Reinforcement Learning and Co-operation in a Simulated Multi-agent System

Kostas Kostiadis and Huosheng Hu

Department of Computer Science, University of Essex
Wivenhoe Park, Colchester CO4 3SQ, United Kingdom

Abstract

The complexity of most multi-agent systems prohibits a hand-coded approach to decision making. In addition to that, a complex, dynamic, adversarial environment like the one of a football game makes decision-making and co-operation even more difficult. This paper addresses these problems by using machine learning techniques and agent technology. By gathering useful experience from earlier stages, an agent can significantly improve its performance. The method used requires no previous knowledge regarding the environment. Since co-operation in adversarial domains is a very challenging task, the proposed learning algorithm assigns each agent a role to play to achieve a certain goal. By distributing the responsibilities among the agents and linking their goals, an efficient way of co-operation emerges.

Key words: Robot World Cup, Multi-agent systems, Reinforcement learning, Cooperation

1. Introduction

The ability to cope with changes in its environment gives an agent a significant advantage, especially if the environment is noisy, complex, and changes over time [4]. In multi-agent environments the need for such adaptability becomes necessary. The first important issue is for an agent to learn from its environment and past experience in order to autonomously operate without the need of human intervention. Another important issue in multi-agent systems is co-operation. It has been shown that groups of agents can derive more efficient solutions in terms of energy, costs, time and quality [5]. The distribution of responsibilities and multiple roles in co-operative frameworks is a common feature. Each agent has an individual role and therefore a set of responsibilities in the team [2][3]. In this paper a form of emergent co-operation through reinforcement learning is presented. In other words, using the back-propagating nature of Q-learning, co-operation is achieved by linking the intermediate local goals each agent is trying to achieve.

Reinforcement learning (RL) addresses the question of how an agent that senses and acts in its environment can learn to choose optimal actions in order to achieve its goals. There are many systems that use RL for learning with little or no a priori knowledge and capability of reactive and adaptive behaviours [1][9][13][16][17]. More specifically, Mataric uses a group of four fully autonomous R2 mobile robots with on-board power and sensing [9]. The learning task consisted of finding a mapping of all conditions and behaviours into the most efficient policy for group foraging. Needless to say, foraging is quite a complex task. Individually, each robot learns to select the best behaviour for each state, in order to find and take home the most pucks. Balch describes an architecture, Clay, for autonomous robots integrating schema-based control and reinforcement learning [1]. Robots using this system can benefit from the real-time performance of motor schemas in continuous and dynamic environments while taking advantage of adaptive RL. Clay co-ordinates assemblages (groups of motor schemas) using embedded RL modules. Learning occurs as the robot chooses assemblages and then samples a reinforcement signal over time. Stone and Veloso introduce a concept of using action-dependent features to generalise the state space, namely TPOT-RL [13]. A learned action-dependent feature space is used to allow a team of agents to learn to co-operate towards the achievement of a specific goal. Uchibe *et al.* describes how they applied vision-based RL on real soccer playing robots [16]. A robot firstly learnt to shoot the ball into a goal given the state space in terms of the size and the positions of both the ball and the goal in the image, then learnt the same task but with the presence of a goalkeeper. Also they describe how RL can be used to obtain optimal behaviours, based on estimated state vectors [17].

The main advantage of reinforcement learning is that it provides a way of programming agents by reward and punishment without needing to specify how the task is to be achieved. On each step of interaction the agent receives an input that normally provides some indication of the current state s of the environment. The agent then chooses an action α to generate as output. The action changes the state of the environment and also provides the agent with a reward of how well it performed. The agent should choose actions that maximise the long-run sum of rewards.

The test-bed used for our research is the soccer simulator [12]. The creators of the robotic soccer define the robot world cup initiative (RoboCup) as an attempt to foster AI and intelligent robotics research by providing a standard problem where wide range of technologies can be integrated and examined [7][8]. Some of the fields covered include multi-agent collaboration, strategy acquisition, real-time planning and reasoning, sensor fusion, strategic decision making, intelligent robot control, and machine learning. Needless to say, given the nature of the environment, machine learning and co-operation become even more important factors for the success of a team in the RoboCup competition.

