Time-based Reward Shaping in Real-Time Strategy Games

Martin Midtgaard, Lars Vinther, Jeppe R. Christiansen,
Allan M. Christensen, and Yifeng Zeng

Aalborg University, Denmark
{initram,larskv,jepperc,allanmc,yfzeng}@cs.aau.dk

Abstract. Real-Time Strategy (RTS) is a challenging domain for AI, since it involves not only a large state space, but also dynamic actions that agents execute concurrently. This problem cannot be optimally solved through general Q-learning techniques, so we propose a solution using a Semi Markov Decision Process (SMDP). We present a time-based reward shaping technique, TRS, to speed up the learning process in reinforcement learning. Especially, we show that our technique preserves the solution optimality for some SMDP problems. We evaluate the performance of our method in the Spring game Balanced Annihilation, and provide some benchmarks showing the performance of our approach.

1 Introduction

Reinforcement learning (RL) is an interesting concept in game AI development since it allows an agent to learn by trial-and-error by receiving feedback from the environment [2]. Learning can be done without a complete model of the environment which means that RL can be easily applied to even complex domains. RL has been applied to many types of problems, including many different board games [3], a simple soccer game [4] and robot navigation [5]. In the context of RTS games, RL has been applied to a small resource gathering task [7], and a commercial real-time strategy game (RTS) called MadRTS [6].

The benefit of applying RL is to create adaptive agents (Non-player characters in computer games) that may change their behaviour according to the way opponents play throughout games. Generally, RL requires that the problem shall be formulated as a Markov decision process. This demands that environmental states, as well as actions, must be well defined in the studied domain. However, a complex and dynamic RTS game always involves a very large state-action space. Moreover, agents' actions may last several time steps and exhibit a dynamic influence on the states. Both of these problems prevent a fast convergence to an optimal policy in RTS games. Recently, Kresten *et al.* [10] showed that the hierarchical decomposition of game states may mitigate the dimensional problem. This paper will investigate solutions to the second problem and speed up the convergence using relevant techniques.

We resort to Semi Markov Decision Processes (SMDPs) that extend MDPs for a more general problem formulation. SMDPs allow one single action to span

multiple time steps and change environmental states during the action period. This property matches well with solutions to the dynamic actions within RTS games of our interest. However, the problem of slow convergence occurs when an optimal policy needs to be compiled in a short period of gameplay.

To speed up the learning process, we present a time-based reward shaping technique, TRS, which makes it possible to apply the standard Q-learning to some SMDPs and to solve SMDPs in a fast way. We let the agent get a reward only after it completes an action, and the reward depends on how much time it takes to complete the action. This means that after many trials, the agent will converge towards the fastest way of completing the scenario, since all other solutions than the fastest possible one will result in a larger negative reward. Our proposed approach puts some constraints on the properties of the SMDP problems on which time-based reward shaping can be applied without compromising the solution optimality.

We compare TRS to the Q-learning algorithm SMDPQ presented by Sutton *et al* [1]. SMDPQ extends Q-learning to support SMDPs by changing the update rule. We theoretically analyse the equivalence of optimal policies computed by both SMDPQ and our proposed method, TRS, given the constraints. We evaluate performance of TRS on a scenario in the RTS game of Balanced Annihilation[1] and show a significant improvement on the convergence of SMDPQ solutions. We also apply TRS to other RL methods such as Q-learning with eligibility traces, Q(λ), and show that this also can be successfully extended to support some SMDP problems.

The rest of this paper is organized as follows. Section 2 describes related work on using RL in computer games. Section 3 discusses some background knowledge. Subsequently, Section 4 presents our proposed method, TRS, and analyse the policy equivalence. Section 5 provides experimental results to evaluate the method. Finally, Section 6 concludes our discussion and provides interesting remarks for future work.

2 Related Work

RL methods have been well studied in the machine learning area where much of the work focuses on the theoretical aspect of either the performance improvement or the extension from a single-agent case to a multi-agent case. Good survey papers can be found in [11]. Although RL techniques have been demonstrated successfully in a classical board game [12], computer games are just recently starting to follow this path [13].

RTS games are very complex games, often with very large state- and action-spaces and thus when applying RL to these problems we need ways to speed up learning in order to make the agent converge to an optimal solution in reasonable time. For example, some methods like backtracking, eligibility traces [2], or reward shaping [9] have been proposed for this purpose. Laud demonstrates[9]

[1] `http://springrts.com/wiki/Balanced_Annihilation`

that reward shaping allows much faster convergence in RL because the reward horizon is greatly decreased when using reward shaping, i.e. the time that passes before the agent gets some useful feedback from the environment is decreased. Meanwhile, Ng *et al.* [8] prove if the reward shaping function takes the form of the difference of potentials between two states the policy optimality is preserved.

