ABSTRACT
Designing the agent model in a multiagent simulation is a challenging task due to the generative nature of such systems. In this contribution we present an extension to the multiagent simulation platform SeSAm, introducing a learning-based design strategy for building agent behavior models.

Categories and Subject Descriptors
I.2.11 [Artificial Intelligence]: Distributed Artificial Intelligence—Intelligent Agents

General Terms
Design, Experimentation, Performance

Keywords
Multiagent Simulation, Agent Learning

1. INTRODUCTION
The generative nature of multiagent simulations makes it hard, specially at the design phase, to identify the particular local agent behavior that will produce the desired macro-level system behavior. It is necessary to devise a systematic way of modeling the behavior program of the agent, thus bridging the micro-macro levels gap. We recently suggested a methodology for designing agent behavior models using adaptive agents [1]. Instead of being equipped with their behavior program since the beginning, the agents learn during simulation and report their learned behavior program to the modeler. However, specially for inexperienced modelers or researchers without programming knowledge, it is essential to have the tools supporting our proposed design strategy. This contribution presents the implementation of the learning tools that allow agents to learn their behavior models. The chosen multiagent simulation platform is SeSAm (www.simsesam.org).

In the following we give a short overview of the agent design strategy in Section 2, then we describe its integration into SeSAM in section 3. The paper ends with the test case and the results obtained, in Section 4, and conclusion in Section 5.

3.3 Learning Architecture

The learning architecture explores the combinations of perceptions and actions, which are evaluated based on a utility value, estimated using the objective function. The final model should be abstracted into a decision tree format, which not only is an intuitive notation for decision-making processes, but also allows the model to be exported and used as a standard activity-graph agent in SeSAm; to be post processed, as for instance using graph-matching algorithms to find differences in models learned in different configurations of the environment.

The actual learning algorithms are implemented as interchangeable modules inside the learning reasoning engine. We call these modules learning cores. Any variation of learning algorithms that follows the design strategy presented here can be implemented as a learning core. At the current stage, three learning cores are available: a) Q-learning+: Q-learning is used for learning the situation-action pairs governing the agent behavior. The best pairs are used as the training set to build the decision tree with the C4.5 algorithm; b) LCS: the XCS learning classifier system is used to learn situation-actions pairs, which are abstracted into decision trees using the C4.5 algorithm; c) Genetic Programming: perceptions and actions are assembled into decision tree programs, evolved through generations using random genetic programming. The objective function is used as the fitness function of each individual tree.

When the behavior becomes interesting during the simulation, the modeler can get the abstracted decision tree model of the behavior learned and export it as a standard activity-graph agent. Figure 1(b) shows the visualization of the simulation and Figure 1(c) shows the decision tree model learned in a simple pedestrian evacuation scenario. We want to develop a behavior program for pedestrian agents, which should leave the room as fast as possible, avoiding collisions.

We followed a number of steps to use the learning tools: 1) model the environment in SeSAm; 2) add to the simulation model a Learning Agent; 3) create the interfaces of the learning agent: sensors are created as boolean functions (perception of obstacles and exit) and actions as functions operating the agent (moving), using the available primitives in SeSAm. Figure 1(a) shows the edit window with the defined sensors and actions; 4) define a numeric function evaluating the state of the agent, which will be called by the simulator after every action and the value reported to the learning core. Here, it is the sum of a reward for avoiding collisions and a reward for reducing the distance to the exit; 5) choose one of the available learning cores – in this case Q-learning+.

We ran different experiments changing the settings of the environment: size of the room, number of agents and obstacles. Figure 1(b) depicts one of these cases, where one agent leaves a 20x30m room with 12 randomly positioned column-type obstacles. Experiments are composed by trials; in each trial the agent is randomly positioned in the environment and has to find the exit. The final decision tree model generated is an abstraction of the full behavior of the agent. The objective function directly affects the outcome: a higher pressure on avoiding collisions results in a decision tree with more rules for turning; if the objective prioritizes the distance-to-the-exit factor, the learned decision tree is more compact. More learning results can be found in [2].

5. CONCLUSION

We presented new tools, available in SeSAm, to learn agent behavior models. The model learned serves as an inspiration for the modeler in further steps of the modeling process or can be directly used in the simulation.

6. REFERENCES