Generating Inspiration for Multi-Agent Simulation Design by Q-Learning

Robert Junges
Modeling and Simulation Research Center
Örebro University, Sweden
Email: robert.junges@oru.se

Franziska Klügl
Modeling and Simulation Research Center
Örebro University, Sweden
Email: franziska.klugl@oru.se

***Abstract*—One major challenge in developing multi-agent simulations is to find the appropriate agent design that is able generating the intended overall phenomenon respectively dynamics, but does not contain unnecessary details. In this paper we suggest to use agent learning for supporting the development of an agent model: The modeler defines the environmental model and the agent interfaces. Using rewards capturing the intended agent behavior, Reinforcement Learning techniques can be used for learning the rules that are optimally governing the agent behavior. However, for really being useful in a modeling and simulation context, a human modeler must be able to review and understand the outcome of the learning. We propose to use additional forms of learning as post-processing step for supporting the analysis of the learnt model. We test our ideas using a simple evacuation simulation scenario.**

I. Motivation

Methodological questions are more and more in the focus of research on agent-based simulation as the number of challenges in developing a good multi-agent simulation model are numerous. The central issue hereby concerns what behaviors the agents should exhibit so that the intended outcome is generated. What particular detail must be included, what part of the modeled behavior is not necessary? How to set the parameters involved? However, if it is not fully clear from the beginning how this local behavior should be - even if the original agents behavior can be easily observed - the development may result in a painful try and error procedure. The modeler may add, respectively remove behavioral elements, try different parameter values and test the overall outcome again and again. Such a procedure might be feasible for an experienced modeler who knows the critical starting points for modifications and is capable of using complex calibration tools for multi-agent simulation such as described in [1], but this cannot be assumed for less experienced modelers.

In this contribution we are suggesting to solve this search for the appropriate agent-level behavior by using agent learning. The vision is hereby the following procedure: the modeler starts by developing an environmental model as a part of the overall model, then, determines what the agent might be able to perceive and to manipulate and finally describes the intended outcome based on a reward function that evaluates the agents performance. The agents then use a learning mechanism for determining a behavior program that together generates the intended overall outcome in the given environment. This strategy might be also described as a variant of an environment-driven strategy for developing multiagent simulations [2].

A major issue in this overall procedure refers to the selection of the particular learning agent architecture. An initial analysis of different learning techniques applicable for this problem has already been described in [3]. There, Learning Classifier Systems (LCS), Feed Forward Neural Networks (FFNN) and Reinforcement Learning (Q-Learning) have been evaluated with regards to learning performance and resulting behavior representation, using the same evacuation scenario problem as in the following. In this contribution we are further investigating Reinforcement Learning for its suitability in such a learning-driven model development process, focusing more on the interpretability of the state-action mapping produced. We are not focussing on mere optimization performance, but on softer factors that define the usability of Q-Learning in the model development setting: the completeness, the complexity and the generalization capabilities of the behavior learnt.

In the next section we will review existing approaches for learning agent architectures in simulation models. This is followed by a more detailed treatment of the learning-driven methodology and a presentation of the reinforcement learning architecture. In section IV and V we describe the used testbed, the experiments conducted with it and discuss the results. The papers ends with a conclusion and an outlook to future work.

II. Learning Agents and Simulation

Adaptive agents and multi-agent learning have been one of the major focuses within distributed artificial intelligence since its very beginning [4]. Many different forms of learning have shown to be successful when working with agents and multiagent systems. Obviously, 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 directly related work on agent learning in simulation settings. In general our contribution is special concerning the objective of our comparison: not mere learning performance but its suitability for the usage in a modeling support context.

Reinforcement learning [5], learning automata [6], evolutionary and neural forms of learning are recurrent examples of learning techniques applied in multi-agent scenarios. Besides that, techniques inspired by biological evolution have been applied for agents in the area of Artificial Life [7], [8], where evolutionary elements can be found together with multiagent approaches. An example of a simulation of a concrete scenario is [9], in which simulated ant agents were controlled by a neural network that was designed by a genetic algorithm. Another experiment, with an approach similar to a Learning Classifier System (LCS) can be found in [10], where a rules set was used and modified by a genetic algorithm.

Although there is a wealth of publications dealing with the performance of particular learning techniques, especially reinforcement learning approaches, there are not many works focussing on the resulting behavioral model dealing with usability. An early example can be found in [11], where an evolutionary algorithm is applied to behavior learning of an individual agent in multi agent robots. Another example, from [12], describes a general approach for automatically programming a behavior-based robot. Using Q-Learning algorithm, new behaviors are learned by trial and error based on a performance feedback function as reinforcement. In [13], also using reinforcement learning, agents share their experiences and most frequently simulated behaviors are adopted as a group behavior strategy. [14] compares reinforcement learning and neural networks as learning techniques in an exploration scenario for mobile robots. The authors conclude that both learning techniques are able to learn the individual behaviors, sometimes outperforming a hand coded program, and behavior-based architectures speed up reinforcement learning.