Recently, Marthi *et al.* made a hierarchical concurrent approach to a small gathering task in an RTS game [7]. This is a very basic scenario only constituting a small part of an entire RTS game, but it is a clear proof that a hierarchical division of an RTS game may very well be possible. This means that in some cases we are able to identify isolated subtasks which we want to solve as fast as possible, so we will be able to apply time-based reward shaping to achieve this goal.

The Q-learning variant SMDPQ[1] is used for proving policy equivalence with TRS on the supported SMDP problems.

3 Background

In this section we will describe some of the prerequisites needed for discussing reward shaping in problems with non-discrete time. We will cover Semi Markov Decision Processes (SMDPs) and Q-learning (for MDPs). This forms the basis for our proposal on time-based reward shaping.

3.1 SMDP

An SMDP is a simple generalisation over an MDP where an action does not necessarily take a single time step, but can span several time steps. The state of the environment can change multiple times from when an action has been initiated until the next decision is initiated. Rewards are also slightly different from those of standard MDP, as the decision maker will both receive a lump sum reward for taking an action and also continuous rewards from the different states entered during the decision epoch. This all means that SMDPs can describe more general scenarios and environments than MDPs [1].

Formally an SMDP can be described as the 5-tuple (S, A, T, F, R), where:

- S : the finite state set
- A : the finite set of actions
- T : the transition function defined by the probability distribution $T(s'|s, a)$
- F : the probability of transition time for each state-action pair defined by $F(s', \tau|s, a)$
- R : the reward function defined by $R(s'|s, a)$

The F function specifies the probability that action a will terminate in s' after τ timesteps when starting from s.

3.2 Q-Learning

Q-learning is an algorithm for finding an optimal policy for a MDP (not SMDP). It does so by learning an action-value function (or Q-function) that returns the expected future reward of taking an action in a specific state. Q-learning does not need a model of its environment to be able to learn this function. The Q-function is a mapping from a state and an action to a real number, which is the expected future reward: $Q : S \times A \mapsto \mathbb{R}$.

After the Q-function has been learned, it is trivial to find the optimal policy, as one simply has to choose the action with the highest return from the Q-function in a given state.

Learning the Q-function is done using an action-value, and with an update rule shown in Eq. 1. In addition, the algorithm chooses its actions in an ϵ-greedy manner. It has been shown that by using this update rule the algorithm converges to Q^*, the optimal value function, and using this to choose actions will result in the optimal policy, π^* [2].

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t)] \tag{1}$$

Q-learning is called an off-policy algorithm as it does not use its policy to update the Q-function, but instead uses the best action given its current Q-function. This however is not necessarily its policy, as the algorithm will have to perform exploration steps from time to time. These choices are however needed for the algorithm to converge to an optimal solution as it will have to take all actions in all states to be sure that it converges to the optimal policy.

Sutton *et al* [1] proposed an algorithm, SMDPQ, which extends Q-learning to support SMDPs by changing the update rule. Normally the discount factor, γ, is multiplied with the expected future reward, as all actions in MDP problems are assumed to take constant time to complete. In the modified version, the discount factor is raised to the power of the number of time steps that the action took to complete. This results in an update in Eq. 2.

$$[Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha(R_t + \gamma^k \max_a Q(s_{t+1}, a) - Q(s_t, a_t))] \tag{2}$$

3.3 Reward Shaping

Ng *et al.* [8] proposed reward shaping to speed up Q-learning while preserving the policy optimality. Every optimal policy in an MDP, which does not use reward shaping will also be an optimal policy in an MDP which uses reward shaping, and vice versa. To achieve this, we need a function $\Phi(s)$ that can calculate a value based on state s, which should represent the potential of the state, making it comparable to another state. The potential-based shaping function is formally defined as:

$$F(s, a, s') = \gamma\Phi(s') - \Phi(s) \tag{3}$$

An example of such a function could be the euclidean distance to a goal from a current position in a game world, as negative. As a rule of thumb, $\Phi(s) = V_M^*(s)$, where $V_M^*(s)$ means the value of the optimal policy for MDP M starting in state s, might be a good shaping potential.

4 Time-based Reward Shaping

We perceive that reward shaping can be used to extend the standard Q-learning algorithm to support SMDPs. In this section, we firstly propose a time-based reward shaping method, *TRS*, and discuss the solution optimality in connection with SMDP. Then, we elaborate the validity of our proposed method through two counter examples.

