Proceedings of the Federated Conference on
Computer Science and Information Systems pp. 1757–1763

DOI: 10.15439/2015F213
ACSIS, Vol. 5

# Multi-Agent System Simulation of Indoor Scenarios

Rafael Pax
Universidad Complutense de Madrid
Facultad de Informática,
28040 Madrid, Spain
Email: rpax@ucm.es

Juan Pavón
Universidad Complutense de Madrid
Facultad de Informática,
28040 Madrid, Spain
Email: jpavon@fdi.ucm.es

***Abstract*—This paper presents a flexible agent decision model for the simulation of indoor scenarios. There are different kind of applications with varying requirements, from the typical emergency evacuation where the physical interaction of crowds of agents are more relevant, to those that demand more sophisticated agent decision models as when testing smart environment applications. Existing tools usually focus on one of these issues, looking for efficiency in the solutions.**

**The agent decision model in this paper tries to get a balance between efficiency and flexibility in the specification of the agent behavior in simple and complex situations. This is applied in a simulation framework for indoor scenarios, although it could be extended to other settings.**

## I. Introduction

TESTING APPLICATIONS for smart environments is a difficult task. It requires the installation of sensors and actuators, the communications and the software for the control system, and the participation of persons who have to play the different scenarios. This is costly, both in economic sense as well as in time. Also, there are some situations that cannot be tested for practical reasons (e.g., a fire, people accidents). Furthermore, from the point of view of the developers, who are used to iterative processes, it is difficult to repeat the tests if they have to perform these with persons. At least for these reasons it is interesting to use simulation tools that provide some support for the development of smart environment applications.

A relevant aspect to be considered in this kind of tests is the modelling of the behavior of humans under different situations. The behavior for these scenarios requires at least the following: interactions among agents, with the environment, and the process for decision making.

There are several tools for simulation and design of how people behave in indoor scenarios (some of them commercial) [1]–[6]. They focus on the design of spaces and 3D appearance, where the agents are seen more like a crowd that can be characterized by simple behaviors with a fixed number of parameters, instead of considering them as individuals. Although they are appropriate to simulate specific scenarios, it is important to consider the human and social behavior of individuals when simulating how people interact with their environment, including other individuals.

Other works have better addressed the specification of the agent behavior, such as [7]–[13]. However, they have not sufficiently taken into account the methodological aspects for a design process when developing the agents' behavior. This is relevant when the simulation framework has to be used for different purposes and by other developers. In those cases, there is a need for a clearer agent model, with some support for the design at a higher level of abstraction that can be easily translated to an implementation. This is the purpose of MASSIS (**M**ulti-**A**gent **S**ystem **S**imulation of **I**ndoor **S**cenarios), a framework for modelling and simulation of the decision-making process of agents in multiple situations in indoor scenarios domain.

The rest of the paper is structured as follows. Section II describes the MASSIS framework. Section III explains how is modelled the agent's behavior. Section IV shows a case study to illustrate the approach. Finally, Section V presents our conclusions.

## II. The MASSIS framework

MASSIS is an agent-based simulation framework for indoor scenarios. It has a component-based architecture, built on open-source software components. An overview of the framework is shown in Figure 1.

Graphical modelling of the indoor environment is supported by SweetHome3D [14]. This is a well known package that is used to model all components involved in the simulation environment, such as walls, doors, stairs, people, etc. (see Fig. 2). Sweet Home 3D allows to design buildings with enough realism, in a relatively short time. It also allows the integration of extensions, providing significant flexibility when adding new features. MASSIS adds a set of plugins for this application, which lets the user to specify the characteristics of the elements of the building that will act as agents. In the case of people, can be weight, speed, inherent characteristics of the person (fear, courage, etc.) and link to their behavior. It is also possible to specify elements of the environment, such as sensors and actuators, which will have a reactive behavior.

