Evaluation of Techniques for a Learning-Driven Modeling Methodology in Multiagent Simulation

Robert Junges and Franziska Klügl

Modeling and Simulation Research Center
Örebro University, Sweden
{robert.junges,franziska.klugl}@oru.se

Abstract. There have been a number of suggestions for methodologies supporting the development of multiagent simulation models. In this contribution we are introducing a learning-driven methodology that exploits learning techniques for generating suggestions for agent behavior models based on a given environmental model. The output must be human-interpretable. We compare different candidates for learning techniques – classifier systems, neural networks and reinforcement learning – concerning their appropriateness for such a modeling methodology.

1 Motivation

Methodological questions are more and more in the focus of research on agent-based simulation. The central question hereby concerns what behaviors do we have to create on the agent level so that the intended outcome is produced. However, if it is not fully known which local behavior need to be included in the model, designing and implementing the simulation model might be painful and result in some try-and-error process: modifying the local agent behavior, running and analyzing the simulation, followed by modifying the local behavior again. Such a procedure might be feasible for an experienced modeler who knows the critical starting points for behavior modifications, but not so experienced modelers might get lost.

In this contribution we are addressing this search for the appropriate agent-level behavior by using agent learning. The vision is hereby the following procedure: the modeler develops an environmental model as a part of the overall model, determines what the agent might be able to perceive and to manipulate and describes the intended outcome. The agents then use a learning mechanism for determining a behavior program that generates the intended overall outcome in the given environment. In this contribution we are testing three well-known techniques for their suitability in such learning-driven model development process: Learning Classifier Systems, Reinforcement Learning and Neural Networks.

In the next section we will review existing approaches for learning agent techniques in simulation models explaining why we particularly selected these three techniques. This is followed by a more detailed treatment of the learning-driven methodology and a presentation of the candidate techniques. In section 4 and 5 we

J. Dix and C. Witteveen (Eds.): MATES 2010, LNAI 6251, pp. 185–196, 2010.

describe the used testbed and the experiments conducted with it. The results are then discussed. The papers ends with a conclusion and an outlook to future work.

2 Learning Agents and Simulation

Many different forms of learning have shown to be successful when working with agents and multiagent systems. Unfortunately, we can not cover all techniques for agent learning in this paper, the following paragraph shall give a few general pointers and then give a short glance on related work. In general our contribution is special concerning the objective of our comparison: not mere learning performance but its suitability for a usage in a modeling support context.

Reinforcement learning [1], learning automata [2], evolutionary and neural forms of learning are recurrent examples of learning techniques. Besides that, techniques inspired by biological evolution have been applied in the area of Artificial Life [3], where evolutionary elements can be found throughout the multiagent approach. An example of a simulation of a concrete scenario is [4], in which simulated ant agents were controlled by a neural network that was actually designed by a genetic algorithm. Another approach similar to a learning classifier system (LCS) can be found in [5], where a rules set was used and modified by a genetic algorithm. The interesting point in this last case, is that rule conditions are based on situation descriptions.

Related work approaches the behavior modeling task also from the point of view of usability. This can be seen in [6], where an evolutionary algorithm is applied to behavior learning of an individual agent in multiagent robots. Another example, from [7], describes a general approach for automatically programming a behavior-based robot. Using Q-Learning algorithm, new behaviors are learned by trial and error using a performance feedback function as reinforcement. In [8], also using reinforcement learning, agents share their experiences and most frequently simulated behaviors are adopted as a group behavior strategy. Performance is also analyzed for instance in [9], where reinforcement learning and neural networks are compared as learning techniques in an exploration scenario for mobile robots. The authors conclude that learning techniques are able to learn the individual behaviors, sometimes outperforming a hand coded program, and behavior-based architectures speed up reinforcement learning.

3 A Learning-Driven Methodology

The basic idea behind a methodology using a learning-driven approach consists in the transfer of agent behavior design from the human modeler to the simulation system. Specially in complex models, a high number of details can be manipulated. This may make a manual modeling and tuning process cumbersome specially when knowledge about the original system or experience for implicitly bridging the micro-macro gap is missing. Using self-adaptive agents might be a good idea for supporting the modeler in finding an appropriate agent behavior model. Before we continue with candidates, we give a short idea of the basic modeling process for such a learning-driven development.