III. Agent Learning Architectures for Model Design

The basic idea behind a learning-driven design methodology consists in the transfer of the agent behavior design and test activity from the human modeler to the simulation system. Specially in complex models, a high number of details can be manipulated. This could make a manual modeling, debugging and tuning process cumbersome, especially when knowledge about the original system or experience for implicitly bridging the micro-macro gap is missing. Using agents that learn at least parts or initial versions of their behavior might be a good idea for supporting the modeler in finding an appropriate low level behavior model. Such a learning-based approach can also be part of something as the adoption of a Living Design [15] like methodology for multi-agent simulation models. Nevertheless, the first question on a way to such a learning-driven methodology, is about the selection of the appropriate learning technique – for this form of application, for a particular domain, or maybe just for a particular model. In this paper we focus on the suitability of a well know learning technique, Q-Learning, for such a modeling approach. Before we continue with focussing on this particular learning architecture, we discuss what we have identified as requirements for the applicability of an learning technique to our problem.

A. Requirements for Learning Agent Architectures

Not all agent learning architectures are equally apt for usage in the modeling support context. There are a number of properties that an appropriate learning technique may be able to exhibit for indicating a successful application.

1) *Feasibility*: The learning mechanism should be able to cope with the level of complexity that is required for a valid environmental models. Thus, it should not be necessary to simplify or even to reformulate the problem just for being able to apply the learning mechanism; That means the theoretical prerequisites for applying the learning technology must be known and fulfilled by the environmental model in combination with the reward function. The learning architecture must be able to find a good-enough solution;
2) *Interpretability and Model Accessibility*: The mechanism should produce behavior models that can be understood and interpreted by a human modeler. The architecture shall not be a black box with a behavior that the human modeler has to trust, but must be accessible for detailed analysis of the processes involved in the overall agent system;
3) *Plausibility*: The mechanism in the learning architecture should be well-established and well-understood. The motivation is that its usage shall not impose additional complexity to the modeler for example in setting a number of configuration parameter. How the learning architecture works, shall be explainable to and by the modeler.

There is a variety of possible learning agent architectures that might be suitable for the aim presented here and the requirements identified – as discussed in section II. We selected Q-Learning, as a Reinforcement Learning technique, as we describe it in the next paragraph.

1) Q-Learning: Q-Learning [16] 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 Markov Decision Processes. Thus, it needs sufficient information about the current state of the agent for being able to assign discriminating reward. Although there are a number of extensions that improve the convergence speed of Q-Learning [5], we include the standard Q-Learning algorithms in our experiment due to its simplicity.

We suppose that Q-Learning meets the requirements for the application by providing both sufficient performance (if applicable) adaptability and also gives interpretability of the result. This interpretability is achieved by its rule-based structure (represented by the state action mapping) with a clear evaluation of those rules, by means of the Q-Value. The processing of this mapping, weighted by the provided utility value could be used as a bias for the interpretation of the rules, as an input for the behavior modeling.

IV. Testbed

The scenario we use for evaluating the learning architecture approach is the same as in [17] 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 (see [18] for a real-world application). Albeit the employed scenario may be oversimplified, we expected that the relative simplicity of the scenario will enable us to evaluate the potentials of the learning technique as well as to deduce the involved challenges.

A. 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. In contrast to [17], we did not test a large variety of configurations as it was not the goal of this research to find an optimal one, but a more modeling-oriented evaluation of the architecture.

The basic scenario consists of a room (40x60m) surrounded by walls with one exit and a different number of column-type obstacles (with a diameter of 3.5m). 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, 5, 10 and 20 agents, and the number of obstacles was set to 10. At the beginning of a test-run, all agents were located at random positions in the upper half of the room.

All experiments alternated between explore and exploit phases. During the explore phase, the agents randomly execute an action. In exploitation trials, the best action was selected in each step. Every trial consists of 100 iteration steps. Every experiment took 1000 explore-exploit cycles.

Reward was given to the agent a immediately after executing an action at time-step *t*. It was computed in the following way: $reward(a,t) = reward_{exit}(a,t) + reward_{dist}(a,t) + feedback_{collision}(a,t) + feedback_{damage}(a,t)$ with $reward_{exit}(a,t) = 1000$, 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 $\beta = 5$; $feedback_{collision}(a,t)$ 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; $feedback_{damage}(a,t)$ was set to -1000 if a collision with column obstacle has occurred, and 0 otherwise. Together, the different components of the feedback function stress goal-directed collision-free movements. It is goal-directed because the agents are positively rewarded every time an action results in reaching the exit or getting to a state closer to the exit. Complementary, it is collision-free oriented because the agents are positively rewarded for moving without collisions and negatively rewarded every time an action results in a collision.