When the building is created, the SweetHome3D representation of the building is transformed inside the simulation engine, which adapts it to the internal representation of MASSIS. One of the important issues when modeling

This work has been developed in the context of the project MOSI-AGIL-CM (with grant S2013/ICE-3019, by the Directorate General for Universities and Research of CAM) and by the Programa de Financiación de Grupos de Investigación UCM-Banco Santander with reference GR3/14.

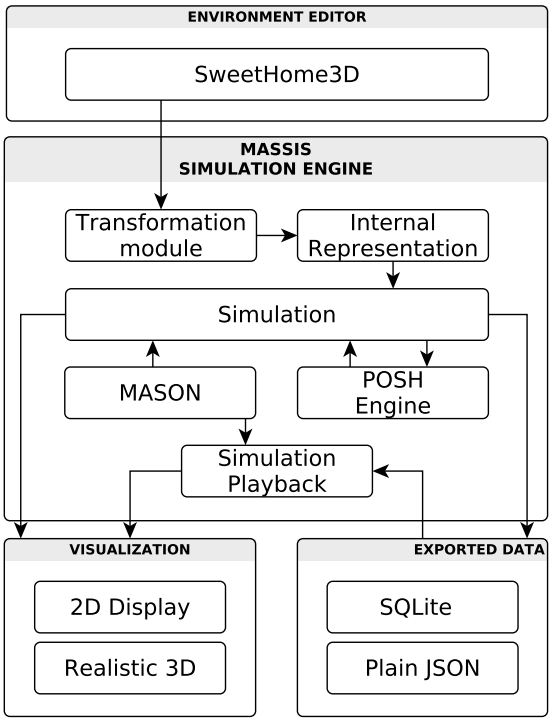


Fig. 1. MASSIS framework overview

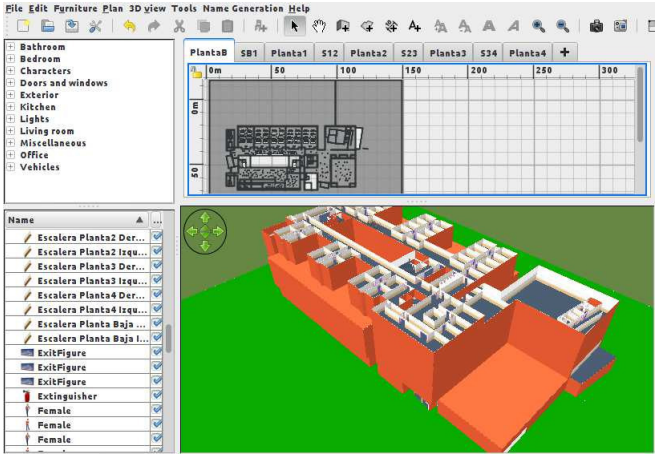

Fig. 2. Screenshot of the MASSIS's 3D editor, based on SweetHome3D.

indoor scenarios is the accuracy of the representation. Inside a building it is very common to have various elements at very small distances. If the elements of space are assigned to cells, the accuracy is reduced considerably. To solve this problem, MASSIS represents the elements in the building without discretizing the space; each element is represented as a polygon. For efficient computation of the locations of agents, data structures appropriate to this representation are used, such as different models of quadtree and polygon meshes. These data structures are used both for locating agents, perception

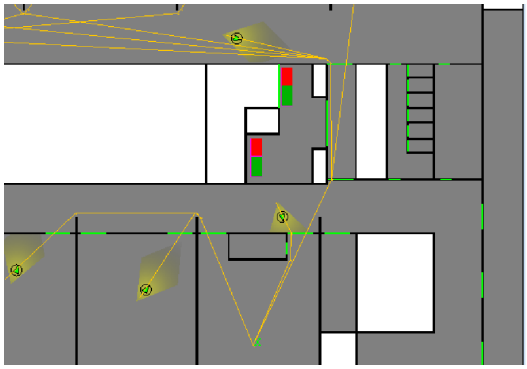

Fig. 3. Standard schematic display of MASSIS, showing bodies radio, vision areas and paths (*black circles green arrows*, *yellow polygons* and *yellow lines*, respectively). The green lines are doors, and the red and green squares are stairs.

