, the avatar consults his mental map to locate which room to go to. If the avatar does not have the room layout or the position of the furniture in memory, it scrolls through the rooms until it finds the desired furniture or device. The last level is the level of small objects that the avatar can grab, e.g., food, dishes, detergent, etc. This type of element is usually placed in (or on) furniture. Thus, if the object is not visible to the avatar, our abstraction forces it to have the position of the furniture and the arrangement of the rooms in memory so that it can go directly to them. If it does not have this information, it must check the furniture to be able to use it. Technically, this mental map is a simple database where each level represents a table.

This mental map of the environment is updated automatically and in real-time as the avatar travels through the environment. When the mind map is complete, the avatar has perfect knowledge of its environment, this is equivalent to allowing the avatar to know the position of any object directly.

However, we do know that individuals with cognitive impairments can have memory problems, particularly those with AD. This disorder is characterized by memory loss or alteration of stored information (McKhann et al. 2011). To emulate this characteristic, we introduce a mechanism that aims to degrade memory of the avatar. The main idea of this mechanism is that at each time interval defined by the user, there is a probability of modifying (deleting or altering) random information stored in memory. The longer the interval of time and the higher the probability, the less knowledge the avatar has about the environment. Therefore, it is potentially necessary to browse the environment until it finds the desired objects for any actions to be completed. The memory degradation process can also make the avatar forget the action for which it is searching for the object. The principle is that from the moment the avatar starts searching for the object, the task is placed in suspense in the avatar memory and at each predefined time interval, the process has a probability of removing the task from memory. Thus, when the avatar finds the object, it may have forgotten the task it had to perform with that same object. The memory degradation process can also cause the avatar to forget the object it is looking for and therefore make it stop searching.

This process can also affect the probability of error of the remaining composites in the behavior tree. The idea is to simulate the fact that the risk of forgetting tasks can increase over time. Similarly, at each interval of time, it has a probability to increase the probability of triggering the composites “Omission” and “Substitution”. Figure 12 shows the interactions between the different elements responsible for avatar behavior.

5.1.3 Correction mechanism

All the elements introduced previously are used to generate errors in the simulated activities. However, this can create blocks in the activities. For example, the avatar forgets the task and the object it is looking for, or during a cooking activity, the avatar starts a task where it has to drain the water of the pasta that it cooked out of it and has omitted to put the pasta in the water. The main idea is that a person can make an error at a time t and realize that if at a time $t + n$ that error prevents

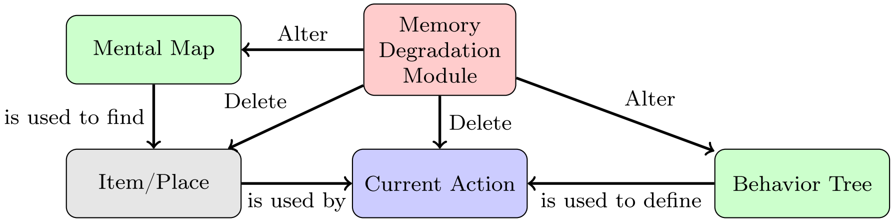


Fig. 12 Diagram of interactions between the memory degradation module and other components of our approach

continuing an activity. The purpose of this mechanism is, therefore, to allow an activity to continue.

Our approach incorporates an error resolution mechanism to address this type of problem. This module is triggered in two cases: (1) forgetting an action because of the memory degradation mechanism; (2) blocking when an action starts because the required pre-requisites are not met.

In the case of forgetting an action, our approach offers two options: (1) perform a default action, for example, the avatar stops the scenario or asks for help; (2) perform a tick on the behavior tree to determine the next action. It is also possible to carry out two both possibilities: perform a default action and then perform a tick on the current behavior tree or on a new behavior tree. In particular, this approach was used in the experimentation presented in Sect. 6.

In the case of an impasse due to unsatisfied prerequisite (due to an action not carried out correctly, for example, the pasta has not been poured into the water and the current action consists of draining the water from the pasta), the avatar is placed in a state where it tries to find the missing action(s). In this state, the approach presents two options. The first is the same as the previous case, it consists in making the avatar perform an action by default. In the second possibility, the avatar goes back to the parent node of the blocking action (the error composites “omission” and “substitution” are ignored, the selected parent node cannot be one of these two composites) and try to restart the activity. During this new iteration, the error operators can apply again, in order to avoid a possible infinite loop, it is possible to set a threshold where the avatar takes the other possibility.

5.2 Approach II: Error intervals

Our second approach is based on the idea that error injection should not be triggered only according to the probability given by the error nodes. The probability of triggering an error should also depend on the number of errors already injected in the sequence, the number of errors that can still be injected and the number of errors desired by the scenario. The purpose is to increase the probability of triggering errors if a minimum number of errors has not been injected. In the opposite case, it may be desirable to reduce the probability of triggering errors to zero if the maximum number of errors has been reached. Thus, the error injection system receives an interval $[a, b]$ as a parameter and its objective is to inject a number of errors within this interval.

This approach depends on the same foundation as the approach I described above; behavior trees composed of nodes with an error probability. For this second approach, the error injection system is modified by adding a process in charge of injection control. At each "Tick()", this process calculates a probability of triggering an error. The formulas used to calculate this probability are the following.

$$P = b/\mathrm{err}_{\mathrm{left}} \tag{1}$$

- b: is the upper bound of the interval.
- $\mathrm{err}_{\mathrm{left}}$: is the number of error nodes remaining in the tree.

$$P = \mathrm{err}/\mathrm{err}_{\mathrm{left}} \tag{2}$$

- a: is the upper bound of the interval.
- $err = b -$ number of errors injected.

When the system intends to inject an error, it searches the tree to select from the remaining error nodes one node to be activated. In this approach, the probabilities assigned to the error nodes are interpreted as priorities associated with the errors. The system tries to activate the node with the highest priority. Before activating the node, it performs a preliminary check to see if the number of errors that will be injected does not cause the maximum limit of the interval to be exceeded. There are three possible cases:

- The number of errors injected will not exceed the upper bound: the system triggers the activation of the error for the node.
- The number of errors injected will exceed the upper limit. The system searches for and finds another error node to be activated that allows it to stay in the interval. This new node is marked and will be activated when the tree reaches this point.
- The number of errors injected will exceed the upper limit. The system searches for and does not find another error node to activate that allows it to stay in the interval. The system is allowed to make a partial error activation that depends on the type of error node.
    - In the case of an omission, the system makes only an omission for the n last nodes, where n is the number of errors remaining.
    - In the case of a substitution, the first n elements are performed.

Algorithm 11 presents how this approach works.

Another modification to the first approach resides in the correction mechanism. It now performs checks before triggering a correction when the avatar is stopped for the execution of a task because the prerequisite actions have not been performed. The purpose of these checks is to ensure that the correction keeps the avatar within the desired error range. Thus, the system checks:

- If after correction there are still enough error nodes in the tree to remain above the minimum error threshold. In case yes, the correction mechanism is triggered.
- In case the non-correction of the error leads to other errors that cause the upper bound of the interval to be exceeded, the system triggers the correction mechanism. If there are not enough nodes left after the correction. The system triggers a partial correction.

```
Data: root, root of the behavior tree;
[errMin,errMax], error interval
Result: Errors can be inserted in the action sequence generated by the tree root
while ¬root.isEmpty() do
    root.tick();
    //function that returns the list of remaining error nodes;
    errNodeLeft ⇐ getRemainingErrorNodes();
    prob ⇐ computeProbability([errMin,errMax], errNodeLeft);
    if rand(0,1) < prob then
        nextErrNode ⇐ root.getNextErrorNodeToTrigger();
        if errMax < nextErrNode.lenght() then
            newErrNode ⇐ root.getNextErrorNodeToTriggerThatHasMaxLenghtUnder(errMax - nextErrNode.lenght());
            if newErrNode.isEmpty() then
                //this function takes as argument the number of nodes to activate;
                nextErrNode.partialErrorTriggering(errMax - nextErrNode.lenght());
            else
                newErrNode.triggerError();
            end
        else
            nextErrNode.triggerError();
        end
    end
end
```

Algorithm 12: Algorithm of the error injection approach based on an interval

6 Experiments

6.1 Experiment 1: Evaluation of generation of expected behaviors with approach 1 without human experts

The approach allows the creation of scenarios that generate errors in the simulated ADLs. In order to evaluate our approach and its ability to simulate the behavior of a person with AD or MCI, an experiment was conducted. The main objective of this experiment was to verify if a person without expertize on the clinical diagnosis of this type of pathology could, using our model, generate behaviors that belong to the desired categories. In this experiment, the goal was not to evaluate the efficiency or accuracy of the system.