B. Agent Interfaces

As agent interfaces, the perceived situation and the set of possible actions have to be defined. Similar to [17], the perception of the agents is based on their basic orientation of the agent, respectively its movement direction. The overall perceivable area is divided into 5 sectors with a distinction between areas in two different distances as depicted in figure 1. 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.

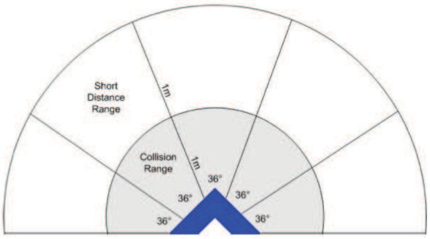


Fig. 1. Agent perception sectors

The action set is shaped for supporting the collision-avoidance behavior. 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 any 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. The combination of this action set and the perceptions of the agents represents an intentional simplification of the problem, as we implicitly represent the orientation task in the actions, in order to have a MDP. This simplification allows concentrating the learning on the collision avoidance, facilitating the learning process.

C. Architecture Configuration

The testbed was implemented in the visual modeling and simulation platform SeSAm (www.simsesam.de). The Q-Learning could be implemented by means of the standard high-level behavior language in SeSAm.

It was not our objective to find the optimal configuration for the tested architecture in the given scenario, we will not give a discussion of the effects of different parameter settings on the learning outcome should not be necessary. Clearly, we tested a number of configuration for finding a reasonable configuration. This is also true for the the appropriate overall configuration including different numbers of obstacles, sizes of scenarios or the particular numbers of the reward function.

In the context of this paper, we assume an initial Q-value of 0 for all untested state-action pairs. We set the learning rate to 0.5 and the discount factor to 0. It means that the agents' actions are selected based on recent experiences and not taking into consideration the future rewards (only the best action for the current state), respectively. This is another intentional simplification for the problem, as the agents don't need to maximize future rewards.

V. Experiments and Results

In this section we analyze the results of the simulations, first with respect to learning performance showing that the learning technique is actually applicable to the test scenario, but then we focus on the analysis of what the agents actually did learn.

A. *Performance Evaluation*

The metric used for evaluating learning performance is the number of collisions. The time to reach the exit does not vary significantly, as a collision is not influencing the behavior directly, but indirectly via the reward the agent got. The collisions, with other pedestrians or obstacles, do not impose any effect on future movement. They only count as negative rewards. Obviously in the early stages, the agents don't have enough experience to learn from, and therefore a higher number of collisions is expected.

Table I presents the mean number of collisions for each tested situation. The values are aggregated only after the first 50 explore-exploit cycles for avoiding the inclusion of any warm-up data. The mean and deviation over the results of the different exploit cycles are given. Despite of having the runs repeated, we did not give means and standard deviations over different runs as currently the number of repetitions is too low. Clearly the number of collisions increases with the number of agents and obstacles.

TABLE I
Mean number of collisions per run - Rows represent the number of agents and column the number of obstacles.

	10
1	0.01 ±0.23
5	1.39 ±1.78
10	6.66 ±3.88
20	25.17 ±8.77

Figure 2 illustrates the adaptation speed by depicting the number of collisions over time for an exemplary run with 5 agents and 10 obstacles. We can see that the number of collisions decreases fast in the beginning, but then the behavioral knowledge converges quite fast. After 50 cycles, there is no further improvement.

To have a better illustration of the learning process, we show in figure 3 the trajectories of the agents in exploit phases after a) 10, b) 100, c) 500 and d) 1000 exploit trials. In this figure we consider the situation with 5 agents and 10 obstacles. We can see the progress of adaptation with more and more collision-free and goal-directed movement. Experience hereby does not just mean positive reinforcement. Even if the agents don't know what is the best action, they know which one to avoid by checking the negative rewarded actions.

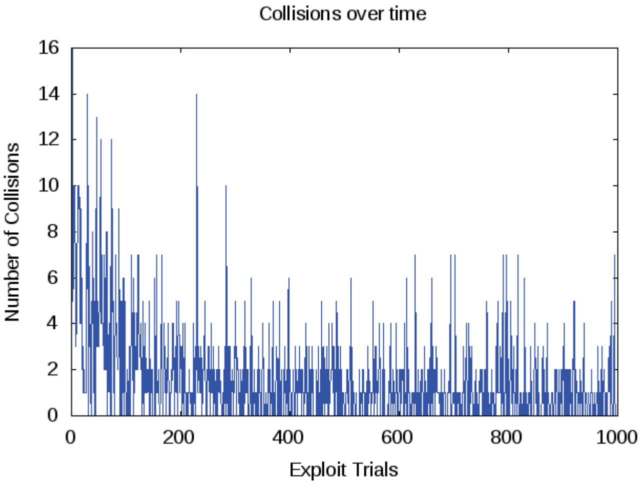


Fig. 2. Development of the number of collisions for an exemplary run with 5 agents and 10 obstacles