4.1 Our Solution

We propose the simple solution of using the time spent for an action as an additional negative reward given to the agent after it completes that action. Formally, we define the time-based shaping function in Eq. 4.

$$F(s, a, s') = -\tau(a) \tag{4}$$

where $\tau(a)$ is the number of time steps it takes the agent to complete action a

As the time spent for an action is independent of transition states, the reward shaping is not potential based. Consequently, we can not guarantee the solution optimality for any SMDP problem. However, we observe that solution optimality may be preserved for some SMDP problems if the problem satisfies some properties.

We begin by showing that using reward shaping (in Eq. 4), the agent will never learn a solution consisting of a cyclic sequence of states for which the individual rewards total to a positive reward, as this allows for an infinite loop. Then, we proceed to prove a guaranteed consistency between the optimal policy when learning by our approach, time-based reward shaping, and when using an approach like SMDPQ, i.e. that our approach converges to the optimal policy as SMDPQ.

No Cyclic Policy As shown in Eq. 4, only negative reward is given in the learning process so that it is clear that no cyclic sequence of states can result in a positive reward. This ensures that the possible cyclic issue of a poorly chosen reward function can not occur in our solution.

Policy Equivalence We find that the termination reward is actually insignificant using our TRS method, since it will converge to the fastest way to termination even when reward is e.g. zero for all states in the SMDP problem. However, the termination reward is required in order to compare it to SMDPQ. In SMDPQ the algorithm selects the fastest path to the goal, by discounting

the reward over time, while our approach gives a negative reward for each time step used until termination. Both of these approaches make sure that a faster path has a higher reward than all of other paths. This however is only true if the reward received at the terminal states is the same for all terminal states. If they were to differ, the two approaches are not guaranteed to find the same policy, as the negative reward earned by the time spent, together with the discounting, may have conflicting "priorities".

The Q functions for *TRS* and SMDPQ , denoted Q_{TRS} and Q_{SMDPQ}, can be seen respectively in Eqs. 5 and 6. The shown values are only for special cases where α is 1 for both approaches and γ is 1 for Q_{TRS} and between 0 and 1 for Q_{SMDPQ}. τ is here the time to termination by taking action a in state s and following the policy after this. The equation shows that any action that leads to a faster termination, will have a higher Q-value, and as this is the case for both algorithms they end up with the same policy.

$$Q_{TRS}(s,a) = r - \tau \tag{5}$$

$$Q_{SMDPQ}(s,a) = \gamma^{\tau} * r \tag{6}$$

Formally, we assume that a specific group of SMDPs shall have the following properties:

1. Reward is only given at the end, by termination,
2. The goal must be to terminate with the lowest time consumption,
3. The reward must be the same for all terminal states.

Then, time-based reward shaping preserves the solution optimality of SMDPQ as indicated in proposition 1. However, the policies can be equal even though the restrictions do not hold, but this is not guaranteed.

Proposition 1 (Policy Equivalence)

$$\forall s : \arg\max_a Q^*_{TRS}(s,a) = \arg\max_a Q^*_{SMDPQ}(s,a)$$

We note that our approach is not simply a matter of guiding the learning, but actually giving it essential information to ever finding the optimal policy. This however is only the case when using a MDP algorithm to solve the problem. When the algorithm converges the agent will choose a policy allowing for the fastest solution of the given problem—measured in time and not number of actions. The application of the reward penalty encourages the agent to achieve a goal as fast as possible, thus making our approach independent of game-specific properties such as specific units etc.

4.2 Counter Examples

Here we present two examples of why the previousy mentioned properties on applicable SMDPs have to be obeyed in order for TRS to converge to the optimal policy, i.e. the same as the SMDPQ algorithm. We create examples for properties 1 and 3, and show that if the propeties are not obeyed, the two algorithms are not guaranteed to converge to the same optimal policy.

Rewards in Non-Terminal States Figure 1 shows the state-space of an example and illustrates why it is important that any reward should only be given in terminal states in order for Q_{TRS} to converge to the same optimal policy as Q_{SMDPQ}. For this case we use $\gamma = 0.99$, define the actions of the environment as A_1, A_2 and A_3, and the time steps required to take transitions as $\tau(A_1) = 1$, $\tau(A_2) = 2$, $\tau(A_3) = 50$. This results in the optimal policy using Q_{TRS} would go directly from S to T, but the optimal policy for Q_{SMDPQ} would be from S to T through S'.