6.1.1 Protocol

We have defined the scenario of ADL presented by the tree in Fig. 13. In this scenario, the avatar has to prepare a dish, place cutlery on the table, wash the used utensils and store these utensils as well as a bottle placed in the apartment. We performed twelve simulations with this scenario by changing the values of the error composites, the memory degradation process and the impasse resolution process. Videos of these simulations were recorded, and three cognitive disorder specialists were asked to classify the avatar of each video in one of the following categories: healthy subject, MCI, AD. The videos were produced by the same person who is not a specialist in this type of behavior. The videos were subjectively classified by this person according to the following principle:

- A healthy person does not make mistakes or ask for help,
- A person with MCI makes a few mistakes, misunderstandings and requests for help,
- A person with AD makes many mistakes, misunderstandings and requests for help.

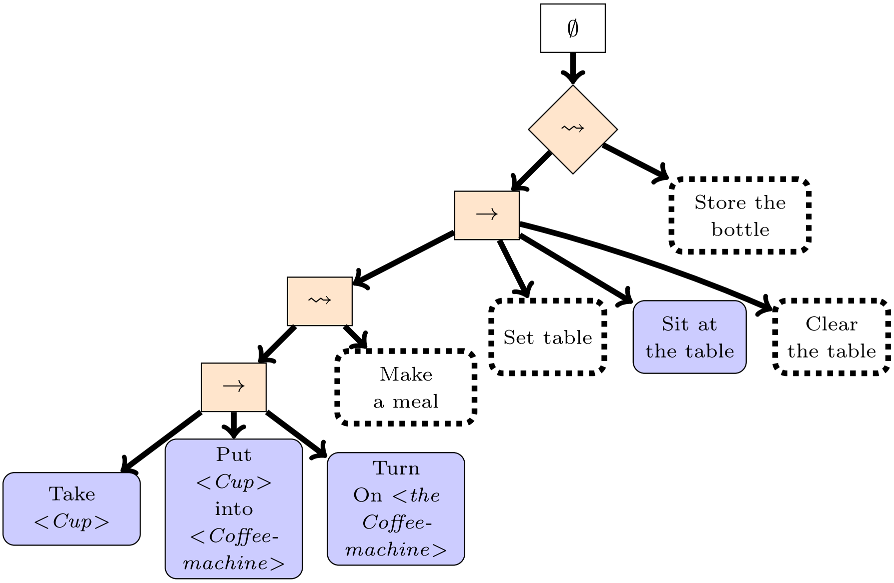


Fig. 13 BT for the experiment. The nodes between dashed lines are detailed in Figs. 14, 15, 16 and 17

The choice of parameters was made arbitrarily. The idea was also to check if it is possible to generate the same behavior with different configurations. Our simulations were implemented on our simulator available at https://github.com/Iannyck/shima. This simulator is implemented on Unity3D.

We simulate a scenario where the person is at home. Therefore, at the beginning of the simulation, the avatar had a complete knowledge of the layout of the apartment and the location of all the cutlery and ingredients (the goal is to play a scenario where the occupant is at home). When the avatar was blocked, the default action was "asking for help". This action was symbolized on the videos by the fact that the avatar made a wave of the hand. When it asked for help, the system would provide him with a sub-tree of the remaining actions. The remaining actions were those that were not in an end state.

The persons who had to perform the diagnosis were specialists in studying this type of behavior. They were informed that the gesture of the hand meant a request for help and that the avatar started with a perfect knowledge of the environment. They were informed about all the actions the avatar had to do but they were not aware of the configuration parameters used for the simulation. They were also informed that the actions did not have to be carried out in the order indicated on

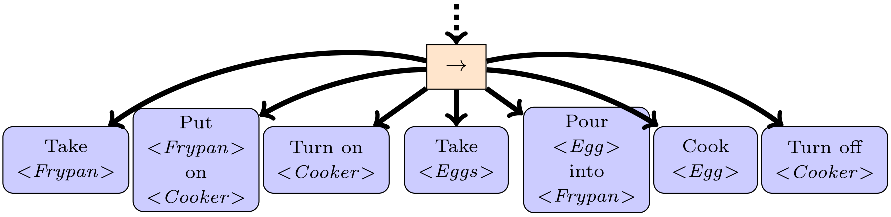


Fig. 14 Sub-tree "Make a meal" of the BT of the experiment presented in Fig. 13

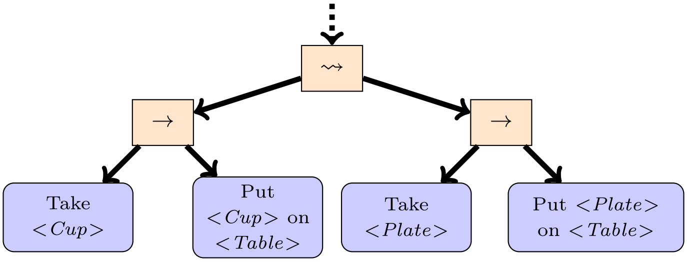
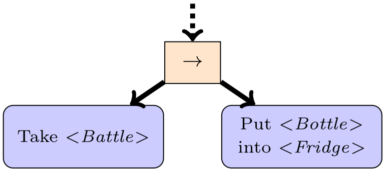


Fig. 15 Sub-tree "Set table" of the BT of the experiment presented in Fig. 13

Fig. 16 Sub-tree "Store the bottle" of the BT of the experiment presented in Fig. 13



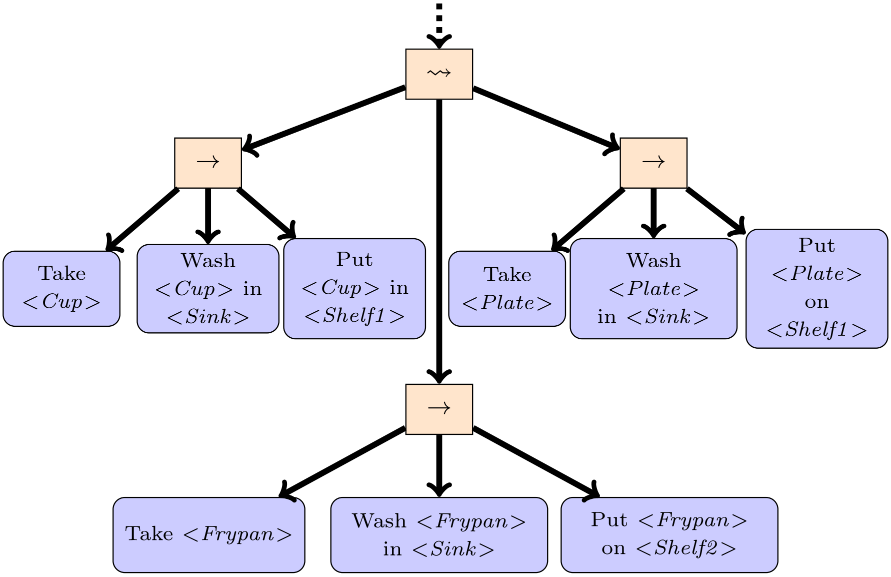


Fig. 17 Sub-tree "Clear the table" of the BT of the experiment presented in Fig. 13

the instruction sheet and that the fact that the avatar went to wash the dishes in the bathroom was not a mistake. A composite omission is placed on each arc of the tree in Fig. 13, with parameters changed for each simulation. It should be noted that our action nodes also takes as parameter the speed of execution of the actions. These settings have also been changed between videos. The number of videos not selected by the video creator is not counted, as the goal is to generate at least three videos of each class and to note for each video the type of class he/she thinks he/she has generated. Some of the simulations were repeated several times until a video was obtained that was judged to correspond to a desired behavior class (for example, to obtain video behavior that is similar to the behavior of a person with AD). Thus, the video creator did not retain videos that he/she felt did not visually match the desired class behavior. The fact that videos had to be eliminated comes from the randomness of the error triggering. A solution for limiting this is presented at the end of the discussion sub-section. Table 5 shows the parameters for each scenario. Figure 18 shows screenshots of these videos. The videos lasted between 3 minutes 18 seconds for the shortest simulation and 12 minutes 8 seconds for the longest.