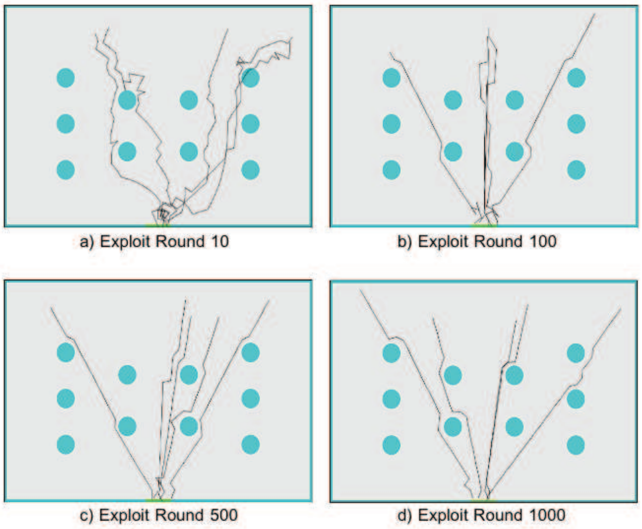


Fig. 3. Exemplary trajectories during exploit trials, for 5 agents and 10 obstacles

Alternating between explore and exploit trials plays an important role in the performance outcome. The agents must explore the possible actions set in order to maximize their experience in terms of the route to be chosen. At the end we can see the emergence of a collision-avoidance behavior.

B. *Behavior Learning Outcome*

In this section we are interested in analyzing the rules learned by the Q-Learning process in terms of the complexity of the resulting rule structure and potential use as source of inspiration in a modeling process.

In the following analysis we will examine two simulation scenarios: 1 agent and 10 obstacles; and 5 agent and 10 obstacles. In both cases we consider the outcome of one agent from an exemplary simulation.

1) Raw Q-Learning Rules: The rules generated by the learning process can be determined by taking for every situation the action with the highest q-value as it is done in the exploit phases. Depending on the situation, there might be no action with a positive q-value. The rules with a Q-Value of zero represent situation-action pairs that have not been tested during the simulation. Figure 4 depicts two out of 12 rules with the highest Q-value on the 1-agent scenarios.

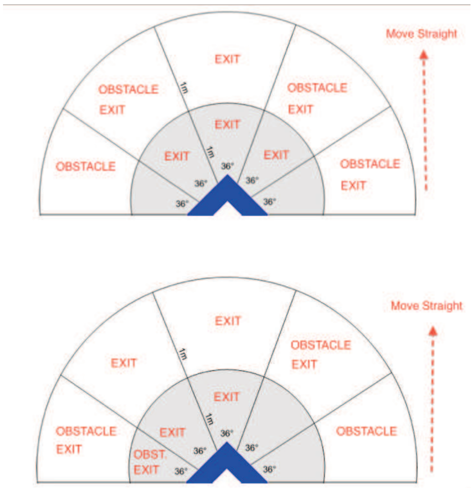


Fig. 4. Two out of 12 rules with the highest Q-value for the agent in the 1-agent scenario.

Figures 5 and 6 show the distribution of the reward prediction, i.e. the Q-value, for the complete rules set for the single agent, respectively a randomly selected exemplary agent from a simulation with 5 agents. One can see that there are only a few rules with a high Q-value.

It is obvious that 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. It is also possible to see that the agent in this case has a majority of rules with Q-Value 0, which means that a lot of state-action mappings have not been tested. This is not case for the simulation with 1 agent and 10 obstacles, as seen in figure 5, where the majority of rules have been tested. The agent has explored more, resulting in a more elaborated representation of the behavior. This difference is caused by the fact that the simulation with only 1 agent presented a smaller set of possible states to be tackled due to the simplicity of the interactions just with static obstacles.

Another important aspect about the agents' experience is that, since the agents are randomly positioned in the scenario at the beginning of each trial, the rules are not biased by a fixed position, so the rules set is more elaborated than it would be if they had to know only one best way to get to the exit.

The agent from the simulation with only one agent has a positive rules set – consisting of rules with positive, non-zero q-values – of 229 rules, while the agent from the simulation with 5 agents has a number of 1507 true positive rules. This can be seen as an effect of the interaction with other agents, generating different situation to be visited, specially when it gets closer to the exit, the situation becomes more dense and the agents must avoid the collisions, and get to the exit.