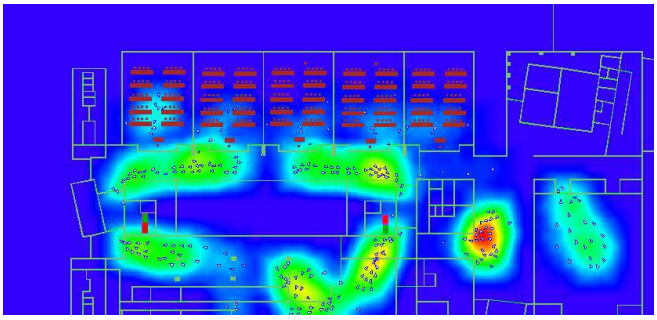

Fig. 4. Example of an user-defined schematic display: crowd density.

and pathfinding.

As simulation engine, MASSIS uses MASON [15], a lightweight multi-purpose agent-based simulation library. MASON has been chosen because it provides a good support for agent-based simulation platform, with well proven efficiency. Also, the clear separation of the simulation core and the GUI, allows MASSIS to use the MASON simulation core, while using its own display system.

The agents' behavior is controlled with the Pogamut's [16] POSH engine. (See Section III for more details)

All the changes made in the environment are reflected in real time by MASSIS' 3D (Fig. 9) and 2D (Fig. 3) displays, and can be logged in JSON format (Listing. 1), as a single zipped file or in a SQLite database for further analysis. Although 3D display is more realistic, the 2D view is useful for analysis and debugging. Also, the 2D visualization API allows the creation of user-defined layers (Fig. 4).

Once a simulation is performed, the exported data can be used to playback all events that have occurred during the execution of the simulation, i.e., the agents will behave in the same way they did during the simulation.

## III. Agent decision model in MASSIS

The aspects of human behavior that are of interest for modelling indoor scenarios in the MASSIS framework are the mechanisms that humans use to deal with problems reasoning

```
{
    "velocity": { "x": 32,"y": 58},
    "visionRadio": 300,
    "maxforce": 10,"maxspeed": 15,
    "properties": {"steering.separation":
        70,...},
    "locationState":{
    "angle": 0.7853982,"floorId": 8,
    "centerX": 4975.2285,"centerY":
        4108.2695,...},
    "id": 3673
}
```

Listing 1. An agent's saved state in *JSON* format.

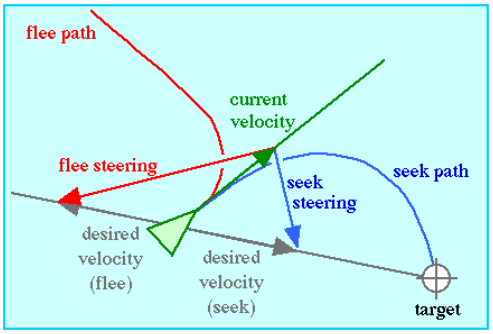


Fig. 5. *Seek and Flee* Steering behavior [17].

from context, making use of collective intelligence, and how this intelligence is used in problem solving.

Each agent in MASSIS has its own behavior, which is computed at a high-level (reactive plans) and at a low level (speed, position, angles, density, etc.)

When the high level decides *what to do next*, the action is executed by the low-level module, which carries all the necessary operations (movement, animation, etc.) Both high-level and low-level behaviors are affected by the state of the agent, altering the decision making process (e.g., a *scared* agent may choose a different route to reach a target, probably a longer one) and the action execution (it will be moving faster).

### A. Low-level behavior

The low-level module deals with the perception of the environment and a set of basic behaviors for interacting with it. These behaviors are mostly a set of *steering behaviors* [17], which control the most basic movement component of the agent.

