DAInamite 2011 Team Description Paper

Axel Hessler, Martin Berger, and
Holger Endert

DAI-Labor, TU Berlin
Faculty of Electrical Engineering
and Computer Science
{axel.hessler|martin.berger|holger.endert}@dai-labor.de

Abstract. This paper presents some of the recent improvements of the DAInamite team, which was developed by the DAI-Laboratory [1] of the Institute of Technology of Berlin.

1 Introduction

DAInamite has quite a long history in the RoboCup 2D Simulation League. However, due to a personal reorganisation of the developer team, it didn't participate in the last championship in 2010. Therefore, most of the results presented subsequently were developed in the context of courses held at TU Berlin. Nevertheless, we believe that the results are of some value to the robocup community, and that the competitiveness of the team increased such that it is almost ready for the upcoming competition.

DAInamite was implemented without relying on any sourcecode released by others. In the beginning however some ideas where taken from [1]. Our main research focus lies on agent architectures, and in this regard no external sources influenced the development process. The general architecture is already published in previous TDP's. For an overview refer to the DAInamite 2008 TDP [3]. Our current efforts mainly aim at increasing the competitiveness of the team, and to further improve the design of the agent. Furthermore, providing support by means of tools is also of importance. We subsequently present the following sections that highlight our recent developments: First, we describe the integration of reinfrocement learning in our framework from the point of view of software design. Then, we briefly describe our latest tool - a logfile analyser. And finally, we mention our new movement planner based on A*-like algorithms.

2 Reinforcement Learning Framework

Reinforcement Learning (RL) [10] is one of the main approaches when applying machine learning to agent oriented systems, and therefore it is hardly surprising that it is also widely used in robotic soccer. Due to its trial and error nature, RL

[1] http://www.dai-labor.de

applies very well to the simulation leagues (see for instance [4], [8], [7] or [9]), where training runs fast and without the costs of maintaining real robots.

Since our research focus lies on agent architectures, we believe that supporting RL is important. Hence we developed a well designed learning toolkit on the basis of PIQLE [2] and WEKA [5], and integrated it into our agent framework. In the following, we outline the characteristics, which distinguish from work that is found elsewhere.

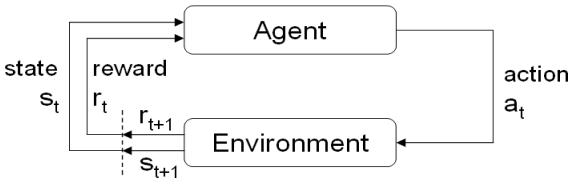

Fig. 1. General reinforcement learning framework, adopted from [10]

In general, RL follows a very simple pattern (see Figure 1): During learning, the agent beliefs to be in state s_t, and performs an action a_t. He observes the successing state s_{t+1} and a numerical reward r_{t+1}. The complete information (e.g. the tuple $<s_t, a_t, s_{t+1}, r_{t+1}>$ in Q-learning [2]) is then used to improve his policy. Once found, the agent can behave on behalf of the (near) optimal policy by choosing the action that will return the highest expected long-term reward for his current state.

Many of the tasks that arise in simulated soccer have to be solved by more than one agent (e.g. every agent needs a dribbling skill), and therefore it is advantageous to a) let them share their experiences during learning, and b) let all of them use the policy found during learning thereafter. A similar approach was taken in [6], where communication was used to share the experiences. We decided to implement this in a service oriented way. The complete functionality is hidden behind an interface that is available to every agent. The interface provides the following operations:

1. **newRLScenario(Environment)**: This operation is used to setup a new RL scenario. A specification of the environment has to be provided as argument (defining how states and actions look like). Each scenario has a unique id, which is returned by this operation on successful creation.
2. **learnFromPast(Id, State, Action, State, Reward)**: This operation is used for learning. The scenario id must be provided as argument.
3. **getAction(Id, State, List<Action>)**: This operation is used to retrieve the next action depending on the policy and the current learning progress. Again, the id must be provided, together with the current state of the agent and a list of available actions.

[2] The correct update rule is slightly different, but can be determined using these values

4. **Configuration Methods**: Most other operations are used to manage the learning tasks, i.e. to change the learning rate, the policy (greedy or, ϵ-greedy, etc.), and other variables.