In section 2, a brief description of the proposed agent architecture is presented. The main problems a reinforcement learner is faced with, and ways to resolve them, are discussed in section 3. Section 4 shows our current implementation of Q-learning in the team. Some initial results are given in section 5 to show the learning agent that can outperform non-learning ones. Finally conclusions are presented in section 6.

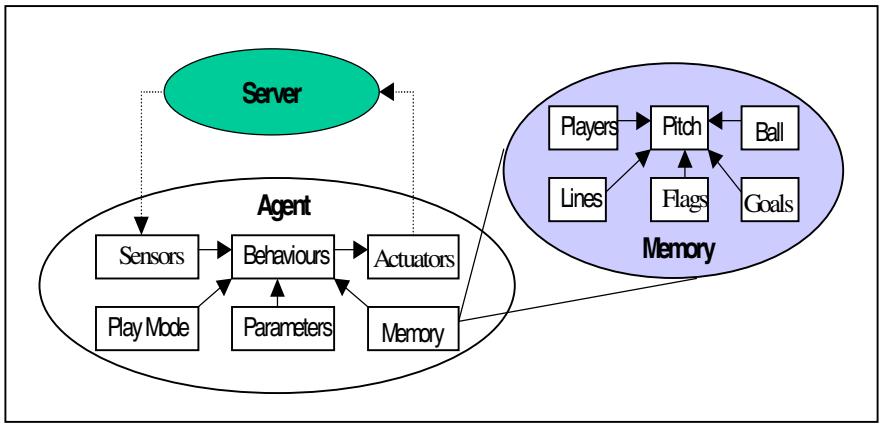


Figure 1 The Proposed Soccer Agent Architecture

2. Agent Architecture

2.1 Modular Design

Before describing how RL has been integrated into the agents, a novel architecture for building RoboCup agents that use RL for decision making is presented in this section. Basically the agent contains six different modules: the agent's sensors, a set of behaviours, the actuators, the current play mode, a set of predefined parameters, and a memory module, as shown in figure 1. The dashed arrows between the agent and the server in this diagram represent the communication links. The server sends information received by the "Sensors" module and the agent sends actions back to the server through its "Actuators" module. The solid arrows give dependence relationships between the modules. More specifically,

- *Sensors* -- are responsible for receiving and analysing the visual or auditory information transmitted by the server. After receiving the data for each cycle, the agent updates only part of the environment in each cycle due to the limited field of view. When new information arrives from the server, the old information is passed to the memory module that holds a probabilistic representation for the whole environment.
- *Behaviours* -- is the most important module within the agent. It is responsible for generating actions according to the current state of the environment. The state of the environment is determined using all the modules within the agent apart from the actuators. The "Behaviours" module should process all the information gathered from other modules, and determine the best course of action.
- *Actuators* -- are responsible for timing, and sending actions to the server at 100ms cycles.
- *Play Mode* -- holds and updates the current play mode using the information received by the "Sensors" module. This module directly affects the behaviour of the agent.
- *Parameters* -- hold the information regarding various settings both for the server and the agent. The parameters affect agent's behaviours since they include information regarding the current formation, the role of the agent in the team, and most of the server parameters including use of offside rule, player's size and maximum speed.
- *Memory* -- is a representation of the whole soccer pitch which contains players, lines, flags, goals and a ball. Each of these objects is associated with a confidence value that represents the agent's confidence that an object is at the current coordinates. The confidence value is 1 if an object was seen in the last cycle. Otherwise, it is multiplied by a confidence decay factor for every cycle that the object has not been seen. When the confidence value falls under a certain threshold the object is "forgotten".

2.2 Learning Module

The core of the decision making process is the "Behaviours" module. However the raw data input from the other modules, including data regarding the agent's position and role in the team, the current formation, the position of the ball and other agents, the current play mode and so on, is inappropriate for the agent to make a decision. Given the nature and size of the environment, it would take numerous iterations before the agent's training can yield acceptable levels of performance. Therefore, the raw information provided by the "Sensors" and other modules, needs to be processed before it can be passed to the RL component of the "Behaviours" module, as shown in figure 2.

As seen earlier the "Sensors" provide information regarding the whole visible part of the environment, and the "Memory" provides a probabilistic representation of the whole environment. However to determine all the possible state-action pairs is not practical and also makes inefficient use of experience.