Using a simple force model, steering behaviors produce smooth, life-like movements, providing agents the ability to navigate around the environment in a realistic manner. The forces applied to the agent can be combined in order to create more complex behaviors (e.g. collision avoidance, path-following, leader following, queuing, etc.). Fig. 5 shows the forces present in the basic *Seek and Flee* behavior.

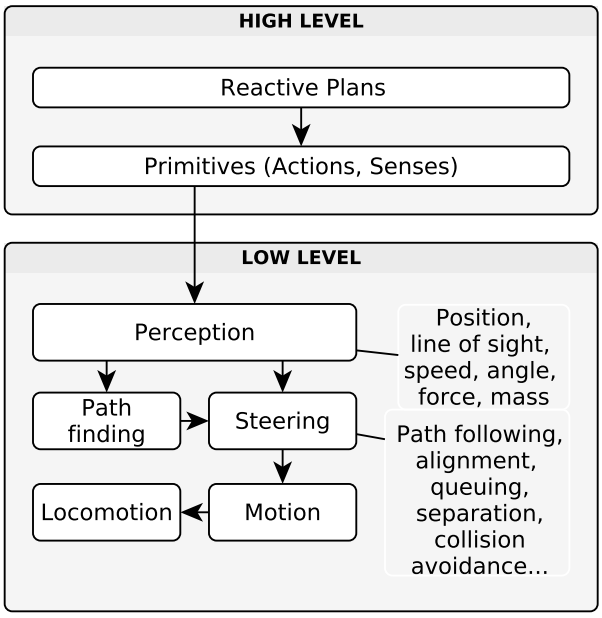


Fig. 6. MASSIS 's human behavior agent model.

### B. High-level behavior

The high-level behavior deals with decision making, learning and communications with other agents and makes decisions based on the knowledge of the environment, which is provided by the low-level module.

The architecture of this behavior follows the BOD (Behavior Oriented Design) method [18]. This method for building agents combines the advantages of Behavior Based AI [19], [20] and object-oriented design approaches.

In MASSIS this is applied to facilitate the design of agents that are capable of running in parallel and of generating a behavior that can satisfy multiple objectives that may conflict with each other.

The difficulty of making an autonomous agent is that many of the goals that the agent wants to be accomplished must be carried out at the same time. An agent may have the desire to be loved, be promoted at work and having breakfast in the morning.

Additionally, these goals must be achieved in an unpredictable environment, which can complicate or even make easier the way in which the agent tries to accomplish its goals. Developing a system of agents under BOD involves dividing the implementation into two different parts:

1) A library of Behavior modules. They consist of a set of classes representing a set of modules for perception, action and learning. These are *primitives*, actions and senses that can be called from the mechanism of action selection. They also provide a place where certain states and knowledge can be stored in order to perform those actions, and they contain code that describes any sense that needs to be carried out to acquire that state and

```
public ActionResult run() {
    boolean isInLoc;
    SimulationObject target=getTarget();
    Location tLoc=target.getLocation();
    isInLoc=agent.approachTo(tLoc);
    if (isInLoc) {
        return ActionResult.RUNNING_ONCE;
    }
    else {
        return ActionResult.FINISHED;
    }
}
```

Listing 2. Example of the primitive action *Go to target*

```
(C search-for-object
vars($type, $storeFlag="IS_NEAR_TARGET")
(elements
 ((has-target (trigger ((HasTarget))) approach
    ))
 ((is-object-visible
  (trigger ((SeesElement($attr=$type,$value=1)
       true ==)))
  setTarget($target="?LAST_SEEN_OBJ")))
((search ExploreAction)))
)
```

Listing 3. POSH code of parametrized competence *search-for-object*. *ExploreAction* is a primitive action, *approach* is a competence and words preceded by $ are variables.