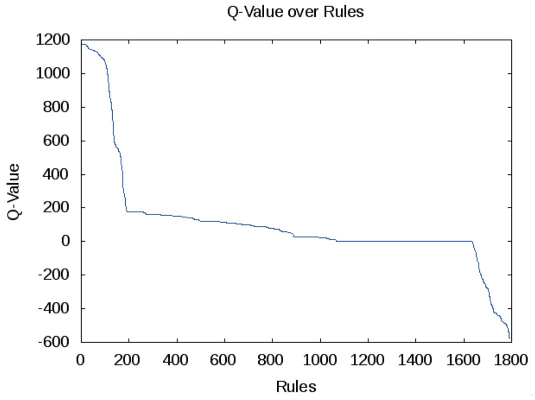


Fig. 5. Q-Learning value distribution for an exemplary agent from a simulation with 1 agents and 10 obstacles

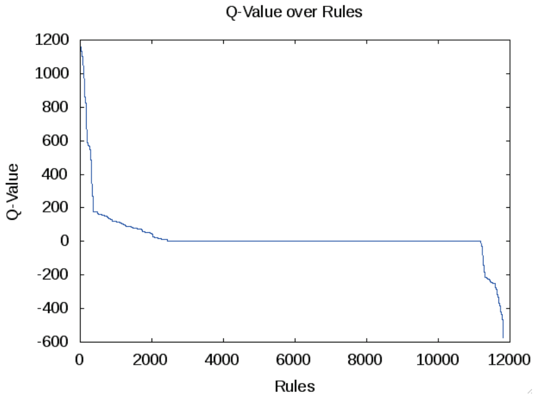


Fig. 6. Q-Learning value distribution for an exemplary agent from a simulation with 5 agents and 10 obstacles

Figures 7 and 8 show the distribution of these final rules over the possible actions, for the cases with 1 and 5 agents respectively. We can see the effect of the initial random positioning in each trial. We have a balanced distribution for the rules determining going to the left or right, which makes sense, since the agent must learn to find its way out of the scenario no matter where it has started. The majority of the rules indicate the $Move_{Straight}$ action. This comes from the fact that the agent is reoriented towards the exit after the execution of any action. Unless the agent needs to avoid a collision, $Move_{Straight}$ is the best action to choose.

We can identify the collision-avoidance behavior focussing on an exemplary element of the perceptions of the agent (1 agent scenario in this case). Considering action $Move_{Right}$ and perception *ObstacleImmediatelyRight*, we see that there is a larger number of rules indicating *false* in this perception in all rules with the $Move_{Right}$ action, see figure 9.

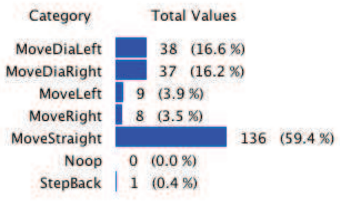


Fig. 7. Rules distribution over the actions for an exemplary agent from a simulation with 1 agent and 10 obstacles

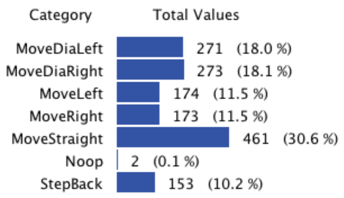


Fig. 8. Rules distribution over the actions for an exemplary agent from a simulation with 5 agents and 10 obstacles

2) Processing the rules: As the set of rules with truly positive Q-value in all scenarios is far too large to be transparently presented to a human expert, we suggest to use a post-processing step for improving the analysis of the rule set on a detailed level. As there are a number of candidates that may be suitable for generalizing the rule set in a way that all learnt rules are captured in a compact form.

For this aim, we tested three different machine learning algorithms – mainly classification learners – using all rules with non-zero, positive Q-Value: K Nearest Neighbors (KNN) [19], CART Decision Trees [20] and the CN2 rule inductor [21]. The K-Nearest Neighbors is arguably one of the simplest machine learning algorithms, while Decision Trees and CN2 are of particular interest to this work because of the interpretability provided by their resulting representation of the knowledge captured in the training set. We used KNN with a K value of 5 for the experiments. The Decision Tree is a simple CART with Gini's index of impurity for node splitting. CN2 algorithm uses the Laplace method for rule quality estimation.

As mentioned above, the results of this post-processing step have to be evaluated with two criteria: How well they capture the given rule set and how good they are able to generalize the rule set for bringing the rules. The first can be measured in terms of classification accuracy, the second is the generalization and compactness of the resulting behavior description.

a) Classification Accuracy: Table II shows the classification accuracy for the above mentioned algorithms, both in the 1 agent and 5 agents experiments, using 10 fold cross validation in the training set. Table III shows the average classification accuracy, when model built from one agent's experience is tested with another agent's experience: We can see that the classification accuracy for the case with 1 agent outperformed the case with 5 agents. This is clearly an effect of the exploit-explore tradeoff. The agent from the 1 agent simulation has a lower number of states to visit during the simulation, and this reflects on the accuracy of the rules as they are tested more times and converge faster to the optimal solution (state-action mapping). The agents from the 5 agents scenario have a larger set of states that potentially may occur, reflected also in the number of rules. This requires more cycles to converge to an optimal solution.