3.1 Basic Modeling Process

The starting point of this learning-driven modeling process is the environmental model. In [10] we denoted a more general version of this approach as "environment-driven" design is an analysis of the environmental structure. Based on this, the agent interface and its behavior definition are determined. The steps are in particular:

1. *Identify relevant aspects* (global status, global dynamics and local entities);
2. *Determine the primitive actions of the agent and the reaction of the environmental entities to these;*
3. *Determine what information from the environment must be given to an agent;*
4. *Decide on a learning technique* that is apt to connect perceptions and actions of the agent appropriately for actually producing the agents behavior;
5. *Determine the feedback function for the agents.* This reward has to measure performance of the agents in the given environment and is ideally derived from a description of the overall objective or observed aggregate behavior;
6. *Implement the environmental model* including reward function if needed;
7. *Specify and implement the agents* behavior program or agent interfaces in combination with the chosen learning mechanism;
8. *Test and analyze the overall outcome*, the simulation results and individual trajectories carefully for preventing artifacts that come from an improper environmental or reward model or weak interfaces.

After this process ended, ideally a description of the agent behavior is available that fits to the environmental model and the reward given and thus produces the aggregate behavior intended. Analysis of this model – as indicated in the last step – is essential.

There is a variety of possible learning agent techniques that might be suitable for the aim presented here and requirements such as general applicability or accessibility of the resulting model identified. We selected three standard techniques for further examination: Learning Classifier Systems, Q-Learning and a Feed Forward Neural Network, which we will describe in the next paragraphs.

3.2 Learning Classifier Systems: XCS

The accuracy-based learning classifier system XCS is an iterative online learning system [11]. Behavioral knowledge in XCS is represented by a fixed-size population of condition-action-prediction classifiers. Each classifier predicts the consequences (reward) of executing the specified action given that the conditions are satisfied. This basic framework provides means to represent the knowledge in a way that we can clearly identify the agent behavior model - the conditions and the actions. It is possible to determine the quality of a rule based on the predicted reward and the additional evaluation of the accuracy of this prediction. Conditions are represented using three values: true, false and don't care – allowing for generalized situation descriptions with concentration on the relevant aspects.

XCS has a built-in evolutionary rule discovery component. It approximates prediction values by means of credit assignment mechanisms. A successful learning process in XCS, however, requires two conditions to be satisfied: the underlying problem has to be approximately Markov and random exploration during learning should sufficiently ensure complete problem coverage.

3.3 Q-Learning

Q-Learning [12] is a well-known reinforcement learning technique. It works by developing an action-value function that gives the expected utility of taking a specific action in a specific state. The agents keep track of the experienced situation-action pairs by managing the so called Q-Table, that consists of situation descriptions, the actions taken and the corresponding expected prediction, called Q-Value. Q-Learning is able to compare the expected utility of the available actions without requiring a model of the environment. Nevertheless, the use of the Q-Learning algorithm is constrained to a finite number of possible states and actions. As a reinforcement learning algorithm, it also is based on modeling the overall problem as a Markov Decision Process.

3.4 FFNN - Feed Forward Neural Networks

A Feed Forward Neural Network (FFNN) is an artificial neural network where the information moves in only one direction, forward, from the input nodes, through the hidden nodes and to the output nodes [13]. FFNN is usually using supervised training, yet the application situation here is designed for online reward-based learning in a given environment. Therefore, we modified the overall learning process for producing an appropriate setting for the FFNN. There are three phases that are repeatedly executed. The first phase is an explore simulation with randomly selected actions in given situations. These situation-action pairs are recorded with the reward they produced. After a number of steps, a neural network is trained using the best n situation action pairs. The so trained FFNN is then used in a exploit simulation. The rewards of the selected actions during this exploit phase are recorded in the table. After every explore phase, a new FFNN is trained. We had to notice that the influence of the first weak situation action samples was too high, when barely retraining the network. The results of the explore phase are cumulated.