6.1.2 Result

Table 6 shows the errors made by the avatars of each scenario.

Table 7 shows the diagnoses for each scenario.

On all the videos recorded, 5 out of 12 (videos: 1, 2, 3, 9 and 11) obtained the same diagnosis by the three specialists and this diagnosis was the one targeted when simulating behaviors. For the videos where at least two specialists selected the correct diagnosis, there are 10 out of 12 videos (videos: 1, 2, 3, 4, 5, 7, 8, 9,

Table 5 Sample of parameters used for simulation

Video identifier	Parameters	Desired class

1	Error = 0%
	Speed = [1–2];
	Memory degradation off
		Healthy

2	Error = 40%
	Speed = [2–4];
	Memory degradation = 50%, [5–10]
		Alzheimer

3	Error = 20%
	Speed = [2–4];
	Memory degradation = 20%, [10–20]
		MCI

4	Error = 0%
	Speed = [3–5];
	Memory degradation off
		Healthy

5	Error = 0%
	Speed = [2–3];
	Memory degradation off
		Healthy

6	Error = 30%
	Speed = [1–2];
	Memory degradation = 30%, [10–20]
		MCI

7	Error = 0%
	Speed = [1–3];
	Memory degradation off
		Healthy

8	Error = 50%
	Speed = [2–4];
	Memory degradation off
		Alzheimer

9	Error = 30%
	Speed = [2–4];
	Memory degradation = 15%, [10–20]
		Alzheimer

10	Error = 20%
	Speed = [2–3];
	Memory degradation = 40%, [10–20]
		MCI

11	Error = 40%
	Speed = [2–4];
	Memory degradation = 50%, [5-10]
		Alzheimer

12	Error = 60%
	Speed = [2–4];
	Memory degradation = 40%, [5–10]
		Alzheimer

The desired class column indicates the type of behavior that wanted to be simulated

11 and 12) that meet this criterion. Finally, all videos have at least one specialist who has diagnosed the desired class. If the categories are merged into two categories: (1) healthy subject and (2) unhealthy subject (MCI and AD), there are 11

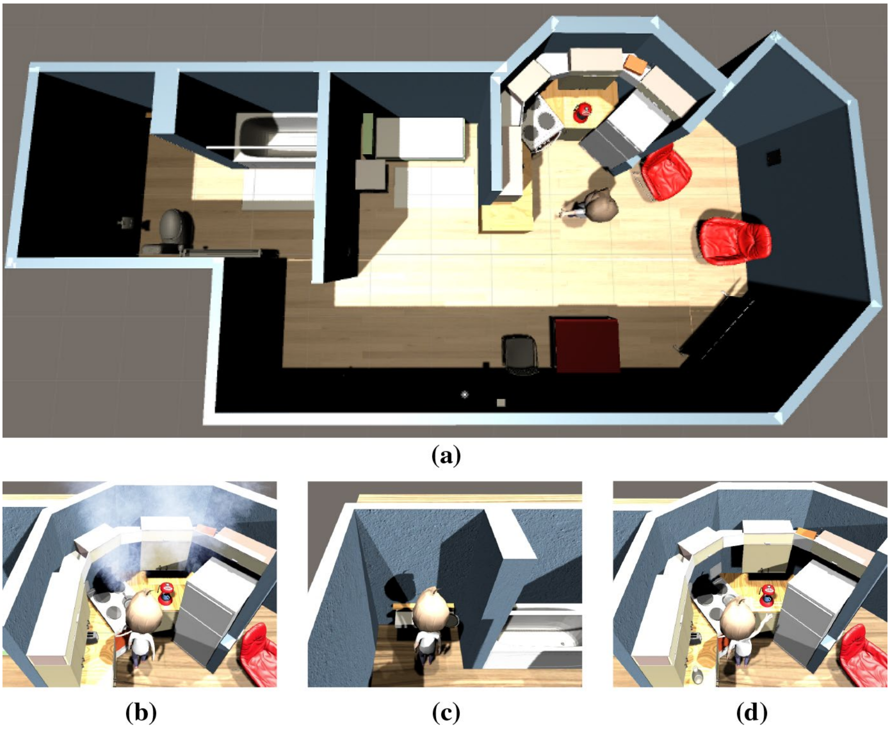


Fig. 18 Screenshots of **a** the environment used for the simulation; **b**–**d** extracts of the generated videos

Table 6 Sample of errors made by the avatar

Video identifier	Errors
1	None
2	Substitution: bottle in wrong place
3	Omission: pour egg into plate
4	None
5	None
6	Many requests for assistance
7	None
8	Several omissions
9	Several omissions
10	Omission: wash the frypan and store it on the shelf
11	Several omissions
12	Several omissions

Only errors of type omissions and substitutions are noted here because the video creator focused on this type of error. Requests for assistance are also listed. Errors of other types may have been generated in the videos, but these have not been noted. Errors such as misuse of a tool were not intentionally included because they would have been difficult to see in the videos (due to a lack of animation assets, in particular)

Table 7 Diagnosis of the specialists who watched the videos

Video identifier	Analyst 1	Analyst 2	Analyst 3	Desired class
1	Healthy	Healthy	Healthy	Healthy
2	Alzheimer	Alzheimer	Alzheimer	Alzheimer
3	MCI	MCI	MCI	MCI
4	MCI	Healthy	Healthy	Healthy
5	Healthy	MCI	Healthy	Healthy
6	Alzheimer	Alzheimer	MCI	MCI
7	Healthy	Healthy	Healthy	Healthy
8	Alzheimer	MCI	Alzheimer	Alzheimer
9	Alzheimer	Alzheimer	Alzheimer	Alzheimer
10	MCI	Alzheimer	Healthy	MCI
11	Alzheimer	Alzheimer	Alzheimer	Alzheimer
12	Alzheimer	Alzheimer	MCI	Alzheimer

out of 12 videos (all videos except video 10) where the diagnosis corresponds to the desired diagnosis. It can also be noted that except for video number 10, no specialist diagnosed the avatar with two classes of discrepancies (a healthy person is AD, and vice versa).

Tables 8 and 9, respectively, give the recall and the precision of the analysts in relation to our desired classes, in order to get an idea of the average sensitivity of analysts over the three classes. However, keep in mind that for the non-expert the classification of a video to the AD or MCI class was totally subjective.

6.1.3 Discussion

Our first observation is at the level of the simulated classes. Our proposal was able to simulate (“healthy”,“MCI”, “AD”) an appropriate behavior for each desired class. In addition, several configurations are possible to simulate each class. For example, in our paper, the behavior of persons with AD was simulated with several different parameters’ values.

Table 8 Table showing the “recall” for each analyst

Class	Analyst 1	Analyst 2	Analyst 3
Healthy	3/4	3/4	4/4
MCI	2/3	1/3	2/3
Alzheimer	5/5	3/5	4/5

Table 9 Table giving the “precision” for each analyst

Class	Analyst 1	Analyst 2	Analyst 3
Healthy	3/3	3/3	4/5
MCI	2/3	1/3	2/3
Alzheimer	5/6	4/6	4/4

However, it is important to keep in mind that some generated sequences were not retained, as the user felt that they did not correspond to the desired behavior class. The user had no apriori instructions on how to do this selection. Therefore, it was done based on common sense. Since one of our main objective is to enable non-expert to create such scenarios and to generate desired behaviors, it was decided to proceed without adding constraints on the user.