It is normally expected that similar states have similar optimal actions. The "Filter" component within the "Behaviours" module generalises the input using action dependent features. Generalisation of the state space is a complex task itself. After the input has been processed by the "Filter" component it is passed to the "RL Component". The agent holds a table with a probabilistic representation of all the possible actions it can take given the current state of the environment. Based on these probabilities, the agent chooses an action that generates as output. A reinforcement signal is then used to update the probabilities of execution for each action in the current state, which we will address later.

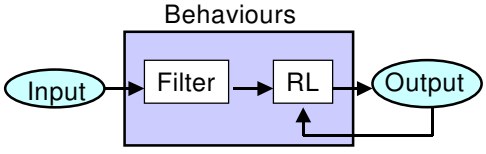


Figure 2 Block Diagram of Learning Module

3. Reinforcement learning

Reinforcement learning differs from the widely studied problem of supervised learning in that the agent is not provided with input/output pairs. Instead, after choosing an action the agent is told how well it performed, but is not told which was the best action. Hence it is necessary for the agent to gather useful experience about system states, actions, transitions and rewards to act optimally.

3.1 Standard RL Model

The standard RL model consists of:

- A discrete set of states, $s_i \in S$
- A discrete set of actions, $\alpha_i \in A$
- A set of reinforcement signals, either *{0, 1}* or the real values $r_i \in R$

An intuitive way to understand the interaction of the agent with its environment is:

$$s_0 \xrightarrow{\alpha_0} r_0 \; s_1 \xrightarrow{\alpha_1} r_1 \; \cdots$$

where $\{s_0, s_1, s_2, \ldots, s_n\}$ is the set of states, and $\{\alpha_0, \alpha_1, \alpha_2, \ldots, \alpha_n\}$ is the set of actions. The real values r_i indicate the immediate reward an agent receives when performing action α_i from state s_i.

Note that apart from the immediate reward an agent receives after each action, there is also a delayed reward arriving from the subsequent (state, action) pairs the agent encounters. For example in the diagram shown above, the agent receives the total reward:

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \ldots \gamma^N r_N = \sum_{i=0}^{N} \gamma^i r_i \qquad (1)$$

where $\gamma \in [0, 1)$ is a constant that determines the relative value of delayed versus immediate rewards, and N is the expected lifetime of the agent. As we will see in the next section N might be unknown or ∞. The agent's task is to learn a control strategy, or policy, $\pi: S \rightarrow A$, which maximises the expected sum of these rewards, with future rewards discounted exponentially by their delay.

3.2 Models of Optimal Behaviour

Before choosing a behaviour that can learn to act optimally, a model of optimality needs to be established. As surveyed in [6], there are 3 models that have been the subject of the majority of the work in this area:

- The finite-horizon model ($N=h$)
- The infinite-horizon discounted model ($N=\infty$)
- The average-reward model ($N \rightarrow \infty$ as $\gamma \rightarrow 1$)

where N and γ are the agent's expected lifetime and reward discount factor respectively.

According to the finite-horizon model, the agent should optimise its expected reward for the next h steps, and should not worry what would happen after that.

$$E\left(\sum_{t=0}^{h} r_t\right) \qquad (2)$$

where r_t represents the reward received t steps into the future. This model has two variations.

In the first, the agent has a non-stationary policy, that is, one that changes over time. On the first step the agent will choose what is termed an h-step optimal action. This is the best action available, given that the agent has h steps remaining in which it can act and gain reinforcement. However on the next step the agent will choose an *(h-1)* optimal action, and so on until it finally takes a 1-step optimal action and terminates.

Secondly, the agent does receding-horizon control, in which it always performs the h-step optimal action. The agent always uses the same policy, but h limits how far ahead the agent looks in choosing its actions. The finite-horizon model is not always appropriate, especially in situations where the precise length of the agent's life is not known in advance. The infinite-horizon-discounted model takes the long-run reward of the agent into account, but rewards received in the future are discounted exponentially by a factor γ, (where $\gamma \in 0, 1$)):

$$E\left(\sum_{t=0}^{\infty} \gamma^t r_t\right) \qquad (3)$$

This model is similar to the receding-horizon control, but the discounted model gives a better mathematical interpretation than the finite-horizon model. This is the reason it has received wide attention and it has been chosen in this project.