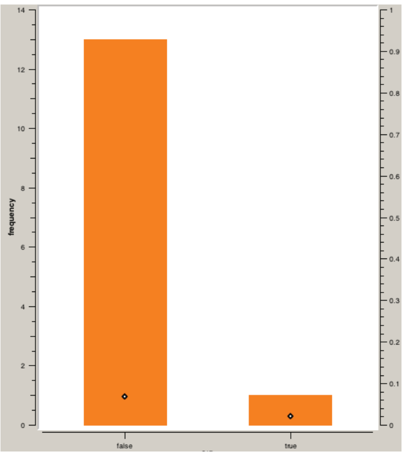


Fig. 9. Frequency of rules with perception *ObstacleImmediatelyRight* as false (left bar) and true (right bar) for action MoveRight

TABLE II
CLASSIFICATION ACCURACY - 10 FOLD CROSS VALIDATION

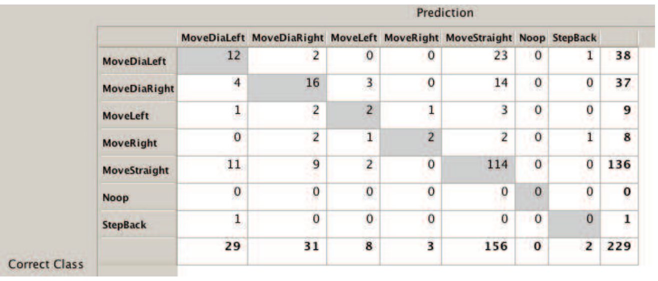
	KNN	Decision Tree	CN2
1 agent	0.6593	0.6375	0.6334
5 agents	0.2907	0.2654	0.2980

TABLE III
CLASSIFICATION ACCURACY - VALIDATION AMONG DIFFERENT AGENTS

	KNN	Decision Tree	CN2
1 agent	0.6724	0.6983	0.6897
5 agents	0.3098	0.3230	0.3316

While they are all good models – as providing a solution to the problem (as seen in section V-A) – they can not be generalized to other good solutions (other agents' experiences). The convergence of the solution, which determines its generalization to the problem is therefore a function of the configuration of the learning, and more important, a function of the explore-exploit distribution, the number of agents and the set of perceptions and actions, that determine the size of the state-action mapping.

Figure 10 shows the confusion matrix for the decision tree learnt from the simulation with 1 agent, testing with cross-validation: Rows represent the expected class (action) from the classification model, as presented in the Q-Learning mapping and columns represent the classification determined by the decision tree. We highlight the number of correctly classified instances. The majority of misclassified instances falls on cases where different actions could result in similar, good rewards. For instance, there is a common misclassification among the actions $Move_{Straight}$, $Move_{SlightlyRight}$ and $Move_{SlightlyLeft}$. This comes from the fact that when the agent is facing the exit, all these three actions will maximize the reward (represented by reaching the exit).



Prediction

Correct Class	MoveDiaLeft	MoveDiaRight	MoveLeft	MoveRight	MoveStraight	Noop	StepBack	
MoveDiaLeft	12	2	0	0	23	0	1	38
MoveDiaRight	4	16	3	0	14	0	0	37
MoveLeft	1	2	2	1	3	0	0	9
MoveRight	0	2	1	2	2	0	1	8
MoveStraight	11	9	2	0	114	0	0	136
Noop	0	0	0	0	0	0	0	0
StepBack	1	0	0	0	0	0	0	1
	29	31	8	3	156	0	2	229

Fig. 10. Confusion matrix for the decision tree in the simulation with 1 agent and 10 obstacles

b) Compactness and Readability of the Learnt Behavior Representation: The second dimension is to be analyzed, with regards to the improving the representation of the behavior for a human modeler. We assumed that the best result could be produced by the decision tree learner. However, the CART decision tree learner was not able to produce an understandable, compact model in this problem. In the case of 1 agent the tree has 117 nodes and 59 leaves. For the case with 5 agents the tree has 1637 nodes and 819 leaves. For illustration, figure 11 outlines a part of the tree generated from the experience of the agent in the case of 1 agent and 10 obstacles. In this figure, the codes represented in the rule stand for different agent perceptions. For instance EIA means *Exit Immediately Ahead*.

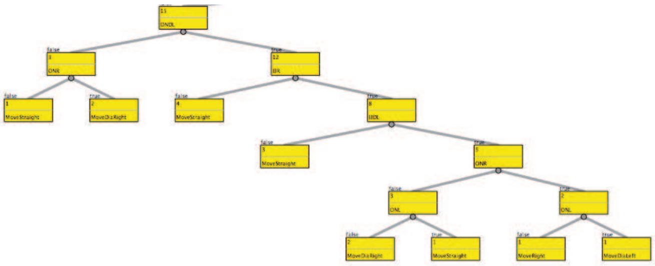
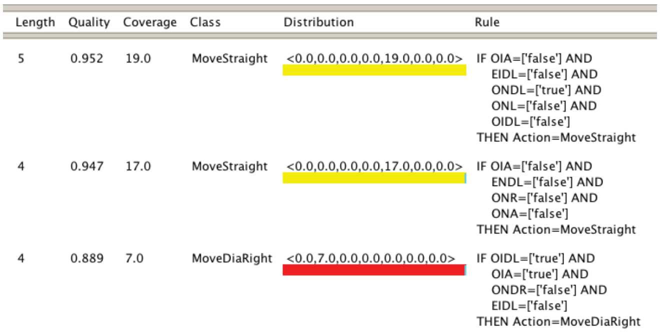