The justification behind the selection of the simulated sequences resides in the random aspect of the error injection. In fact, the random drawing at each tick does not guarantee that a minimum or maximum number of errors is triggered. Thus, putting probabilities considered very low (e.g., less than 10%) on several actions and on the memory degradation mechanism to try to simulate the behavior of someone with MCI does not prevent the system from triggering all or none of the errors.

In a second step, the team studied the reasons that may have prompted the specialists to give different diagnoses on certain videos. On video number 4, one specialist indicated that his diagnosis was conditioned by the fact that even if the avatar did not make any mistakes, it was a little slow. This was the case because its actions execution length was longer. On video number 5, the avatar did not make any errors. However, it did the actions disorderly. We suppose that this may be a cause of the MCI diagnosis of this sequence. For video 6, the person in charge of video generation considered the character to be a MCI even though it did not make any mistake. The avatar often asked for help and has had moments of misunderstanding. This explains why two specialists have diagnosed a person with AD. In videos 8 and 12, the characters performed the actions disorderly and omitted some of them. Some omissions may have been missed by specialists who have diagnosed a person with MCI. Video 10 is the one that raises most questions because there are three different diagnoses. In this video, the character forgets to wash the frying pan but does all the other actions by asking for help and having misguided moments. The "healthy" diagnosis was probably given because the omission was missed. For the choice between MCI and AD, it may be a different perception of the severity of misunderstandings and requests for help. It should also be noted that even in real-life some cases may be difficult to categorize and specialists may also disagree.

In conclusion, this approach allowed us to generate sequences of actions similar to those that people with MCI or AD would produce. However, the main drawback of this approach is the number of tests that must be performed to produce the action sequences that belong a priori to the desired class. The approach does not guarantee a minimum or maximum number of errors. Thus, if the user can associate his or her modeling with that of a person with MCI, the system can generate an action sequence belonging to that of a person with AD.

6.2 Experiment 2: Approach I versus Approach II, study of the disparity of behaviors generated by a scenario

The second approach proposed allows one to inject a number of errors into a sequence that is bounded by an interval predefined by the user. This additional feature is compared against the first one on the action sequences generation.

6.2.1 Protocol

First, let us define a reference behavior tree to be used by both approaches. The reference behavior tree is shown in Fig. 19.

Here, the idea is to generate several action sequences and to observe the difference between them. The test environment is a simulation of the behavior of a person with MCI. Simulating this profile is the most difficult because it is

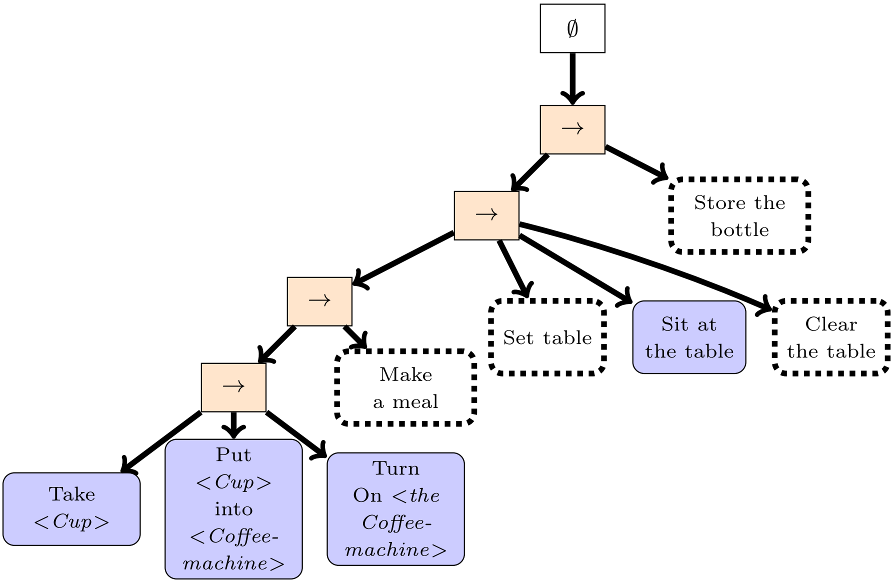


Fig. 19 BT for the experiment. The nodes between dashed lines are detailed in Figs. 20, 21, 22 and 23

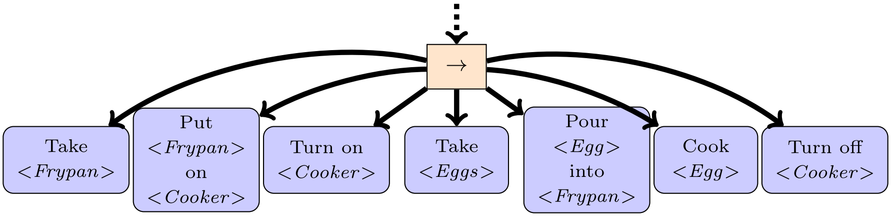


Fig. 20 Sub-tree ”Make a meal” of the BT of the experiment presented in Fig. 19

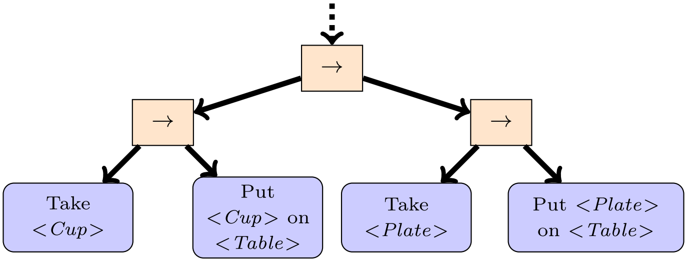
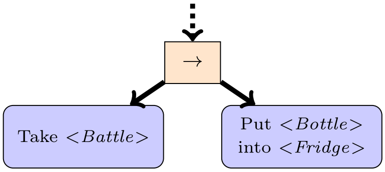


Fig. 21 Sub-tree ”Set table” of the BT of the experiment presented in Fig. 19

Fig. 22 Sub-tree ”Store the bottle” of the BT of the experiment presented in Fig. 19



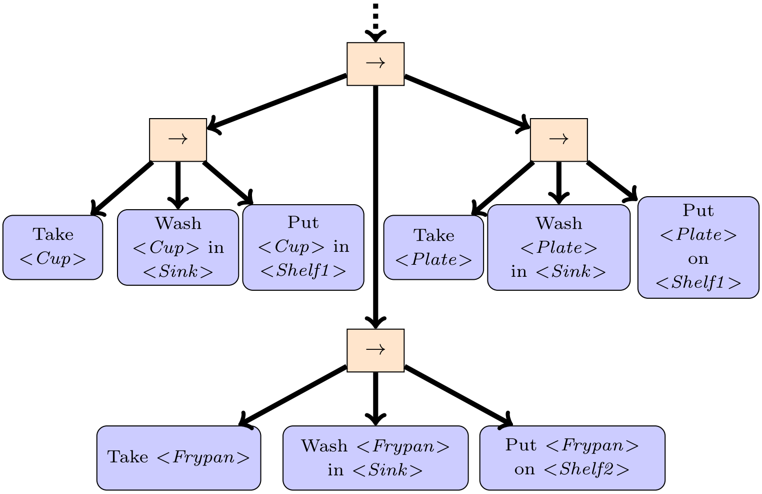


Fig. 23 Sub-tree ”Clear the table” of the BT of the experiment presented in Fig. 19

necessary to generate enough errors to fall into the category of MCI but not too many to be in the category of AD. To simulate these behaviors with the first approach, the same configuration of the 3rd avatar simulation is used. As a reminder, the error probability was set to 20%. With the second approach, the error interval is set to [2–4]. A hundred sequences were generated by recording each action completed. The following null hypothesis is formulated: there are no differences between the number of sequences acceptable for approach 1 and approach 2. We also propose to study the behavior of approach 2 both with and without the modification of the correction mechanism that checks before allowing to correct an error. This leads to the second null hypothesis: There are no differences between the number of sequences acceptable for approach 2 with the modification of the correction mechanism and approach 2(b) without the modification of the correction mechanism.

