Learning Agent Models in SeSAm

(Demonstration)

Robert Junges
Örebro University
70182 Teknikhuset
Örebro, Sweden
robert.junges@oru.se

Franziska Klügl
Örebro University
70182 Teknikhuset
Örebro, Sweden
franziska.klugl@oru.se

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.

Appears in: *Proceedings of the 12th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2013)*, Ito, Jonker, Gini, and Shehory (eds.), May, 6–10, 2013, Saint Paul, Minnesota, USA.

2. LEARNING AGENT BEHAVIOR

The design strategy adopted here is based on the following idea: an appropriate conceptual model of the overall system can be developed by setting up a simulation model of the environment, determining agent sensors and effectors, and giving a function to evaluate agent validity as "performance". Put into in the simulated environment, agents are then able to learn an appropriate behavior program that fulfills the objective expressed by the performance function. The learned behavior is then providing the modeler with a draft agent design that can be used in later phases of model development.

There are two critical steps in this overall process: the definition of the performance function and the actual learning mechanism used. The performance function should express the agent's goal with regards to the observed valid aggregated behavior of the system, so we also achieve a valid macro level behavior. The learning mechanism should be able to cope with the situation-action-reward model. But more importantly, it should produce a readable description of the behavior learned, making the whole process transparent to the modeler.

3. INTEGRATION

This section presents SeSAm and the changes made to it.

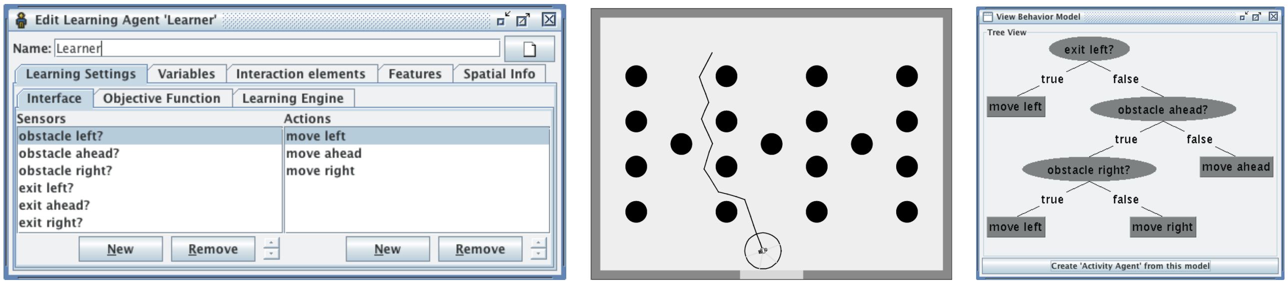
3.1 SeSAm

SeSAm [3] stands for **Shell for Simulated Multiagent Systems** and it is a modeling and simulation environment that combines concepts of declarative high-level model representation with visual programming. It is unique as it replaces programming in a standard language with visual programming, but scales with complex models. SeSAm contains all components that are necessary for a useful simulation and modeling environment: different data input and export options, visualization of the ongoing dynamics, etc.

The high-level modeling language for behavior consists of elements called primitives, representing agent actions, predicates for perception or other functions for processing information. In the standard SeSAm, these primitives are organized as a UML-like **activity-graph with rule-based transitions between activities**. Activities represent action scripts that are executed by an agent repeatedly while the agent is in the respective activity.

3.2 Learning in SeSAm

To integrate the design strategy described in Section 2, we added a new agent type: the learning agent, with the *learn-*



(a) Agent creation window. (b) Visualization of the simulation. (c) Decision tree learned.

Figure 1: Simulation and learning tool interfaces.

ing reasoning engine. Contrary to the standard agent, for which the entire behavior reasoning needs to be formulated as an activity-graph, the learning agent generates its behavior controller during simulation. The modeler only **needs to define what the perception and action functions do and not how they will be used** by the agent. Also, the modeler needs to define the objective function. The selected learning engine operates on creating a model, which will depend on the particular learning mechanism chosen, but will finally report a decision tree abstraction of the behavior (see Section 3.3).

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.

4. TEST CASE

The test case chosen for this contribution is a simple room 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

[1] R. Junges and F. Klügl. How to design agent-based simulation models using agent learning. In *Proceedings of the Winter Simulation Conference 2012*. WSC, 2012.

[2] R. Junges and F. Klügl. Programming agent behavior by learning in simulation models. *Applied Artificial Intelligence*, 26(4):349–375, 2012.

[3] F. Klügl, R. Herrler, and M. Fehler. Sesam: implementation of agent-based simulation using visual programming. In *AAMAS*, pages 1439–1440. ACM, 2006.