knowledge. In brief, they determine *how* to do something. These senses and actions are created in the native language for the problem space (in the case of MASSIS, Java; see for instance the code in the listing 2)

2) POSH Dynamic action selection scripts. These allow to determine priorities between modules. The BOD architecture uses a POSH dynamic plan when an action should be carried out.
   A POSH(Parallel-rooted, Ordered Slip-stack Hierarchical) plan is a prioritized set of conditions and the related actions to be performed when the conditions have been met.
   It consists of drive collections, competences, and action patterns.
   - Drive collections are the root of every POSH plan. On the action selection step, the drive collections select which goal the agent must try to accomplish. They can be seen as a set of conditional rules, that are evaluated from highest to lowest priority.
     Every time the condition of the drive collection with highest priority is satisfied (a higher rule interrupts a lower one), the POSH engine executes the corresponding action pattern or competence.
   - Competences are a set of nested if-then conditional trees, which can be reused several times inside the reactive plan. They differ from the drive collections in the way they are executed; rules they do not interrupt each other.

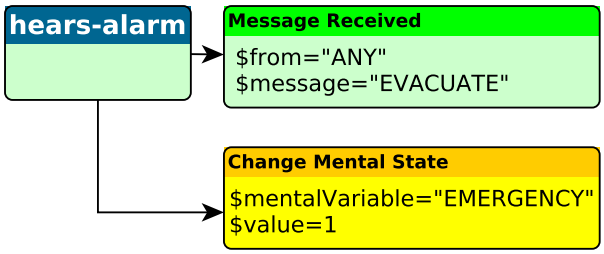


Fig. 7. *Mental state* modification in a POSH plan (*yellow*), triggered by the sense "Message Received" (*green*) on the Drive Element "hears-alarm"(*blue*)

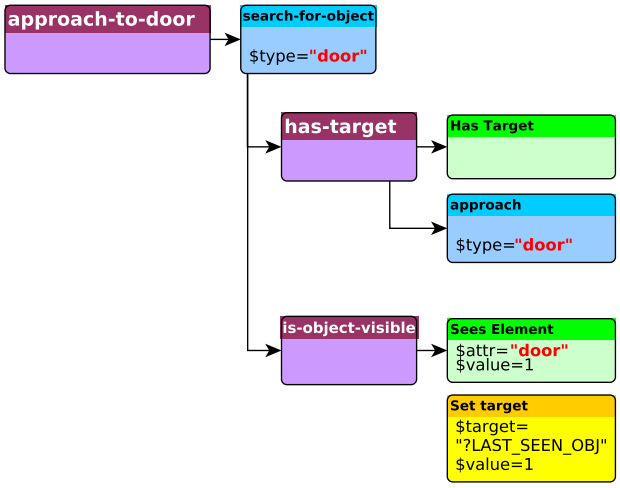


Fig. 8. Propagation of the value "door" in the parametrized competence of searching an object. **Note**: Some elements were omitted for clarity.

- Action Patterns are simple sequences of actions. Although they are not very flexible, they provide a layer of abstraction very useful when grouping actions.

MASSIS encourages the use of variables and the agent's *Mental State* in POSH plans. *Mental States* are intended for representing the knowledge of the agent about its environment as a set of key-value pairs, but can be used for any other purpose, such as storing control variables in order manage the plan execution (see Fig. 7).

Also, as MASSIS uses the Pogamut's extension of the POSH language, *actions*, *senses*, *action patterns* and *competences* can be parametrized. This provides considerable flexibility, allowing the reuse of elements in the reactive plans (see Fig. 8 and Listing 3).