6.2.2 Result

The generation of the 100 sequences allowed to obtain Table 10 and the graphs of Figs. 24, 25, 26, and 27. The graphs show for each action, the number of errors obtained out of 100 simulations.

Table 10 Descriptive statistics of the number of errors for generations of 100 sequences with both approaches

Approach	Average	Standard deviation
1	7.01	2.97971
2	4	0
2(b)	4.33	0.68246

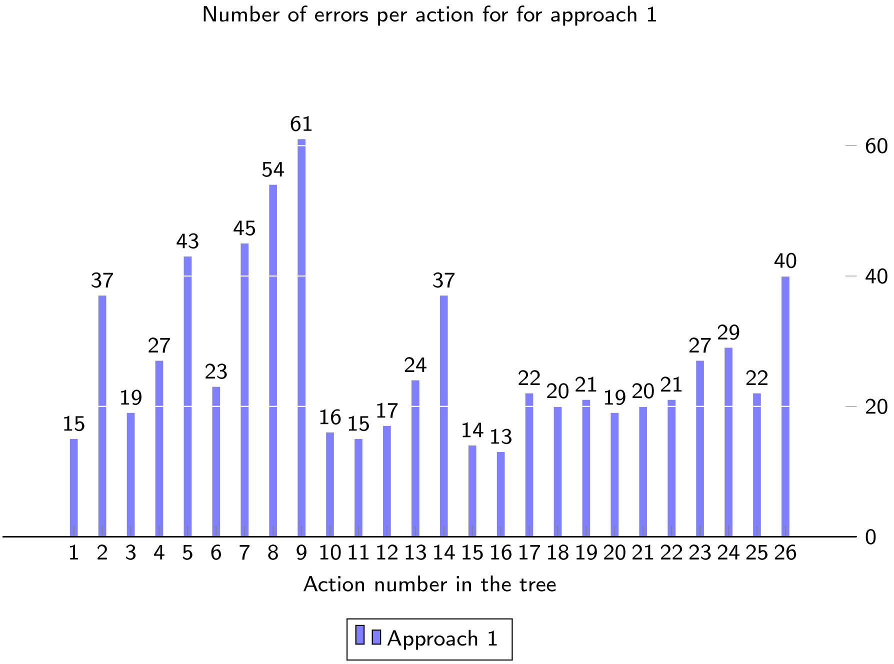


Fig. 24 Error distribution for approach 1

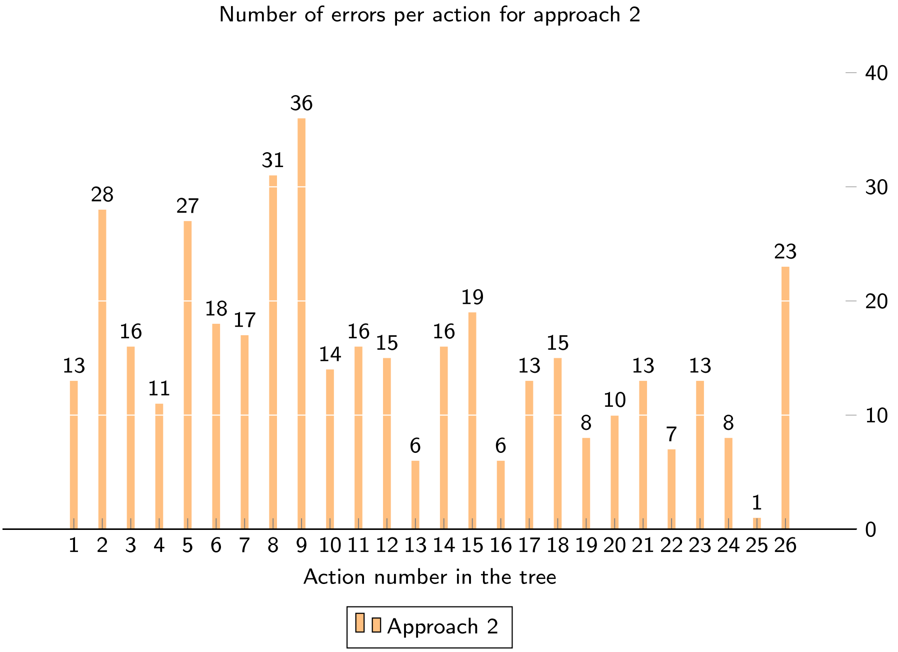


Fig. 25 Error distribution for approach 2

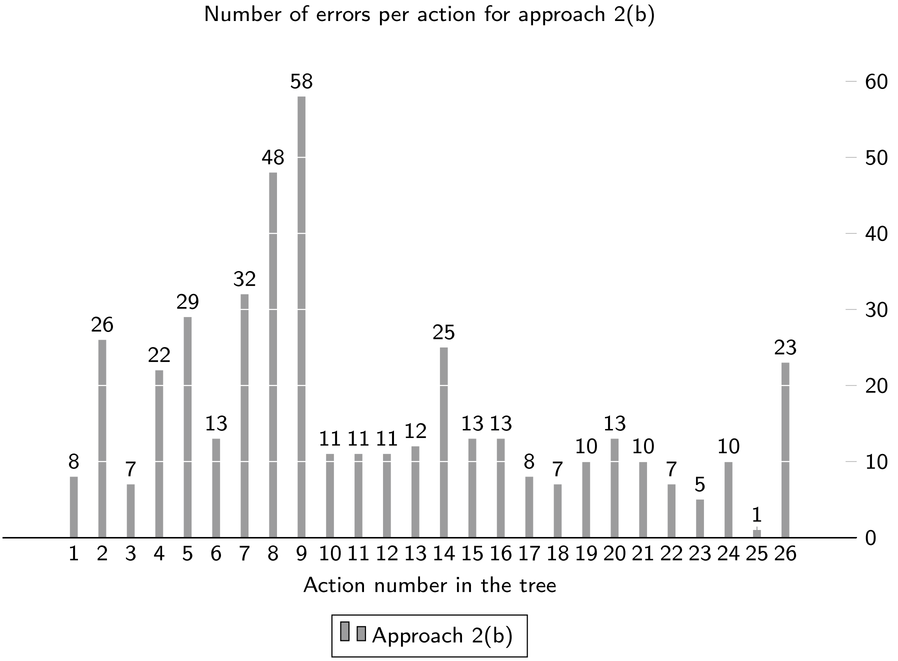


Fig. 26 Error distribution for approach 2(b)

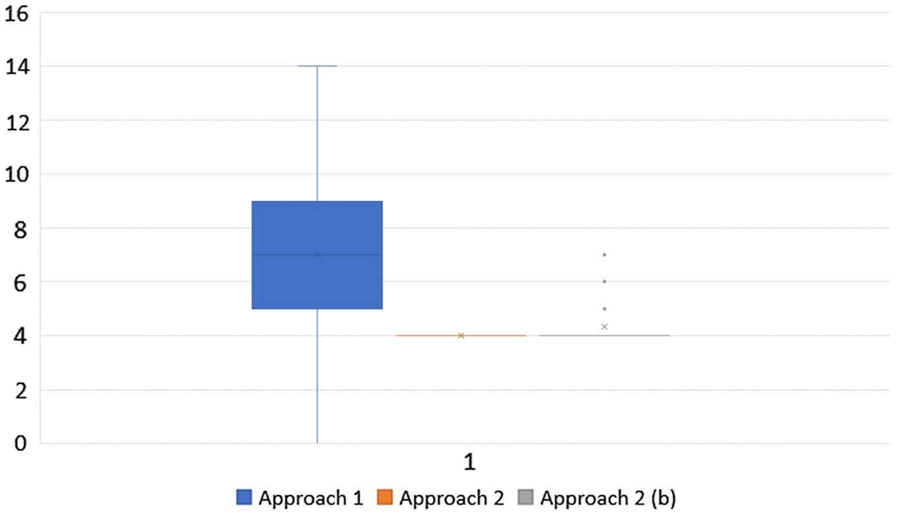


Fig. 27 Box plots of error distribution for approach 1, 2 and 2(b)

6.2.3 Discussion on Experiment 2

As it can be observed, the average error rate is higher with approach 1 (7.01) than with approach 2 (4) and approach 2(b) (4.33). Approach 2 can generate action sequences with an error number close to the upper bound, while approach 1 can generate several types of sequences. In some cases, approach 1 generated sequences without any error and in others sequences with more than 10 (14 errors in the worst case). If we were to keep only sequences with less than 4 errors, we would have kept only 21 sequences. With approach 2 without the correction mechanism, we would have kept 71. The correction mechanism seems to constrain the error injection system to stay close to the upper limit of the interval. Comparing the two versions of approach 2, we can see that the approach without error correction can exceed the upper bound of the interval because some uncorrected errors trigger the occurrence of other errors later in the simulation.