Finally, the average-reward model favours actions that optimise the long-run average reward when $h \rightarrow \infty$:

$$\lim \; E\left(\frac{1}{h}\sum_{t=0}^{h} r_t\right) \qquad (4)$$

The main disadvantage of this model is that it can not distinguish between two policies, one that gains a large amount of reward at the initial stages and the other that gains a large amount of reward at the final stages.

3.3 Difficulties in RL

Although RL offers a wide range of advantages, unfortunately a reinforcement learner also has a few problems to overcome. In this section we will describe the main difficulties that arise and ways to tackle them.

Delayed Reward

Delayed rewards pose the temporal credit assignment problem. As we saw earlier in this section, the agent's action determines not only the immediate reward, but also (at least probabilistically) the next state of the environment. Hence the agent must take into account not only the immediate reward, but also the reward it can receive from the future states the current action will lead. The models of optimality described above decide how the future should be taken into account. The agent however must be able to learn an optimal policy based on rewards that can take place arbitrarily far in the future.

Delayed reward is considered to be one of the most important problems in RL. The longer the reward sequences are, the more trials the learning algorithm requires in order to converge. As we will see later on, because RL is a back-propagating method, it is not very efficient for long term goals. However, it can be proven very powerful for near-goal states. By introducing

intermediate goals, through distribution of responsibilities, each agent will only need to achieve a "local" goal depending on its role. For example a goalkeeper need not worry about how to score a goal, which is a task numerous stages away from his normal responsibilities. The goalkeeper's task is to make a successful pass to a nearby defender.

By introducing various roles, and assign different goal states for each role, we distribute the responsibilities between the team members. Although the ultimate goal is to score against the opposition, this is not, and should not be treated as the aim of each individual agent.

Exploration vs. Exploitation

Another important problem is that of exploration versus exploitation. The agent is responsible for the actions it chooses and these actions influence the distribution of training examples. This raises the question of which experimentation strategy produces most effective learning. At any given point an agent faces a trade-off in choosing whether to favour exploration or exploitation. Exploiting a known action guarantees a certain reward, however exploring new actions might result in higher rewards for a given state.

What the agent chooses to do heavily depends on how long it is expected to live for. The longer it lives, the worse the consequences of prematurely converging on a sub-optimal action are. As surveyed by Kaelbling *et al.* [6], there are a number of ways to tackle this problem. We have chosen a *probabilistic assignment* approach to assign each action with a probability of execution. Hence when the agent reaches a state s, it has a set of *actions* $\{\alpha_1, \alpha_2,\ldots, \alpha_n\}$ with corresponding probabilities of execution $\{p_1, p_2,\ldots, p_n\}$. These probabilities are re-calculated after each iteration according to the following formula [8]:

$$P(\alpha_i \mid s) = \frac{k^{Q(s,\alpha_i)}}{\sum_j k^{Q(s,\alpha_j)}} \qquad (5)$$

where $P(\alpha_i|s)$ is the probability that the agent will chose action α_i, given that it is in state s. $Q(s,\alpha)$ represents the agent's hypothesis of the reward value for the (s,α) state-action pair. $k>0$ is a constant that determines how strongly the selection favours actions with high Q values.

As a result, larger values of k will assign higher probabilities to actions with above average Q, causing the agent to exploit what it has already learnt, and seek actions that will maximise its reward. On the other hand small values of k will allow higher probabilities for other actions, leading the agent to explore actions that do not currently have high Q values.

There are two main reasons for choosing this approach. Firstly, this method is very efficient computationally comparing to the methods described above. This is a very important factor since these calculations occur in every cycle. Note that the agent has 100msec to produce an action. Secondly, the property of k can be used to alter dynamically the explorative or exploitative strategy an agent should follow. In the current team, small values of k are used to favour exploration at the early stages of the game, and larger values to favour exploitation as the game progresses. In this way the agent does not over-commit to sub-optimal actions at early stages, and does not carry on exploring when near-optimal actions have been found.