## IV. Case Study: Emergency Simulation

Public buildings have some protocols for dealing with emergency situations, which may involve, for instance, evacuation of the building. Testing these protocols requires some planning and cost. Simulation can help to this task. This case study addresses this kind of situation for the building of the Facultad de Informática at UCM. In this scenario, a teacher is responsible for guiding students safely to the building's exit in case of emergency. For illustrating purposes, this is the protocol for a teacher in an emergency situation:

> *When the alarm sounds the teacher of the group should go to the classroom door and order the students to close the windows if there is a fire. If instead of a fire there is a bomb threat, windows and doors should be left open. Students will leave the classroom through the door and they will be waiting for the teacher outside. The teacher will be the last person to leave the classroom. Once there is nobody in the classroom, the teacher will place a chair at the entrance of it,as an indication that the room has been evacuated entirely. Then the teacher will guide students toward the nearest exit.*

Modelling the teacher agent involves the following basic skills:

- Hearing and vision capabilities.
- Ability to communicate with other agents by voice.
- Movement.
- Interaction with objects in the environment: Taking an object, carrying it , dropping it.

These skills are candidates to be *primitive* actions and senses. These primitives, such as the movement one (Listing 2), are used by the reactive plan as *Triggers* of *Drive Elements*, components of *Action Patterns*, or they form part of one or more *Competences*. Figure 10 shows part of the teacher's reactive plan. Figures 9 and 11 show the initial state of the simulation. When the alarm sounds, and the teacher goes to the door (using *go-to-nearest-door* competence). Figure 12 illustrates the moment when the teacher tells the students that they must close the windows (*act-windows* competence). When the windows are closed, and the students outside (Fig. 13), the teacher takes the first chair he sees and he moves it to the