7 Discussion

The methods presented in this paper adhere to the principle of the behavior tree structure to define a scenario and adding random factors on the nodes to inject errors into the execution of the tree in order to generate a sequence of actions that would be similar to those produced by people with MCI or AD. We have observed that with the first approach, where individual probabilities of error are defined for each behavior or sub-activity, it is possible with a good configuration and a favorable draw to generate action sequences that emulate the behavior of persons with MCI or AD. Nevertheless, the large number of random parameters means that several sequences of actions may not correspond to the desired behavior. The second approach seems to limit the number of unwanted behaviors, as it is built upon the first one to force the injection of a minimum and a maximum number of errors to be produced. Indeed, the absence of these boundaries in the first approach could leave the generation of action sequences without error when one wishes to produce at least some. Therefore, the second approach has the advantage to offer more control to the user of the simulation tool. It allows him/her to define a scenario, a hierarchy in error frequencies, and an expected error interval for a simulation.

In the context of simulation, the approaches have the advantage of allowing the community to define scenarios of normal activity (i.e., performed by healthy people) and to observe how their system reacts when errors are injected into these initial scenarios. These approaches are therefore of interest in the pre-evaluation framework.

In terms of the sensors' data that would be generated as a result of these simulations, it should be noted that they remain strongly linked to the simulation model used for the sensors. For example, in the case of RFID-based object tracking algorithms, the model used to simulate signal forces must be able to generate data close to those collected by real antennas. Yet, it can be noted since the avatar can evolve in a virtual environment, it is possible to exploit the simulations to help in the choice and positioning of sensors in its real counterpart.

Regarding our goal of generating behaviors without the help of experts, we can see that the approach allows us to generate the desired behaviors. However, we can assume that in complex scenarios it may be difficult for a non-expert user to parameterize the system. To help with the parameterization of the system, a non-expert user would need a document that gives indications of the error frequencies of the people in each class. Indeed, even if the first evaluation made it possible to generate behaviors corresponding to the desired classes, the scenarios tested were quite simple and short in time. We can consider that they represented laboratory experimentation scenarios with fewer parameters to consider. Ideally, to facilitate the use of this type of tool, the user should be able to define a scenario and choose a profile (healthy, MCI or AD) to apply to that scenario.

On the level of the system's operation, the approach is deliberately simple. It consists in taking a behavior tree, choosing the elements where one wants to add errors, simulate and then observe. Additionally, it is possible to improve the system by exploring the use of contextual error probabilities. Indeed, these probabilities, compared to their preset version, would be more suitable to emulate the behavior of a working memory decreases according to real-world factors such as stress, the number of elements in memory or time, for example. With this type of probability, it would be possible to add a weighting according to these variables to force or reduce the error injection.

Finally, the approach may require large trees when many substitutions need to be modeled. One solution to this problem would be to add a semantic layer so that it would not be mandatory to define all possible substitutions. Instead, sets of interchangeable elements that share common criteria could be defined.

8 Conclusion

Simulation of human activities can be a great help in the field of intelligent environments. This makes it possible to carry out simulations and experiments without necessarily having to acquire expensive equipment such as sensors or a location or build the environment.

In this paper, we presented two approaches to simulate the behavior of individuals with MCI and AD. The principle of these approaches is to define a behavior tree that contains the interaction scenario and to use a process that interprets the tree by injecting errors. This creates action sequences that incorporate errors from the baseline scenario. More specifically, our approach is to define a behavior tree for the scenario and then, first, add composite errors to that tree to generate omissions or substitutions. In a second step, a potentially degraded mental map is used to add confusion to the avatar's behavior. In addition, we proposed two approaches to perform an error injection. The first approach in which each node has its own probability of triggering an error. The second approach includes a mechanism in charge of maintaining the number of errors within a given interval.

In order to evaluate our approaches, we conducted two experiments. In the first one, we generated 12 videos that were submitted to specialists in cognitive disorders for diagnosis. From the diagnoses, it was observed that our proposal allows one to generate different types of cognitive profiles. However, in some cases, it may be necessary to run a simulation several times in order to achieve the desired behavior.

The elimination of behaviors that the user considered invalid was allowed in the experiment, because the objective was to evaluate whether, with our approach, he/she can generate the behaviors he/she wants without any particular knowledge about the real behavior of the target individuals.

In some cases, with the percentages that were chosen, the errors were not triggered (or not enough) and the avatar showed the behavior of another class. In the second experiment, we compared the approaches by generating several videos and observing the distribution of errors. The approach 1 seemed to present a less predictable behavior by injecting more or less errors depending on the probability fixed on each node. Approach 2 seemed to allow one to have more control over the number of errors in a targeted interval. It was observed that the correction mechanism keeps the system close to the upper bound. Without this system, approach 2 may exceed this limit. This is because some errors lead to other errors later in the simulation. Overall, approach 2 seemed more promising for action sequence generation requiring more control.

As future work, we propose to improve the error-trigger probability calculation system to allow more control over the distribution of errors. We also aim to develop an approach allowing another management of the parallel activities and, in particular, the possibility to continue an activity if the other one fails. Moreover, while the current system allows one to make optional actions (for example, to put sugar or not in a coffee) through the omission operator, in the future, we aim to extend the set of operators. For instance, we could add an "optional" operator that would allow this kind of behavior without going through the omission operators in order to facilitate the reading of the behavior trees by a human. We also propose to explore the addition of operators that would allow one to build scenarios with errors or the avatar uses a tool in the wrong way. Finally, an interesting approach to explore would be to learn the behavior trees of the occupants of the premises and to add their behavior in order to validate the layout of the sensors. Error injection would degrade learned behaviors to observe the effects on the system.

Acknowledgements The Natural Sciences and Engineering Research Council of Canada (NSERC) funds and supports our research projects. Nathalie Bier is supported by the Fonds de recherche du Québec- Santé.

References

Alshammari, N., Alshammari, T., Sedky, M., Champion, J., Bauer, C.: Openshs: Open smart home simulator. Sensors **17**(5), 1003 (2017)

Alshammari, T., Alshammari, N., Sedky, M., Howard, C.: Simadl: Simulated activities of daily living dataset. Data **3**(2), 11 (2018)

Anderson, J.R.: Rules of the Mind. Psychology Press, Routledge (2014)

Angelucci, F., Spalletta, G., Iulio, Fd, Ciaramella, A., Salani, F., Varsi, A., Gianni, W., Sancesario, G., Caltagirone, C., Bossu, P.: Alzheimer's disease (ad) and mild cognitive impairment (mci) patients are characterized by increased BDNF serum levels. Curr. Alzheimer Res. **7**(1), 15–20 (2010)

Anvari-Moghaddam, A., Monsef, H., Rahimi-Kian, A.: Optimal smart home energy management considering energy saving and a comfortable lifestyle. IEEE Trans. Smart Grid **6**(1), 324–332 (2015)

Aramendi, A.A., Weakley, A., Goenaga, A.A., Schmitter-Edgecombe, M., Cook, D.J.: Automatic assessment of functional health decline in older adults based on smart home data. J. Biomed. Inform. **81**, 119–130 (2018). https://doi.org/10.1016/j.jbi.2018.03.009

Belchior, PdC, Holmes, M., Bier, N., Bottari, C., Mazer, B., Robert, A., Kaur, N.: Performance-based tools for assessing functional performance in individuals with mild cognitive impairment. Open J. Occup. Therapy **3**(3), 3 (2015)