Partially Observable States

Although it is convenient to assume that the agent's sensors can perceive the entire state of the environment at each time step, this is not the case in most real-life problems, and it is not the case in the RoboCup domain. The sensors provide only partial information due to the limited field of view that each agent has. In such cases, the agent might have to combine previous information together with new sensory data when choosing an action. Also in some cases the best policy might be one that specifically chooses actions that improve the observability of the environment. There are a number of policies that can help an agent resolve the partial observability of a state. They are mainly categorised in state-free deterministic policies, state-free stochastic policies, and policies with internal state

For the purposes of this project policies with internal state are more suitable. That is mainly because they rely on the fact that the only way for an agent to behave effectively is to use memory of previous actions and observations to disambiguate the current state. Therefore a memory model to tackle the problem of partial observability is used to create a probabilistic representation of the current state based on previous observations. Each object on the field is assigned a probability of existence for its present location, which changes when the object appears or disappears.

4. Implementing RL

4.1 The Learning Task

At each discrete time step t, the agent senses the current state s_t, and chooses an action α_t to perform. The environment responds by rewarding the agent $r_t = r(s_t,\alpha_t)$, and producing the succeeding state $s_{t+1} = \delta(s_t,\alpha_t)$. It is important to notice that the functions r and δ are not known to the agent.

The agent's task is to learn a policy $\pi: S \rightarrow A$, for selecting actions based on the current state. Needless to say, the agent should look for a policy that will maximise the discounted cumulative reward achieved from an initial state s. In other words, the agent's learning task is to find a policy π, which will maximise the discounted cumulative reward for all states. Such a policy is called an optimal policy, and is denoted by π^*.

4.2 The Q function

It is difficult for an agent to learn π^* directly given the available training data. Let us remember that the agent is not provided with examples of the form (s,α). The only training information is a sequence of immediate rewards. Given this kind of training information it would be much easier to learn a numerical evaluation function defined over states and actions, and then implement the optimal policy in terms of this evaluation function.

Such an evaluation function is $Q(s,\alpha)$ which is defined so that its value is the maximum discounted cumulative reward that can be achieved starting from state s and performing action α. The main advantage of Q-learning lies in the fact that the value of Q for each state-action pair, summarises in a single number all the information needed by the agent in order to determine the discounted cumulative reward that will be gained in the future. This means that an optimal action can be chosen at each state without ever conducting a look-ahead search to determine what the precise results from that action are going to be. Moreover the agent is now able to select optimal actions even if it has no knowledge of the reward and state transition functions r and δ.

4.3 Training the Agent

Assume that the agent can store a table with a unique Q value for every state-action pair. Also assume that at the beginning of the training all table entries are initialised to 0. We need to find a reliable way to estimate the values of Q for each entry in the table given only a sequence of immediate rewards.

After the agent has identified the current state and has chosen an action to execute, it observes the reward and the following state. Since in uncertain environments the reward and the state transition functions have probabilistic outcomes, the agent can only probabilistically estimate those values. A training rule based on the immediate reward and successive state would fail to converge.

To overcome this problem the agent should take into account this non-deterministic outcome for actions. This can be done by taking the expected value of the immediate reward, and define the successive state probabilistically. So our training rule takes the decaying weighted average of the current Q value and the revised estimate. The main idea behind such a training rule is that revisions to Q are made more gradually. In other words, updates to each Q table entry become smaller and smaller as the training progresses. Given the right training rule, convergence can be guaranteed as proven by [18].

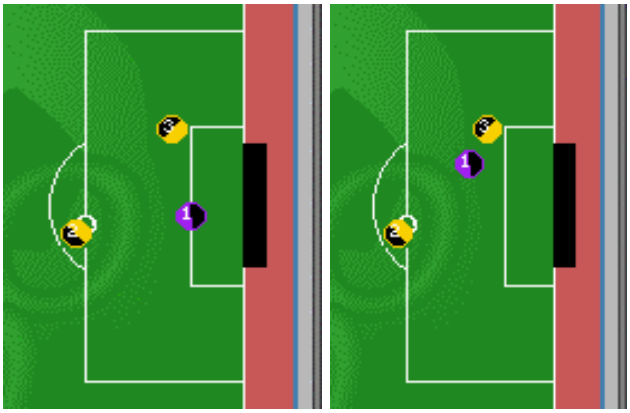
4.4 Generalisation of the State Space

The previous discussion has assumed that it is possible to determine all the possible state-action pairs, and store a table with all the Q values over them. However, this is not practical and also makes inefficient use of experience. We normally expect similar states to have similar optimal actions. There should be a more compact representation than a table with all possible state-action pairs.