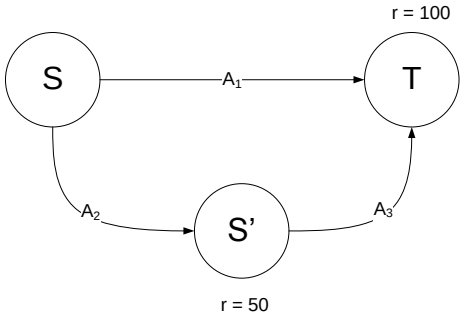

Fig. 1. A case illustrating a counterexample of why it is important that rewards are only given in terminal states.

The following shows the calculations resulting in the optimal policy using Q_{TRS}:

$$Q_{TRS}(S, A_1) = (100 - \tau(A_1))\gamma = \mathbf{98.0}$$

$$Q_{TRS}(S, A_2) = (50 - \tau(A_2))\gamma + (100 - \tau(A_3))\gamma^2 = 96.5$$

While the optimal policy using Q_{SMDPQ} is caculated as follows:

$$Q_{SMDPQ}(S, A_1) = \gamma^{\tau(A_1)}100 = 99.0$$

$$Q_{SMDPQ}(S, A_2) = \gamma^{\tau(A_2)}50 + \gamma^{\tau(A_2)+\tau(A_3)}100 = \mathbf{108.3}$$

The goal of a Q_{TRS} problem should be to reach termination in the fewest time steps, which means that giving rewards in non-terminal states, and thereby not obeying the restrictions, will result in a possible suboptimal solution.

Different Rewards in Terminal States Figure 2 shows its importance that all terminal states must yield the same reward in order to assure that Q_{TRS} converges to the same optimal policy as Q_{SMDPQ}. In this case the following parameter values are used: $\gamma = 0.99, \tau(A_1) = 60, \tau(A_2) = 1$. This results in the optimal policy using Q_{TRS} would go from S to T_2, but the optimal policy for Q_{SMDPQ} would be T_1 instead.

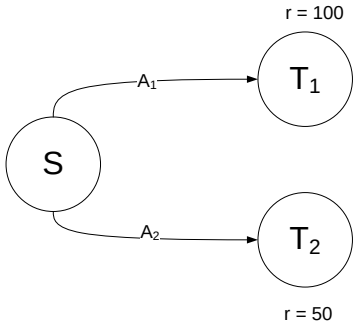

Fig. 2. A case illustrating a counterexample of why it is important that all the rewards given in terminal states need to be the same.

The following shows the reward calculations for the optimal policy using Q_{TRS}:

$$Q_{TRS}(S, A_1) = (100 - \tau(A_1))\gamma = 39.6$$

$$Q_{TRS}(S, A_2) = (50 - \tau(A_2))\gamma = \mathbf{48.5}$$

While the reward calculations using Q_{SMDPQ} are as follows:

$$Q_{SMDPQ}(S, A_1) = \gamma^{\tau(A_1)}100 = \mathbf{54.7}$$

$$Q_{SMDPQ}(S, A_2) = \gamma^{\tau(A_2)}50 = 49.5$$

All terminal states in a Q_{TRS} problem must yield the same reward, and if this is not obeyed the optimal policy will, as exemplified above, not be guaranteed to be equivalent to the optimal policy of Q_{SMDPQ}.

5 Experiments

To test the proposed time-based reward shaping, we set a simple scenario in the RTS game Balanced Annihilation which can be described as an SMDP problem. The optimal solution for this problem is learned using Q-learning and Q(λ)[2] with time-based reward shaping, and the proven SMDP approach SMDPQ[1].

5.1 Game Scenario

The scenario is a very simple base-building scenario. The agent starts with a single construction unit, a finite amount of resources and a low resource income. The agent controls the actions of the construction unit, which is limited to the following three actions:

- Build a *k-bot lab*, for producing attack-units (production building)
- Build a *metal extractor*, for retrieving metal resources (resource building)
- Build a *solar collector*, for retrieving solar energy (resource building)

All actions in the scenario are sequential, as the construction unit can only build one building at a time. The goal of the scenario is to build four of the production buildings as quickly as possible (in terms of game time). The build time depends on whether we have enough available resources for constructing the building; e.g. if we have low resource income and our resource storage is empty, it takes much more time to complete a new building than if we have high resource income. Therefore the optimal solution is *not* to construct the four production buildings at once, without constructing any resource building, as this would be very slow. Figure 3 shows a screenshot of a possible state in this scenario in Balanced Annihilation.

Fig. 3. A screenshot of the scenario in Balanced Annihilation. This shows the state of the game after five *metal extractors*, one *solar collector*, and two *k-bot labs* have been build. The construction unit is currently in the process of building a third *k-bot lab*. At the top of the screen a white and a yellow indicator can be seen, representing the current level of metal end solar energy, respectively.