As a FFNN is a black box, the extraction of the behavioral knowledge is non trivial. There are basically two options. During the exploit run, situation and action pairs are recorded and analyzed. Alternatively – if the number of hidden nodes is not too high – the activation state of the different node can be analyzed showing which particular perceived situation elements were responsible for selecting the action.

This is clearly a very restrictive selection of just three techniques that must be extended in future work using other forms of learning such as evolutionary programming support vector machines, other forms of reinforcement learning, respectively learning automata, etc.

4 Testbed

The scenario we use for evaluating the learning approaches is the same as in [14] where we already describe the integration of XCS-based agents into the agent-based modeling and simulation platform SeSAm. This pedestrian evacuation scenario is a typical application domain for multiagent simulation, and albeit the employed scenario may be oversimplified, we expected that its relative simplicity will enable us to evaluate the potentials of each learning technique as well as to deduce the involved challenges.

4.1 Environmental Model

The main objective of the simulation concerned the emergence of collision-free exiting behavior. Therefore, the reward and interfaces to the environment were mainly shaped to support this.

The basic scenario consists of a room (20x30m) surrounded by walls with one exit and a different number of column-type obstacles (with a diameter of 4m). In this room a number of pedestrians have to leave as fast as possible without hurting themselves during collisions. We assume that each pedestrian agent is represented by a circle with 50cm diameter and moves with a speed of 1.5m/sec. One time-step in the discrete simulation corresponds to 0.5sec. Space is continuous. We tested this scenario using 1, 2 and 5 agents, and the number of obstacles was set to 1, 5, and 10. At the beginning of a test-run, all agents were located at given positions in the upper half of the room.

Reward was given to the agent a immediately after executing an action at time-step t. It was computed in the following way:

$reward(a,t) = reward_{exit}(a,t) + reward_{dist}(a,t) + feedback_{collision}(a,t)$ with $reward_{exit}(a,t) = 200$, if agent a has reached the exit in time t, and 0 otherwise; $reward_{dist}(a,t) = \beta \times (d_t(exit,a) - d_{t-1}(exit,a))$ with $\beta = 5$; $feedback_{collision}$ was set to 100 if a collision free actual movement had been made, to 0 if no movement happened, and to -100 if a collision occurred. The different components of the feedback function stress goal-directed collision-free movements.

4.2 Agent Interfaces

As agent interfaces, the perceived situation and the set of possible actions have to be defined. Similar to [14], the perception of the agents is based on their basic orientation, respectively its movement direction. The overall perceivable area is divided into 5 sectors with a distinction between areas in two different distances. For every area two binary perception categories were used. The first encoded whether the exit was perceivable in this area and the second encoded whether an obstacle was present - where an obstacle can be everything with which a collision should be avoided: walls, columns or other pedestrians.

The action set is shaped for supporting the exiting behavior allowing the agent to ignore the navigation task. We assume that the agents are per default oriented towards the exit. Thus, the action set consists of $A = \{move_{left},$

$move_{slightlyLeft}$, $move_{straight}$, $move_{slightlyRight}$, $move_{right}$, *noop*, *stepback*}. For every of these actions, the agent turns by the given direction (e.g. +36 degrees for $move_{slightlyRight}$), makes an atomic step and orients itself towards the exit again. This allows concentrating the learning on the collision avoidance giving the scenario the Markov property.

4.3 Techniques Configuration

The testbed was implemented using SeSAm (`www.simsesam.de`). Due to an existing integration of XCS as an alternative agent architecture [14] the implementation of a XCS agent in the testbed was basically consisting in assigning the perceptions and actions defined in the testbed implementation to bit string elements of the rule description. The Q-Learning could be implemented by means of the standard high-level behavior language in SeSAm. For the FFNN implementation we used the Joone API (`www.joone.org`), integrating the management and usage of neural networks as additional primitive language constructs to the SeSAm modeling language.