We have chosen a form of representation based on the objects an agent needs to see in order to achieve its goal. Low level filtering removes the infeasible actions from the set of possible actions the agent can choose from. In other words, if an agent cannot see any teammates there is no need to consider passing. Following this, the agent extracts only the necessary features needed to accomplish its goal. The partitioning of the state space uses objects like the enemy goal, the existence of enemy or friendly players, etc. For instance a goalkeeper does not need to worry about where the enemy goal is. Such input is irrelevant and should not affect its action selection since its individual local goal is to pass the ball to a defender. This form of generalisation provides a very compact representation of the large state space in the RoboCup domain.

5. Initial Results

To evaluate the performance of a Q-learning agent the "2 attackers versus 1 defender" problem was used, shown in figures 3. The two attackers are in front of the enemy goal and one defender is trying to stop them from scoring.



(a) Defender blocking shot (b) Defender blocking pass

Figure 3 Implementation of a Learning Agent

The defender (player 1) is randomly placed either blocking a direct shot in Fig. 3(a) or blocking the pass between the two attackers in Fig. 3(b). Depending on the defender's position, the attacking player in possession (player 2) is faced with the dilemma:

- He can shoot and score directly
- He can pass the ball so his team-mate can score

The correct solution to the above problem is unique and is dependent upon the defender's position. If the defender is blocking a direct shot, a pass should be executed, see fig. 3(a). On the other hand if the defender is blocking the pass, a direct shot should be preferred, see Figure 3(b),

To measure the raw performance of this model, certain assumptions had to be made. For reasons of simplicity, two states and two actions are sufficient to fully represent the environment described above. The first state is the one where the defender is blocking the shot, and the second state is the one where the defender is blocking the pass. In each state the agent can choose between two actions, either shoot or pass. Noise is removed from the ball's movement and no offside rule is used. Finally, since there are only 2 actions, they are not assigned probabilities of execution as described earlier. The action that is executed is the one estimated as optimal.

Initially a "random attacker" was used, i.e. the choice between a shot and a pass is completely random and is irrelevant of the defender's position. The results of 141 trials are shown in table 1. Horizontally the table indicates the number of successful, failed, and total number of trials. Vertically the table shows the number of passing, shooting, and total attempts.

Table 1: Results for random striker

Random striker	Pass	Shoot	Total
Success	34 (55.74%)	37 (46.25%)	71 (50.35%)
Failure	27 (44.26%)	43 (53.75%)	70 (49.65%)
Total	61	80	141

For example, 34 in entry 2-2 indicates the number of times the agent has successfully chosen to pass the ball, i.e. when the defender was blocking a direct shot. Similarly, the entry below indicates the times the agent has incorrectly chosen to pass the ball when the defender was blocking the pass.

Following the "random striker", a "Q-learning striker" was used, which tries to learn the optimal action depending on the state it currently is. The results of 120 trials are shown in table 2.

Table 2: Results for Q-Learning striker

Q-Learning Striker	Pass	Shoot	Total
Success	52 (100%)	67 (98.53%)	119 (99.17%)
Failure	0 (0%)	1 (1.47%)	1 (0.83%)
Total	52	68	120

The reason for the extremely high success rate in the choices made lies in the back-propagating nature of Q learning. Initially the learner's Q' hypothesis about all state-action pairs is set to zero. Since both actions have the same Q value (i.e. zero) the agent randomly chooses an action and executes it. If it is the wrong action, no reward is received so at the next trial it again chooses randomly. However if a successful choice is made, the reward that is received makes this action optimal thereafter. Hence, once the agent finds the correct action at each state, it then acts with 100% accuracy. In other words, once the goal-state has been reached, the learning propagates backward to the previous states and so on. It is worth mentioning that for exactly the same problem, experiments carried out in [11] using a back-propagating neural network yield a success rate of 68%.

6. Conclusions

Due to the complexity of the environment and the size of the state space in a robotic football game, machine learning techniques have a key role to play in such a domain. This paper has shown how RL was implemented and how the common problems in machine learning (i.e. exploration versus exploitation, partially observable states, etc.) have been tackled. Experiments revealed that an agent that learns to choose optimal actions significantly outperforms a non-learning agent.