Bellagente, P., Crema, C., Depari, A., Ferrari, P., Flammini, A., Lanfranchi, G., Lenzi, G., Maddiona, M., Rinaldi, S., Sisinni, E., et al.: Remote and non-invasive monitoring of elderly in a smart city context. In: Sensors Applications Symposium (SAS), 2018 IEEE, pp. 1–6. IEEE (2018)

Blender Foundation: Blender. https://www.blender.org/ January 2018 (2016)

Botía, J.A., Campillo, P., Campuzano, F., Serrano, E.: UbikSim website. https://github.com/emilioserra/UbikSim/wiki. Accessed 1 June 2020 (2014)

Bouchard, B., Gaboury, S., Bouchard, K., Francillette, Y.: Modeling human activities using behaviour trees in smart homes. In: Proceedings of the 11th PErvasive Technologies Related to Assistive Environments Conference, pp. 67–74. ACM (2018)

Bouchard, K., Bouchard, B., Bouzouanea, A.: Practical guidelines to build smart homes: lessons learned. In: Opportunistic Networking, Smart Home, Smart City, Smart Systems, pp. 1–37. CRC Press, London (2014)

Bratman, M.: Intention, Plans, and Practical Reason, vol. 10. Harvard University Press, Cambridge, MA (1987)

Brush, A., Lee, B., Mahajan, R., Agarwal, S., Saroiu, S., Dixon, C.: Home automation in the wild: challenges and opportunities. In: Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, pp. 2115–2124. ACM (2011)

Camilleri, G.: A generic formal plan recognition theory. In: Proceedings, 1999 International Conference on Information Intelligence and Systems, pp. 540–547. IEEE (1999)

Charniak, E., Goldman, R.P.: A bayesian model of plan recognition. Artif. Intell. **64**(1), 53–79 (1993)

Chen, L., Hoey, J., Nugent, C.D., Cook, D.J., Yu, Z.: Sensor-based activity recognition. IEEE Trans. Syst. Man Cybern Part C (Appl. Rev.) **42**(6), 790–808 (2012)

Chen, L., Nugent, C.D., Biswas, J., Hoey, J.: Activity Recognition in Pervasive Intelligent Environments, vol. 4. Springer, New York (2011)

Chiaravalloti, N., Goverover, Y.: Changes in the Brain: Impact on Daily Life. Springer, New York (2016)

Crawford, C.: Chris Crawford on Interactive Storytelling. New Riders, Indianapolis (2012)

Dillon, C., Serrano, C.M., Castro, D., Leguizamón, P.P., Heisecke, S.L., Taragano, F.E.: Behavioral symptoms related to cognitive impairment. Neuropsychiatr. Dis. Treat. **9**, 1443 (2013)

Do, H.M., Pham, M., Sheng, W., Yang, D., Liu, M.: Rish: A robot-integrated smart home for elderly care. Robot. Auton. Syst. **101**, 74–92 (2018). https://doi.org/10.1016/j.robot.2017.12.008

Dubois, D., Nkambou, R., Quintal, J.F., Savard, F.: Decision-making in cognitive tutoring systems. In: Advances in Intelligent Tutoring Systems, pp. 145–179. Springer, New York (2010)

Ferber, J.: Les systèmes multi-agents: vers une intelligence collective. InterEditions, Paris (1997)

Francillette, Y., Boucher, E., Bouzouane, A., Gaboury, S.: The virtual environment for rapid prototyping of the intelligent environment. Sensors **17**(11), 2562 (2017)

Franklin, S., Kelemen, A., McCauley, L.: Ida: A cognitive agent architecture. In: SMC'98 Conference Proceedings, 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No. 98CH36218), vol. 3, pp. 2646–2651. IEEE (1998)

Franklin, S., Patterson Jr., F.: The lida architecture: adding new modes of learning to an intelligent, autonomous, software agent. Pat **703**, 764–1004 (2006)

Geib, C.W., Goldman, R.P.: Partial observability and probabilistic plan/goal recognition. In: Proceedings of the International Workshop on Modeling Other Agents from Observations (MOO-05), vol. 8 (2005)

Giovannetti, T., Bettcher, B.M., Brennan, L., Libon, D.J., Burke, M., Duey, K., Nieves, C., Wambach, D.: Characterization of everyday functioning in mild cognitive impairment: a direct assessment approach. Dement. Geriatr. Cogn. Disorders **25**(4), 359–365 (2008)

Grześ, M., Hoey, J., Khan, S.S., Mihailidis, A., Czarnuch, S., Jackson, D., Monk, A.: Relational approach to knowledge engineering for pomdp-based assistance systems as a translation of a psychological model. Int. J. Approx. Reason. **55**(1), 36–58 (2014)

Hallé, S., Gaboury, S., Bouchard, B.: Activity recognition through complex event processing: first findings. In: Workshops at the Thirtieth AAAI Conference on Artificial Intelligence (2016)

Ho, B., Vogts, D., Wesson, J.: A smart home simulation tool to support the recognition of activities of daily living. Proc. South Afr. Inst. Comput Sci. Inf. Technol. **2019**, 1–10 (2019)

Hornbæk, K., Oulasvirta, A.: What is interaction? In: Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems, pp. 5040–5052. ACM (2017)

Humphreys, G., Forde, E.: Disordered action schema and action disorganisation syndrome. Cogn. Neuropsychol. **15**(6), 771–812 (1998)

Jiang, B., Fei, Y.: Smart home in smart microgrid: a cost-effective energy ecosystem with intelligent hierarchical agents. IEEE Trans. Smart Grid **6**(1), 3–13 (2015)

Kautz, H.A.: Reasoning About Plans. chap. A Formal Theory of Plan Recognition and Its Implementation, pp. 69–124. Morgan Kaufmann Publishers Inc., San Francisco, CA (1991)

Kormányos, B., Pataki, B.: Multilevel simulation of daily activities: Why and how? In: 2013 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA), pp. 1–6. IEEE (2013)

Lee, J.W., Cho, S., Liu, S., Cho, K., Helal, S.: Persim 3D: Context-driven simulation and modeling of human activities in smart spaces. IEEE Trans. Autom. Sci. Eng. **12**(4), 1243–1256 (2015)

Lee, J.W., Helal, A.S., Sung, Y., Cho, K.: Context activity selection and scheduling in context-driven simulation. In: Proceedings of the Symposium on Theory of Modeling & Simulation-DEVS Integrative, p. 9. Society for Computer Simulation International (2014)

Leitner, G.: The Future Home is Wise. Not Smart. Springer, New York (2015)

Liu, S., Helal, S., Lee, J.W.: Activity playback modeling for smart home simulation. In: International Conference on Smart Homes and Health Telematics, pp. 92–102. Springer (2015)

Lotfi, A., Langensiepen, C., Mahmoud, S.M., Akhlaghinia, M.J.: Smart homes for the elderly dementia sufferers: identification and prediction of abnormal behaviour. J. Ambient Intell. Humaniz. Comput. **3**(3), 205–218 (2012)

Luke, S., Cioffi-Revilla, C., Panait, L., Sullivan, K., Balan, G.: Mason: a multiagent simulation environment. Simulation **81**(7), 517–527 (2005)

Marcotte, R., Hamilton, H.J.: Behavior trees for modelling artificial intelligence in games: a tutorial. Comput. Games J. **6**(3), 171–184 (2017). https://doi.org/10.1007/s40869-017-0040-9

Marzinotto, A., Colledanchise, M., Smith, C., Ogren, P.: Towards a unified behavior trees framework for robot control. In: 2014 IEEE International Conference on Robotics and Automation (ICRA), pp. 5420–5427. IEEE (2014)

McKhann, G.M., Knopman, D.S., Chertkow, H., Hyman, B.T., Jack Jr., C.R., Kawas, C.H., Klunk, W.E., Koroshetz, W.J., Manly, J.J., Mayeux, R., et al.: The diagnosis of dementia due to alzheimer's disease: recommendations from the national institute on aging-alzheimer's association workgroups on diagnostic guidelines for alzheimer's disease. Alzheimer's & Dement. **7**(3), 263–269 (2011)