Fig. 9. 3D view of the simulation

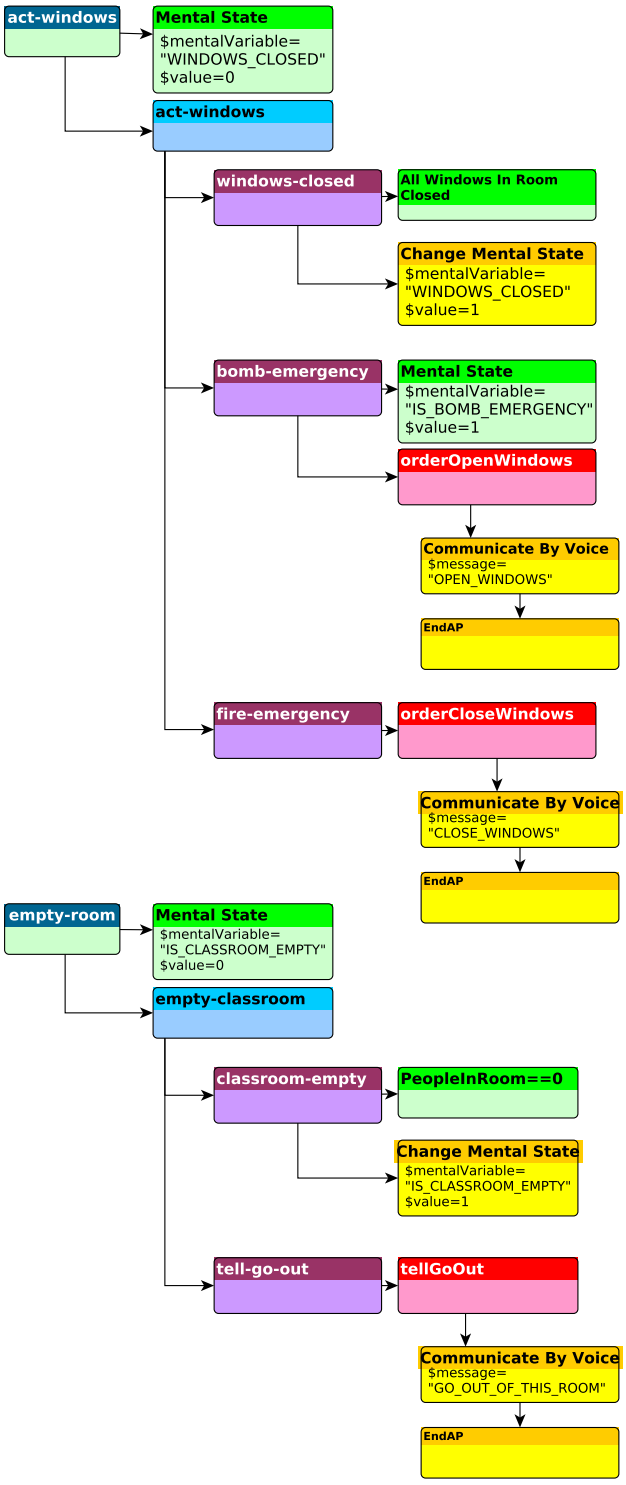


Fig. 10. Partial overview of the Case Study POSH plan

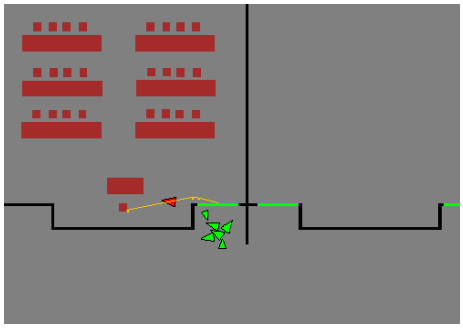

Fig. 13. Teacher going to take the nearest chair.

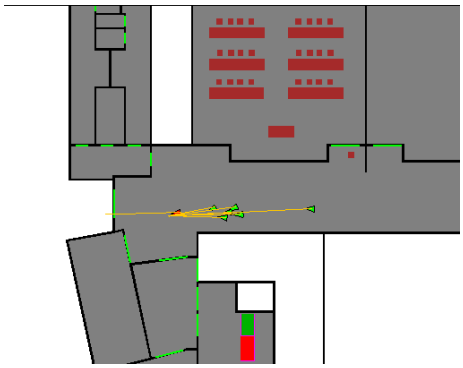

Fig. 14. The students follow the teacher for escaping from the building.

Fig. 11. Teacher going to the door.

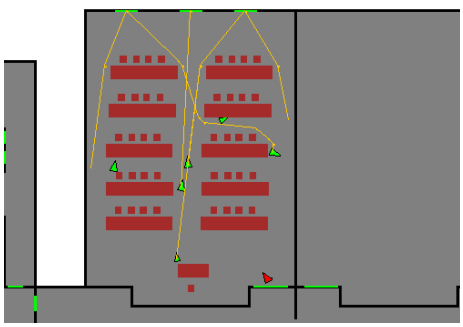

Fig. 12. The students receive the "close windows" message, and they proceed to close the windows.

door. After that, the teacher guides his students to the nearest exit (cf. Fig. 14).

## V. Conclusion and Future Work

This paper has presented MASSIS, a multiagent-based simulation framework that supports the decision making process of humans when solving problems. Agent behavior is structured in low-level and high-level behavior components, extending Pogamut's POSH implementation model, with the addition of features that facilitate the separation of decision-making process and low level actions. MASSIS provides a rich set of low-level behavior components for the simulation of indoor scenarios. This has required to MASSIS the extension of the SweetHome3D environment with plugins for linking agent's behavior in the simulation. In order to apply MASSIS to other kind of scenarios (e.g., a city), new low-level behavior components should be implemented and integrated with another graphical design package that supports the definition of the new environment. In this sense, MASSIS can be easily extended. MASSIS provides as well a rich log capability, which can be the basis for further analysis of the scenarios. The integration of existing analysis tools is one of the most relevant issues for the next version ofthis framework.

## Acknowledgment

A special thanks to P.R. Hijas for her comments that greatly improved the development of the MASSIS' framework.