The XCS comes with a number of 21 configuration parameters ranging from the size of the rule population, via thresholds for the application of the genetic algorithm or diverse initial values for offsprings to discount factors in multistep mode. As in [14], we did not modify these settings, but used the values of the original implementation of XCS [15] following the advice of its developer. This setting of parameters appeared to be reasonable, but might be discouraging the usage of this learning technique.

The Q-Learning technique assumed an initial Q-Value of 0 for all untested situation-action pairs. Additionally, only two parameters have to be settled: we set the learning rate to 1 and the discount factor to 0.5.

The configuration of the FFNN basically concerned the particular setup of the network itself. We use 20 neurons in the input layer (corresponding to the number of elementary perceptions), 10 neurons in the hidden layer, and 7 neurons in the output layer (corresponding to the number of possible actions). The input layer is a linear layer, and the hidden and output layers are sigmoid layers. After each explore phase, the network for 2000 epochs or until the root mean squared error is lower than 0.01. The training set was consisting of the best rule for each situation presented.

5 Experiments and Results

All experiments alternated between explore and exploit phases. During the explore phase, the agents randomly execute an action. In exploitation trials, the best action according to the used learning technique was selected in each step. Every phase consists of 250 iterations. Every experiment took 100 explore-exploit cycles. In contrast to [14], we did not test a large variety of configurations as it was not our goal to find an optimal one, but a more modeling-oriented evaluation of the different techniques.

Table 1. Mean number of collisions - Columns represent the number of agents and number of obstacles

	1 - 1	2 - 1	5 - 1	1 - 5	2 - 5	5 - 5	1 - 10	2 - 10	5 - 10
XCS	0.59 ±0.79	1.1 ±1.23	7.22 ±2.58	0.56 ±0.84	0.89 ±0.82	6.63 ±3.15	0.1 ±0.32	1.79 ±1.46	7.25 ±3.05
Q-L.	0.22 ±0.5	0.98 ±0.96	8.1 ±3.71	0.75 ±0.78	0.53 ±1	8.17 ±4.61	0.14 ±0.38	1.77 ±1.51	9.19 ±4.94
FFNN	1.02 ±1.79	1.56 ±1.33	8.59 ±3.42	0.77 ±0.95	1.35 ±1.57	9.51 ±4.4	0.95 ±1.19	1.37 ±1.53	7.83 ±4.41

In the following we analyze the results of the simulations, first with respect to learning performance and then concerning the usability of the actually learned behavior control for the proposed methodology.

5.1 Performance Evaluation

The metric used for evaluating learning performance is the number of collisions. As we consider a small room for this evacuation scenario, the time to reach the exit does not vary significantly. A collision is not influencing the behavior directly, but the reward the agent got.

Table 1 presents the mean number of collisions for each agent class, at the end of the simulation of each test case. The values are aggregated only over the last 50 exploit iterations (after warm-up period of 10000 steps) to avoid the inclusion of any warm-up data. Means and standard deviation refer to variation within the single runs. One can see that there is no clear tendency that one technique performs better in all scenarios.

Concerning adaptation speed we could observe in all three learning techniques similar dynamics: the number of collisions decreases fast in the beginning, but then the behavioral knowledge converges quite fast.

To have a better illustration of what this means in each technique, we show in figure 1 the trajectories of the agents in exploit phases after a) 10, b) 50 and c) 100 exploit trials. We can see that, since the early exploit trials, XCS and Q-Learning agents already presents a well defined exit-oriented pattern - with small inefficiencies on the way (Q-Learning). The FFNN agents requires more trials to develop such patterns. This happens because the FFNN agent, when facing a collision situation (e.g. in front of an obstacle), must have experienced by exploration, a good solution (positive reward) to avoid that collision. This is not the same case for the XCS and Q-Learning agents, because even if they don't know what is the best action, they know which one to avoid as they also learn from negatively rewarded actions.

5.2 Behavior Learning Outcome

The second objective of this evaluation is the behavior model as the result of the learning process. We base the following analysis on one randomly selected experiment with 5 agents and 10 obstacles.