Morris, M.E., Adair, B., Miller, K., Ozanne, E., Hansen, R., Pearce, A.J., Santamaria, N., Viega, L., Long, M., Said, C.M.: Smart-home technologies to assist older people to live well at home. J. Aging Sci. **1**(1), 1–9 (2013)

Nazerfard, E., Das, B., Holder, L.B., Cook, D.J.: Conditional random fields for activity recognition in smart environments. In: Proceedings of the 1st ACM International Health Informatics Symposium, pp. 282–286. ACM (2010)

Paternò, F.: Concurtasktrees: an engineered notation for task models. In: The Handbook of Task Analysis for Human–Computer Interaction, pp. 483–503. Lawrence Erlbaum Associates, Mahwah (2004)

Paternò, F., Mancini, C., Meniconi, S.: Concurtasktrees: a diagrammatic notation for specifying task models. In: Human–Computer Interaction INTERACT'97, pp. 362–369. Springer, New York (1997)

Patterson, D.J., Fox, D., Kautz, H., Philipose, M.: Fine-grained activity recognition by aggregating abstract object usage. In: Proceedings. Ninth IEEE International Symposium on Wearable Computers, 2005, pp. 44–51. IEEE (2005)

Pavithra, D., Balakrishnan, R.: Iot based monitoring and control system for home automation. In: 2015 Global Conference on Communication Technologies (GCCT), pp. 169–173. IEEE (2015)

Schmitter-Edgecombe, M., Parsey, C.M.: Assessment of functional change and cognitive correlates in the progression from healthy cognitive aging to dementia. Neuropsychology **28**(6), 881 (2014)

Schneider, M.: Resource-aware plan recognition in instrumented environments. Doctoral Thesis (2010). https://doi.org/10.22028/D291-25968

Serna, A., Pigot, H., Rialle, V.: Modeling the progression of Alzheimer's disease for cognitive assistance in smart homes. User Model. User Adapt. Interact. **17**(4), 415–438 (2007)

Serrano, E., Botia, J.: Validating ambient intelligence based ubiquitous computing systems by means of artificial societies. Inf. Sci. **222**, 3–24 (2013). https://doi.org/10.1016/j.ins.2010.11.012

Synnott, J., Nugent, C., Jeffers, P.: Simulation of smart home activity datasets. Sensors **15**(6), 14162–14179 (2015)

Unity Technologies: Unity—game engine. https://unity3d.com. January 2018 (2016)

Van der Mussele, S., Mariën, P., Saerens, J., Somers, N., Goeman, J., De Deyn, P.P., Engelborghs, S.: Behavioral syndromes in mild cognitive impairment and alzheimer's disease. J. Alzheimer's Dis. **38**(2), 319–329 (2014)

Weiss, G.: Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press, Cambridge (1999)

Wobcke, W.: Two logical theories of plan recognition. J. Logic Comput. **12**(3), 371–412 (2002)

Wooldridge, M.: An Introduction to Multiagent Systems. Wiley, New York (2009)

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Yannick Francillette is a professor at the Université du Québec à Chicoutimi since August 2019. He received a Ph.D. in Computer Science from the University of Montpellier (France). He completed a postdoctoral fellowship at the LIARA laboratory of the Université du Québec à Chicoutimi in the field of intelligent environments. He worked on simulating intelligent environments and activities of daily life in smart houses. His main research interests are adaptive systems and serious games.

Eric Boucher contributed to serious game development during the first years of his studies as a programmer. His contributions were focused primarily on the development of a smart home simulator and a mobile game to encourage physical activity amongst individuals with cognitive limitations. After obtaining amaster's degree in project management, Eric worked in the banking industry and most recently in the video game industry. His areas of expertise are organizational transformation and process optimization for agile development teams.

Nathalie Bier is an associate professor of occupational therapy at the Université de Montréal. She is also a researcher at the Research center of the Institut universitaire de gériatrie de Montréal. The main goal of Nathalie Bier's research program is to better understand the impact of cognitive deficits in aging and dementia on everyday function, as well as to develop non-pharmacological approaches to promote aging in place – such as the use of cognitive rehabilitation and new technology. She is leading major projects in the field of smart environments and connected objects for persons with severe cognitive deficits.

Maxime Lussier holds a Psy.D./Ph.D in neuropsychology from the Université du Québec à Montréal and has done a postdoctoral fellowship in Rehabilitation Science at the Université de Montréal. He is doing research on smart homes for elderly in Dr. Nathalie Bier Ph.D. lab. His main research interests are related to gerontechnologies, cognitive plasticity, and factors that can modulate normal cognitive aging. See: M. Lussier, M. Couture, M. Moreau, C. Laliberté, S. Giroux, H. Pigot, S. Gaboury, K. Bouchard, P. Belchior, C. Bottari, G. Paré, C. Consel, N. Bier (2020). Integrating an Ambient Assisted Living monitoring system into clinical decision-making in home care: An embedded case study. Gerontechnology, 19(1), (2020). https://doi.org/10.4017/gt.2020.19.1.008.00.

Kévin Bouchard is a professor at the Université du Québec à Chicoutimi since August 2016. He completed his doctorate in computer science in 2014. He carried out a postdoctoral fellowship on ambient intelligence for the support of people who suffered traumatic brain injuries at the Université de Sherbrooke before spending a short time at CASAS lab of Washington State University. He then worked as Project Scientist at the Center for SMART Health at the University of California Los Angeles where he co-led a deployment project of ambient technology for a rehabilitation center in Santa Monica. His research interests mainly relate to the exploitation of artificial intelligence and machine learning for the development of technologies for health. He has, among other things, worked on RFID localization, ambient sensing, activity recognition with UWB, wearable technologies, etc. He has published over 70 papers and has been supervising/co-supervising more than twenty graduate students.

Patricia Belchior has received her PhD in Rehabilitation Sciences from the University of Florida and completed her post-doctor fellowship in cognitive aging at the same university. She is currently an associate professor at the School of Physical and Occupational Therapy at McGill University and a researcher at Centre de recherche de l'Institut universitaire de gériatrie de Montréal (CRIUGM). Her research focuses on cognitive and technological interventions in later life. She is particularly interested in understanding the potential of technology to promote social participation among older adults and the role of health care professionals in this area of practice.

Sébastien Gaboury received the Ph.D. degree in mathematics from the Royal Military College of Canada, Kingston, ON, Canada, in 2012. He is currently an associate professor and a Scientist with the Department of Mathematics and Computer Science, University of Quebec at Chicoutimi, Chicoutimi, QC, Canada, where he is also the head of the Ambient Intelligence Laboratory for the Recognition of Activities. His research is sponsored by the Natural Sciences and Engineering Research Council of Canada, the Québec Research Fund on Nature and Technologies, and the Canadian Foundation for Innovation. His current research interests include assistive technologies for elders and cognitively impaired people and artificial intelligence.

Affiliations

Yannick Francillette[1] · Eric Boucher[1] · Nathalie Bier[2] · Maxime Lussier[2] · Kévin Bouchard[1] · Patricia Belchior[3] · Sébastien Gaboury[1]

✉ Yannick Francillette
yannick_francillette@uqac.ca

Eric Boucher
eric.boucher1@uqac.ca

Nathalie Bier
nathalie.bier@umontreal.ca

Maxime Lussier
lussier.maxime@gmail.com

Kévin Bouchard
kevin_bouchard@uqac.ca

Patricia Belchior
patricia.belchior@mcgill.ca

Sébastien Gaboury
sebastien_gaboury@uqac.ca

[1] LIARA, Université du Québec à Chicoutimi, 555, Boulevard de l'université, Chicoutimi, QC G7H 2B1, Canada

[2] Centre de recherche de l'Institut de gériatrie de Montréal, 4565, Rue Queen Mary, Montreal, QC H3W 1W5, Canada

[3] McGill University, 3654 Prom Sir-William-Osler, Montreal, QC H3G 1Y5, Canada