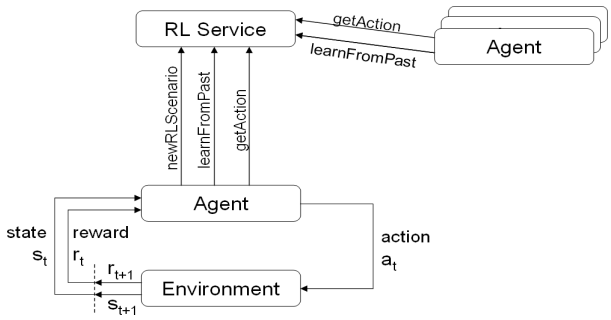

Fig. 2. Reinforcement learning service framework

As can be seen, every scenario is referred to by a unique identifier, which may be shared by a set of agents. This way, an agent can either decide to participate in a scenario by providing learning experiences, use a given trained policy, or setup a complete new learning task. From a very general point of view, this looks like shown in Figure 2, where the agents delegate every RL specific operation to the service. Finally it is noted that the service-oriented approach violates the rules of robocup, since agents are not allowed to have any kind of shared memory. However, we can simply avoid this by assigning a copy of the service to each agent during competitions.

So far, we have implemented Q-learning as algorithm. For value-function representation, we have implemented lookup tables and tile coding by ourselfs, and also made the various classifiers from the WEKA framework available, which provides a variety of function approximators (e.g. neural networks or support vector machines). We used the framework already for teaching, where skills like dribbling and scoring where successfully learned. Possible extensions would be to add other RL algorithms (e.g. SARSA), or to support multi-agent or hierarchical RL.

3 Logfile Analyser

Another new achievement is the implementation of an automated logfile analyser [11], which is capable of generating performance statistics of the teams or of individual players that participated in a logged match. Among other features, the analyser presents statistics on ball-possession, stamina consumption, player and ball trajectories, passes, dribblings and duels. It uses different types of diagrams and charts, and can also draw information directly on our monitor tool.

The main purpose of this tool is to assist the developer in analysing the progress of the team performance. Two example statistics generated from the final match from the 2010 competition are given in Figure 3.

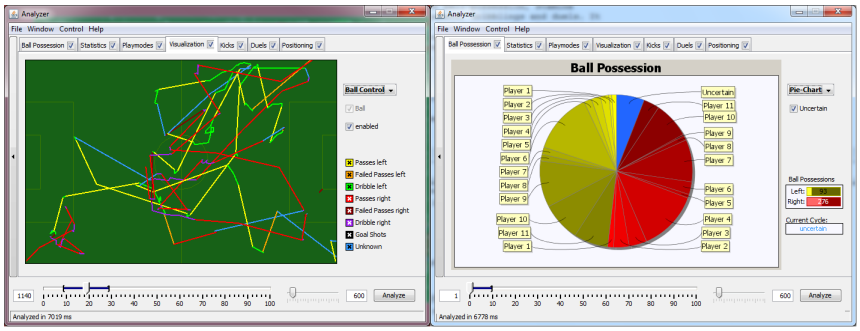

Fig. 3. Screenshot showing two views of the logfile analyer. On the left side, the trajectories of the ball (including how it was controlled) is shown. On the right side a ball-possession statistic is shown. Both were generated from the logfile of the final match of the 2011 championship (left: helios, right: wrighteagle).

4 Planning of Movements

Next, we like to mention our new A^*-like planning algorithm for movements of players. It was developed as a tool that should simplify the calculation of movements in all dash-directions, and also to give accurate estimates on how long a player requires at least to reach a certain position.

Generally, the algorithm consists of two components: One for creating short-term plans of up to 4 cycles ahead exploring all dash-directions, and one for longterm plans of any length, which first turns to the target and then uses forward dashes. Only dashes of maximum power are considered, and dashes that do not accelerate towards the target in at least a fraction are ignored as well. Both components are based on A*, but distinguish in the types of actions that were considered, and in the length of plans that are calculated.