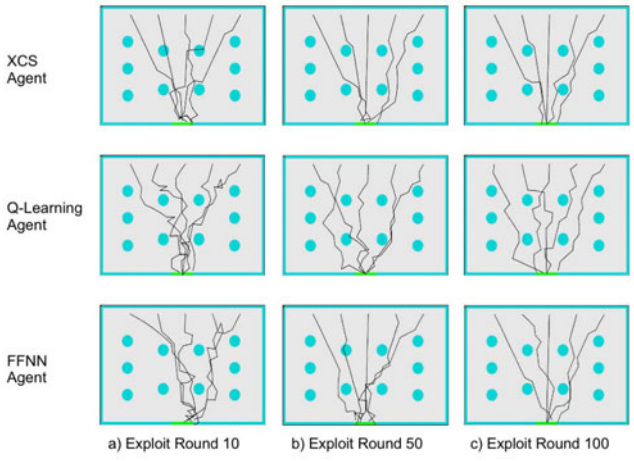


Fig. 1. Exemplary trajectories during exploit trials, for 5 agents and 10 obstacles

XCS Agent. At the end of the experiment, the XCS agent has a rules set with 160 rules, representing his experience after 100 explore and exploit cycles. In this case 71.25% of the rules have a strength value higher than 0, which means that they represent a positive reward experience. Figure 2(a) depicts the strength distribution over the example rule set. Rule strength is hereby a measure of the prediction and fitness, given by: $strength = prediction \times fitness$. Figure 2(b) depicts the experience distribution over the rules set. It presents a small group of experienced rules. This is an effect of the generalization. This small group represents rules that are frequently selected because they generalize the most common situations in this scenario, and therefore are more fitted, which means their reward prediction is more accurate.

Directly considering the rules learned, table 2 outlines the four best XCS rules of an example agent, giving also the fitness and experience information (and the bit string representation of the condition). However, even though the

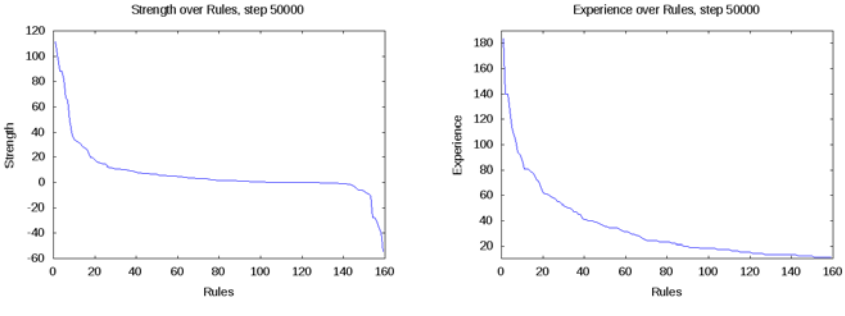


Fig. 2. XCS strenght and experience distribution for an exemplary run with 5 agents and 10 obstacles

Table 2. Best XCS rules for an exemplary run with 5 agents and 10 obstacles (F=Fitness and E=Experience)

Condition (bit string)	Condition Interpretation	Action	F	E
******0******0******	No obst. imdtly. right No obst. near left	$move_{slightlyRight}$	0.71	61
****0*0******0******	No obst. imdtly. ahead or right No obst. near left	$move_{straight}$	0.58	61
00*0*0***0*0*0**	No obst. imdtly. left No exit near left, right or ahead	$move_{slightlyLeft}$	0.56	81
0**0*******00**010**	No obst. or exit left No exit ahead or right Obst. near right	$move_{left}$	0.69	24

generalization has a good impact on the performance, the uncertainties on the reward prediction impact the fitness evaluation. As one can notice from the conditions, the rules are shaped for the particular example agent according to its starting position in the upper right part.

Q-Learning Agent. A Q-Learning agent maintains a table with 1961 situation-action (rule) entries without generalization. In the case with 5 agents and 10 obstacles 44% of the rules either have no experience (they have never been used) or have no reward. Figure 3 shows the distribution of the reward prediction, Q-Value, over the rules set. One can see that there are only a few rules with a high Q-Value. Clearly, the Q-Value alone cannot be a selection criteria for rules forming a behavior model as the ones with the highest Q-Value naturally contain situations where the agent directly perceives the exit. Thus, rules have to be considered for all potentially relevant situations. Although not generalized, the single rules are in principle readable by a human modeler.