Our approach introduces agents that not only have different responsibilities, but also have different roles in the team. In other words, although the overall goal of the team is to score against the opposition, each agent's local goal is different. Hence, every individual agent tries to reach its own individual goal state without worrying about the goals of the other agents. By linking the different goals of each agent, co-operation emerges. Although each agent tries to optimise its actions and reach its own goal-state, since these goal-states are related, the agents co-operate. The ultimate goal, which is to score against the opposition, becomes a joint effort that is distributed between the members of the team.

Acknowledgements

We would like to thank the University of Essex for the financial support to the project by providing the Research Promotion Fund DDP940.

References

1. Balch T. R., Integrating RL and Behaviour-based Control for Soccer. Proc. IJCAI Workshop on RoboCup, 1997.
2. Ch'ng S. and Padgham L., From Roles to Teamwork: A Framework and Architecture. Applied Artificial Intelligence Journal, 1997.
3. Ch'ng S., Padgham L., Team description: Building Teams Using Roles, Responsibilities, and Strategies. Proc. IJCAI-97 on RoboCup, 1997.
4. Hu H., Gu D., Brady M., A Modular Computing Architecture for Autonomous Robots, Int. Journal of Microprocessors and Microsystems, Vol. 21, No. 6, pages 349-362, March 1998
5. Hu H., Kelly I., Keating D., Vinagre D., Coordination of Multiple Mobile Robots via Communication, Proc. SPIE'98, Mobile Robots XIII Conference, Boston, pages 94-103, Nov. 1998
6. Kaelbling P. Leslie, Littman L. Michael, Moore W. Andrew, Reinforcement Learning: a survey. Journal of Artificial Intelligence 4, pp. 237-285, 1996.
7. Kitano H., RoboCup: The Robot World Cup Initiative. In proceedings of The First Int. Conf. on Autonomous Agent (Agents-97)), Marina del Ray, The ACM Press, 1997.
8. Kitano H., Tambe M., Stone P., Veloso M., Coradeschi S., Osawa E., Matsubara H., Noda I., and Asada M., The RoboCup Synthetic Agent Challenge, 97. International Joint Conference on Artificial Intelligence (IJCAI97), 1997.
9. Mataric, J. M., Interaction and Intelligent Behaviour. PhD Thesis, Massachusetts Institute of Technology, 1994.
10. Mitchell M. Tom, Machine Learning. McGraw-Hill International Editions, Singapore, pp. 367-388, 1997.
11. Noda Itsuki, Matsubara Hitoshi, Hiraki Kazuo, Learning Co-operative Behaviour in Multi-Agent Environment – A Case Study of Choice of Play Plans in Soccer. PRICAI'96: Topics in Artificial Intelligence (Proceedings of 4th Pacific Rim International Conference on Artificial Intelligence, Cairns, Australia), pp. 570--579, Aug. 1996.
12. Noda Itsuki and Soccer Server: A Simulator for RoboCup. JSAI AI-Symp.: Special Session on RoboCup, Dec. 1995.
13. Stone Peter, Veloso Manuela, Team-Partitioned, Opaque-Transition Reinforcement Learning. Proc. 15th Int. Conf. on Machine Learning, 1998.
14. Tambe M., Towards Flexible Teamwork. Journal of Artificial Intelligence Research, Volume 7, Pages 83-124 On-line appendix for this article: STEAM Rules and Documentation, 1997.
15. Tambe M., Adibi J., Alonaizon Y., Erdem A., Kaminka G., Marsella S., Muslea I., Tallis M., ISIS: Using an Explicit Model of Teamwork in RoboCup '97. Proceedings of RoboCup'97, Springer Verlag, 1998.
16. Uchibe E., Asada M., Hosoda K., *Strategy Classification in Multi-Agent Environment – Applying Reinforcement Learning to Soccer Agents.* ICMASS'96 workshop 2: RoboCup Workshop: Soccer as a Problem for Multi-Agent Systems, 1996.
17. Uchibe E., Asada M., Noda S., Takahashi Y., Hosoda K., Vision-Based Reinforcement Learning for RoboCup: Towards Real Robot Competition. Proc. of IROS 96 Workshop on RoboCup, 1996.
18. Watkins, C., & Dayan, P., Q-learning. Machine learning, 8, pages 279-292. Kluwer Academic Publishers, Boston, 1992.