The complete algorithm works as follows: It first decides, which component should be used based on a simple heuristic: If the distance to the target divided by the maximum speed of the player is less or equal to a pre-defined small number (we use the limit of 4), the shortterm planner is used. As not always maximum speed of the player is possible, the shortterm planner may terminate without a result. In this case, or if the aforementioned condition is not satisfied, the longterm planner is used (thereafter). With an average computation time of 0.1ms per target [3], the planner is also applicable for complex computations like

[3] using a 2GHz computer

the determination of the interception point [4]. Interestingly, the computation time is not much higher when computing larger plans, because the longterm planner does not branch, and thus has almost linear complexity. Two examples of generated plans can be seen in Figure 4.

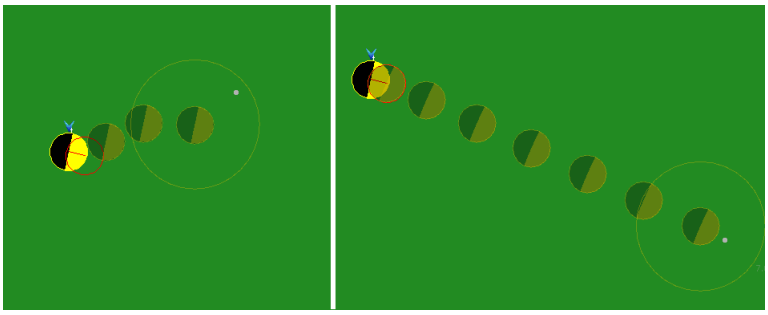

Fig. 4. Screenshot showing the performance of the A* planner. Left: Creation of a shortterm plans using all dash-directions. Right: Creation of a longterm plan by turning first, and then dashing forward (the half transparent players indicate the expected positions in the future when following the plan).

5 Conclusion and Future Work

In this work, we presented some of our recent developments of the DAInamite framework. First, the conceptual integration of a service-oriented RL approach is described, whcih was developed and tested during coursework. Thereafter, a new tool, the logfile analyser, is shown, and finally we sketched our A*-like planner. Each of those features increases the value of our framework by supporting in different stages of the development process. We think that this will lead to better team performance in the long run as well.

In the future, we plan to release the main parts of our framework as open source project, because it might be a good starting point for newbies to the robocup simulation league. W.r.t. the agent design, we plan to better incorporate the action selection for secondary actions (e.g. say or turn-neck), which should seemlessly integrate into our decision tree, and to further develop our tools.

References

1. de Boer, R., Kok, J.: The Incremental Development of a Synthetic Multi-Agent System: TheUvA Trilearn 2001 Robotic Soccer Simulation Team. Master's thesis, University of Amsterdam, The Netherlands (2002)

[4] i.e. the position, where the fastest player can intercept the ball

2. Comite, F.D., Mauch, M.: Piqle (2009), http://mloss.org/software/view/196/
3. Endert, H., Karbe, T., Krahmann, J., an Frank Trollmann, N.K.: The dainamite 2008 team description (2008)
4. Gabel, T., Riedmiller, M., Trost, F.: A Case Study on Improving Defense Behavior in Soccer Simulation 2D: The NeuroHassle Approach. In: L. Iocchi, H. Matsubara, A. Weitzenfeld, C. Zhou (editors), RoboCup 2008: Robot Soccer World Cup XII, LNCS. pp. 61–72. Springer, Suzhou, China (2008)
5. Group, W.M.L.: Weka (2010), http://mloss.org/software/view/16/
6. Kalyanakrishnan, S., Liu, Y., Stone, P.: Half field offense in RoboCup soccer: A multiagent reinforcement learning case study. In: Lakemeyer, G., Sklar, E., Sorenti, D., Takahashi, T. (eds.) RoboCup-2006: Robot Soccer World Cup X. Springer Verlag, Berlin (2007)
7. Riedmiller, M., Withopf, D.: Effective Methods for Reinforcement Learning in Large Multi-AgentDomains. it – Information Technology Journal 47(5), 241–249 (2005)
8. Riedmiller, M., Gabel, T., Hafner, R., Lange, S.: Reinforcement Learning for Robot Soccer. Autonomous Robots 27(1), 55–74 (2009)
9. Stone, P., Sutton, R.S., Kuhlmann, G.: Reinforcement learning for RoboCup-soccer keepaway. Adaptive Behavior (2005)
10. Sutton, R.S., Barto, A.G.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA, USA (1998)
11. Tan, J.: Design und implementierung eines analyse-tools fr die robocup 2d simulationsliga (2010)