Fig. 11. A branch of the decision tree for the case with 1 agent and 10 obstacles

The post-processing result provided by the CN2 algorithm is better than the decision tree learner: For example, figure 12 shows the best 3 (from a total of 29) rules created by the CN2 algorithm, from the training set of non-zero, positive rules in the case of 1 agent. CN2 was able to reduce the rules representation from 229 to 29 rules. The rules can be evaluated by their quality, size and coverage. Here, as for the decision trees, the perceptions are represented by codes. The rules are clear and concise. Because of that, CN2 can be seen as a step further towards interpretability.



Length	Quality	Coverage	Class	Distribution	Rule
5	0.952	19.0	MoveStraight	<0.0,0.0,0.0,0.0,19.0,0.0,0.0>	IF OIA=['false'] AND EIDL=['false'] AND ONDL=['true'] AND ONL=['false'] AND OIDL=['false'] THEN Action=MoveStraight

4	0.947	17.0	MoveStraight	<0.0,0.0,0.0,0.0,17.0,0.0,0.0>	IF OIA=['false'] AND
					ENDL=['false'] AND
					ONR=['false'] AND
					ONA=['false']
					THEN Action=MoveStraight

4	0.889	7.0	MoveDiaRight	<0.0,7.0,0.0,0.0,0.0,0.0,0.0>	IF OIDL=['true'] AND OIA=['true'] AND ONDR=['false'] AND EIDL=['false'] THEN Action=MoveDiaRight

Fig. 12. CN2 best three rules for the simulation with 1 agent and 10 obstacles

In principle, a set of rules for the agents' producing a solution to the evacuation problem could be learnt – using a technique that results in human-readable rules. However, from these rules that were found by the Q-Learning, we could not construct a behavior representation that fully resembles the knowledge coded in the rule set, nor derive a representation of the rules that a human modeler could easily oversee. On the other side, the scenario is so simple, that it is possible to directly program a set of about 10 rules exhibiting almost optimal behavior.

VI. Conclusion and Future Work

In this paper we presented our investigation towards a learning-driven methodology by evaluating Reinforcement Learning as an agent learning architecture. 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.

In a small evacuation scenario, we showed that the employed learning technique can produce plausible behavior in an agent-based simulation. However, the interface between the learning technique and the agent environment is by no means trivial. The environmental model, feedback function, perception, and action sets are critical. There are also ideas on the analysis of the different architecture that may improve the usability of the learned behavior model.

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. An example is the fundamental requirement of the Markov property in reinforcement-based approaches [5] – in our case Q-learning. Provided perceptions need to contain sufficient information to be able to learn the expectation of immediate and future possible reward accurately.

The standard implementation of Q-Learning, used in this paper, offers us only the estimated reward for each possible condition-action pair. For more intelligent interpretation of the rule set – that is in its raw state without any form of generalization – we decide to use three different machine learning algorithms: K-Nearest Neighbors, Decision Trees and CN2 rule inductor. The resulting, full behavior model for the Q-Learning is only partially helpful as a guidance for modeling in this case. Generalization still needs to be improved, as a part of the learning process or as a post-processing step. This could be achieved by using more flexible classification techniques, such as multi label classification, since in this process we have to deal with multiple good solutions. Another important aspect to be considered here is the tradeoff between explore and exploit, and how this scales to the complexity of the problem, in terms of the number of agents and the size of the state-action mapping in a multiagent simulation. This is a relation yet to be analyzed in detail level.

There are admittedly many more challenging application scenarios than an evacuation scenario where all agents have the same goal, the behavior repertoire is quite restricted, and there is no direct communication between agents. In such advanced environments, the learning and environment design will certainly pose additional challenges.

Our next steps include testing other learning techniques to investigate their performance, outcome and appropriateness for this methodology. A short analysis of Learning Classifier Systems and Neural Networks can be found in [3]. We plan to also test approaches such as evolutionary programming support vector machines, and other forms of reinforcement learning, respectively learning automata. An alternative for the post-processing step worth testing could be multi label classification [22], where we could gather the experience from different agents and find different best actions for a given situation, increasing generalization.

Besides that, we will pursue further self-modeling agent experiments. We are considering the application of the learning technique in other, more complex 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 / collaboration is required, in order to investigate the possible emergence of the cooperation in the agent model, through the learning process. This experimentation should consider situations with and without direct communication between the agents.