FFNN Agent. In the FFNN implementation, the access to the behavior model of the agent is more challenging. There are two possibilities: either analyzing the

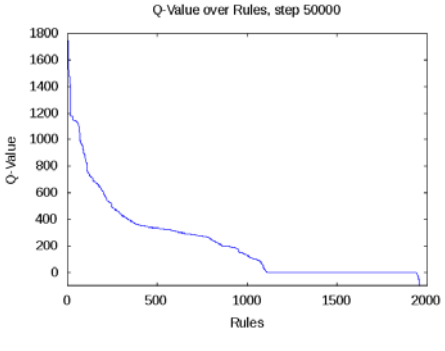
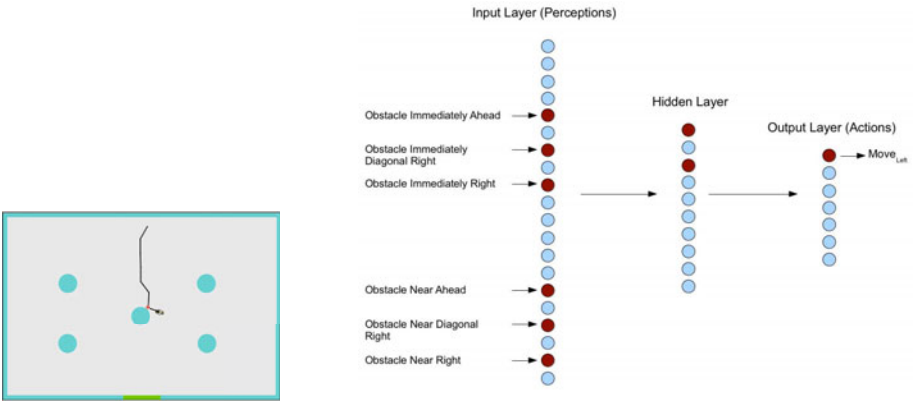


Fig. 3. Q-Learning value distribution for an exemplary run with 5 agents and 10 obstacles



(a) Neural network decision situation

(b) Corresponding Neural network activation

Fig. 4. FFNN exemplary run with 1 agent, 5 obstacles

best rules of final training set – without considering the generalization done by the FFNN or by manually analyzing the activation relations in the net.

The training rules set – in the example containing 45 rules – contains only rule representing a high positive reward which is collected during both explore and exploit phases. In the latter, situation-action combinations used by the FFNN are recorded. Considering the fitness of all rules observed, their fitness is naturally high for all.

More interestedly, yet more effortful to access is the activation state of the network. In figure 4(b), we illustrate the activation for all layer in an example situation. All neurons of one layer are fully connected to the neurons in the next, activation is indicated by (dark) color. This used situation is presented in figure 4(a), where the agent is near the obstacle and can perceive it in the right and front sectors of its perception field. The neural network selects action $move_{left}$.

6 Discussion

The main motivation for this work is investigate the possibilities of creating a learning-based methodology for the design of a multiagent simulation model avoiding a time consuming trial and error process when determining the details of agent behavior. Using a learning technique transfers the basic problem from direct behavior modeling to designing the agent interface and environment reward computation. To do so successfully, a general understanding of scenario difficulties and the available machine learning techniques is necessary.

XCS provides a better interpretability of the rules. For each situation we are able to predict the reward from executing a specific action, how accurate this prediction is (based on the prediction error) and how many times we have executed that action. Combined, these three numbers give relevant information

on quality of rules. Besides that, XCS is able to generalize the representation of the rules based on *don't care* bits in the representation of the perceptions.

Q-Learning showed a good overall performance. It requires less time, which means less explore trials, to learn the possible situations in this scenario. The standard implementation of Q-Learning, used in this paper, offers us only the estimated reward for each possible condition-action pair. This full behavior model for the Q-Learning is only partially helpful as a guidance for modeling.