## References

[1] Legion, "Science in Motion," http://www.legion.com, [Online; accessed Mar. 2015].
[2] M. Owen, E. R. Galea, and P. J. Lawrence, "The exodus evacuation model applied to building evacuation scenarios," *Journal of Fire Protection Engineering*, vol. 8, no. 2, pp. 65–84, 1996.
[3] PedGo, "TraffGo HT." http://www.traffgo-ht.com/de/pedestrians/products/pedgo/index.html, 2006, [Online; accessed Mar. 2015].
[4] M. MacDonald, "STEPS," http://www.steps.mottmac.com/, 2009, [Online; accessed Mar. 2015].
[5] T. Engineering, "Pathfinder," http://www.thunderheadeng.com/pathfinder/, 2006, [Online; accessed Mar. 2015].
[6] Golaem, "Golaem Crowd: Artist-Driven Crowd Simulation," http://golaem.com/content/products/golaem-crowd/overview, 2011, [Online; accessed Mar. 2015].
[7] X. Pan, C. S. Han, K. Dauber, and K. H. Law, "A multi-agent based framework for the simulation of human and social behaviors during emergency evacuations," *Ai & Society*, vol. 22, no. 2, pp. 113–132, 2007.
[8] L. Saîfi, A. Boubetra, and F. Nouioua, "Approaches to modeling the emotional aspects of a crowd," in *Modelling and Simulation (EUROSIM), 2013 8th EUROSIM Congress on*. IEEE, 2013, pp. 151–154.
[9] M. Software, "Simulating Life," http://www.massivesoftware.com/, 2002, [Online; accessed Mar. 2015].
[10] S. Wu and Q. Sun, "Computer simulation of leadership, consensus decision making and collective behaviour in humans," *PloS one*, vol. 9, no. 1, 2014.
[11] A. C. Bicharra, N. Sánchez-Pi, L. Correia, and J. M. Molina, "Multi-agent simulations for emergency situations in an airport scenario," *ADCAIJ: Advances in Distributed Computing and Artificial Intelligence Journal*, vol. 1, no. 3, pp. 69–73, 2013.

[12] T. Bosse, M. Hoogendoorn, M. C. Klein, J. Treur, C. N. Van Der Wal, and A. Van Wissen, "Modelling collective decision making in groups and crowds: Integrating social contagion and interacting emotions, beliefs and intentions," *Autonomous Agents and Multi-Agent Systems*, vol. 27, no. 1, pp. 52–84, 2013.

[13] R. Hocevar, F. Marson, V. Cassol, H. Braun, R. Bidarra, and S. R. Musse, "From their environment to their behavior: a procedural approach to model groups of virtual agents," in *Intelligent Virtual Agents*. Springer, 2012, pp. 370–376.

[14] E. PUYBARET, "Sweet Home 3D," http://www.sweethome3d.com/, 2005, [Online; accessed Mar. 2015].

[15] S. Luke, C. Cioffi-Revilla, L. Panait, K. Sullivan, and G. Balan, "Mason: A multiagent simulation environment," *Simulation*, vol. 81, no. 7, pp. 517–527, 2005.

[16] J. Gemrot, R. Kadlec, M. Bída, O. Burkert, R. Píbil, J. Havlíček, L. Zemčák, J. Šimlovič, R. Vansa, M. Štolba *et al.*, "Pogamut 3 can assist developers in building ai (not only) for their videogame agents," *Agents for Games and Simulations*, pp. 1–15, 2009.

[17] C. W. Reynolds, "Steering behaviors for autonomous characters," pp. 763–782, 1999.

[18] J. J. Bryson, "Intelligence by design: principles of modularity and coordination for engineering complex adaptive agents," 2001.

[19] R. A. Brooks, "A robust layered control system for a mobile robot," *Robotics and Automation, IEEE Journal of*, vol. 2, no. 1, pp. 14–23, 1986.

[20] R. A. Brooks., "Intelligence without reason," *The artificial life route to artificial intelligence: Building embodied, situated agents*, pp. 25–81, 1995.