References

[1] M. Fehler, Kl´ugl, and F. Puppe, "Approaches for resolving the dilemma between model structure refinement and parameter calibration in agent-based simulations," in *AAMAS '06: Proceedings of the 5th international joint conference on Autonomous agents and multiagent systems*. ACM Press, 2006, pp. 120–122.

[2] F. Klügl, "Multiagent simulation model design strategies," in *MAS& S Workshop at MALLOW 2009, Turin, Italy, Sept. 2009*, ser. CEUR Workshop Proceedings, vol. 494. CEUR-WS.org, 2009.

[3] R. Junges and F. Klügl, "Agent architectures for a learning-driven modeling methodology in multiagent simulation," in *MATES 2010: Proceedings of the 8th German Conference on Multiagent System Technologies (to appear)*, 2010.

[4] G. Weiß, "Adaptation and learning in multi-agent systems: Some remarks and a bibliography," in *IJCAI '95: Proceedings of the Workshop on Adaption and Learning in Multi-Agent Systems*. London, UK: Springer-Verlag, 1996, pp. 1–21.

[5] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[6] A. Nowe, K. Verbeeck, and M. Peeters, "Learning automata as a basis for multi agent reinforcement learning," pp. 71–85, 2006.

[7] C. Adami, *Introduction to artificial life*. New York, NY, USA: Springer-Verlag New York, Inc., 1998.

[8] J. Grefenstette, "The evolution of strategies for multi-agent environments," *Adaptive Behavior*, vol. 1, pp. 65–90, 1987.

[9] R. J. Collins and D. R. Jefferson, "Antfarm: Towards simulated evolution," in *Artificial Life II*. Addison-Wesley, 1991, pp. 579–601.

[10] J. Denzinger and M. Fuchs, "Experiments in learning prototypical situations for variants of the pursuit game," in *In Proceedings on the International Conference on Multi-Agent Systems (ICMAS-1996*. MIT Press, 1995, pp. 48–55.

[11] Y. Maeda, "Simulation for behavior learning of multi-agent robot," *Journal of Intelligent and Fuzzy Systems*, pp. 53–64, 1998.

[12] S. Mahadevan and J. Connell, "Automatic programming of behavior-based robots using reinforcement learning," *Artificial Intelligence*, vol. 55, no. 2-3, pp. 311 – 365, 1992.

[13] M. R. Lee and E.-K. Kang, "Learning enabled cooperative agent behavior in an evolutionary and competitive environment," *Neural Computing & Applications*, vol. 15, pp. 124–135, 2006.

[14] R. Neruda, S. Slusny, and P. Vidnerova, "Performance comparison of relational reinforcement learning and rbf neural networks for small mobile robots," in *FGCNS '08: Proceedings of the 2008 Second International Conference on Future Generation Communication and Networking Symposia*. Washington, DC, USA: IEEE Computer Society, 2008, pp. 29–32.

[15] J.-P. Georg, G. Picard, M.-P. Gleizes, and P. Glize, "Living Design for Open Computational Systems," in *International Workshop on Theory And Practice of Open Computational Systems (TAPOCS) at 12th IEEE International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'03)*, M. Fredriksson, A. Ricci, R. Gustavsson, and A. Omicini, Eds. Linz, Austria: IEEE Computer Society, June 2003, pp. 389–394.

[16] C. J. C. H. Watkins and P. Dayan, "Q-learning," *Machine Learning*, vol. 8, no. 3, pp. 279–292, 1992.

[17] F. Klügl, R. Hatko, and M. V. Butz, "Agent learning instead of behavior implementation for simulations - a case study using classifier systems," in *MATES 2008: Proceedings of the 6th German Conference on Multiagent System Technologies*. Springer Berlin / Heidelberg, 2008, pp. 111–122.

[18] F. Klügl, G. Klubertanz, and G. Rindsfüser, "Agent-based pedestrian simulation of train evacuation integrating environmental data," in *KI 2009: Advances in Artificial Intelligence, 32nd Annual German Conference on AI, Paderborn, Germany, September 15-18, 2009. Proceedings*, ser. Lecture Notes in Computer Science, vol. 5803. Springer, 2009, pp. 631–638.

[19] T. M. Mitchell, *Machine Learning*. New York: McGraw-Hill, 1997.

[20] L. Breiman, J. Friedman, C. J. Stone, and R. A. Olshen, *Classification and Regression Trees*, 1st ed. Chapman and Hall/CRC, January 1984.

[21] P. Clark and T. Niblett, "The cn2 induction algorithm," *MACHINE LEARNING*, vol. 3, no. 4, pp. 261–283, 1989.

[22] G. Tsoumakas and I. Katakis, "Multi-label classification: An overview," *Int J Data Warehousing and Mining*, vol. 2007, pp. 1–13, 2007.