The Feed Forward Neural Network as supervised learning requires good behavior examples, e.g. a proper set of situation-action pairs to train the network. An issue of our integration into a reward-based setting is that only the best rules were used to train the network. Thus negative experience is lost. The neural network will not learn to avoid actions. However, here the reward contained positive and negative elements thus being not fully appropriate for the FFNN approach.

7 Conclusion and Future Work

In this paper we started our investigation towards a learning-driven methodology by evaluating three well-known learning agent techniques. In a simple evacuation scenario, we showed that all the employed learning techniques can produce plausible behavior without one technique showing the superior performance. Yet, the XCS technique outclasses the two others when it comes to the accessibility and usability of the learned behavior model.

Our next steps include the analysis of more elaborate perception and actions, such as including the distance to the exit or splitting actions into turn and move primitives. Thus, to the collision avoidance task the agents have to learn navigation-related activities. An additional plan is to use the best rules to directly construct a new agent model – supporting the evaluation of techniques and finally we will apply learning-based post-processing techniques for working with the situation-action pairs improving the generality of the rules. Beyond that, we will pursue further self-modeling agent experiments: we are considering the application of these learning techniques in other scenarios, such as an evacuation of a train with about 500 agents, complex geometry with exit signs and time pressure. We are also interested in a scenario where cooperation is required, in order to investigate the possible emergence of the cooperation.

References

1. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge (1998)
2. Nowe, A., Verbeeck, K., Peeters, M.: Learning automata as a basis for multi agent reinforcement learning, pp. 71–85 (2006)
3. Adami, C.: Introduction to artificial life. Springer, New York (1998)
4. Collins, R.J., Jefferson, D.R.: Antfarm: Towards simulated evolution. In: Artificial Life II, pp. 579–601. Addison-Wesley, Reading (1991)

5. Denzinger, J., Fuchs, M.: Experiments in learning prototypical situations for variants of the pursuit game. In: Proceedings on the International Conference on Multi-Agent Systems (ICMAS-1996), pp. 48–55. MIT Press, Cambridge (1995)
6. Maeda, Y.: Simulation for behavior learning of multi-agent robot. Journal of Intelligent and Fuzzy Systems, 53–64 (1998)
7. Mahadevan, S., Connell, J.: Automatic programming of behavior-based robots using reinforcement learning. Artificial Intelligence 55(2-3), 311–365 (1992)
8. Lee, M.R., Kang, E.K.: Learning enabled cooperative agent behavior in an evolutionary and competitive environment. Neural Computing & Applications 15, 124–135 (2006)
9. Neruda, R., Slusny, S., Vidnerova, P.: Performance comparison of relational reinforcement learning and rbf neural networks for small mobile robots. In: Proceedings of FGCNS '08, Washington, DC, USA, pp. 29–32. IEEE Computer Society, Los Alamitos (2008)
10. Klügl, F.: Multiagent simulation model design strategies. In: MAS& S Workshop at MALLOW 2009, CEUR Workshop Proceedings, Turin, Italy, vol. 494 (September 2009)
11. Wilson, S.W.: Classifier fitness based on accuracy. Evolutionary Computation 3(2), 149–175 (1995)
12. Watkins, C.J.C.H., Dayan, P.: Q-learning. Machine Learning 8(3), 279–292 (1992)
13. Mitchell, T.M.: Machine Learning. McGraw-Hill, New York (1997)
14. Klügl, F., Hatko, R., Butz, M.V.: Agent learning instead of behavior implementation for simulations - a case study using classifier systems. In: Bergmann, R., Lindemann, G., Kirn, S., Pěchouček, M. (eds.) MATES 2008. LNCS (LNAI), vol. 5244, pp. 111–122. Springer, Heidelberg (2008)
15. Butz, M.V.: XCSJava 1.0: An implementation of the XCS classifier system in Java. Illigal report, Illinois Genetic Algorithms Laboratory, University of Illinois at Urbana-Champaign (2000)