As state variables, the number of each type of building is used; the number of production building has the range $[0; 4]$, and the numer of each of the two types of resource buildings has the range $[0; 19]$. This results in a state space of $5 \times 20 \times 20 = 2000$ and an state-action space of $2000 \times 3 = 6000$. This means that it is not possible for the agent to take the current amount of resources into account, as this may vary depending on the order in which the buildings have been constructed. We do not think that this will have a great impact on the final policy though.

5.2 Results

Figure 4 shows the results of four different settings of reinforcement learning in the scenario: Standard Q-learning and Q(λ) with time-based reward shaping, and SMDPQ both with and without time-based reward shaping.

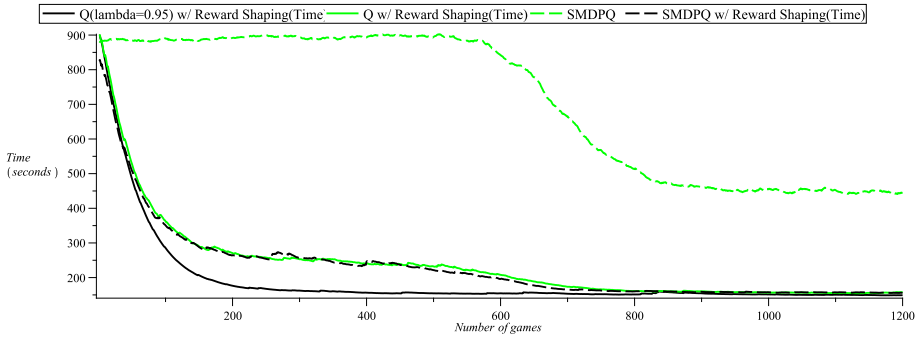

Fig. 4. A graphical representation of the convergence of the different approaches for SMDP support in Q-learning. Q-learning values are: $\alpha = 0.1$, $\epsilon = 0.1$, $\gamma = 0.9$. Exponential smoothing factor = 0.02. This experiment was done on the base-building scenario.

The standard MDP Q-learning algorithm extended with time-based reward shaping shows a significant improvement over standard SMDPQ. SMDPQ with reward shaping also provides much faster convergence than standard SMDPQ. However, there is no difference on applying reward shaping to standard MDP Q-learning and SMDPQ. This can be explained by the fact that SMDPQ and our reward shaping both help solve the problem as fast as possible, since the reward decreases as time increases. The algorithms are not additive, so applying one time penalty-algorithm to another does not increase the convergence rate. In addition, both algorithms only pass reward back one step, and thus the two approaches converge at the same rate.

When using the MDP algorithm on the scenario, which is in fact an SMDP problem, this kind of reward shaping is actually necessary in order to make the algorithm converge to a correct solution. Without reward shaping, the agent

would know nothing about the time it took to complete an action, and so it would converge to the solution of building four production buildings in as few steps as possible, namely four. This solution is not the optimal one in terms of game time, since the production of resources would be very low given the fact that the agent does not build any resource buildings.

From Figure 4 it is not clear whether or not SMDPQ without reward shaping actually ever converges. In this experiment SMDPQ converged to the optimal policy after approximately 22000 runs, but this result is not a part of the figure, since it would obfuscate the other information contained within the graph.

Adding eligibility traces, in this case Q(λ), to the Q-learning with TRS furthermore significantly improves the convergence rate, as it can be seen in Figure 4.

5.3 Discussion

As we have shown in our case, it is important to use reward shaping to reduce the time spent on the learning. It is especially important when each learning session takes more than a few seconds. In this case the change from 22000 sessions to 700, can result in a problem being feasible to learn and not take several hours or even days to learn.

TRS can be a good choice when using RL for a sub-task in a game, where the agent must solve a problem as fast as possible and can not fail to achieve the goal. This however does not include an RTS game as a whole, as it is possible to fail to achieve the goal, by loosing the game. General reward shaping can be applied to all MDP problems, but it is especially important in games, where the state space can be very large. Another task which could be solved using TRS is pathfinding (in general).

6 Conclusions and Future Work

Scenarios in computer games are often time dependent and agents' actions may span multiple time steps. We propose an SMDP solution to have agents learn adaptive behaviours. We make a further step to speed up the learning process through reward shaping techniques. We propose a time-based reward shaping method, *TRS*, that punishes the delayed actions using times. In particular, we show that our method may result in the same policy as SMDP solutions for some specific problems. Our experiments on the Balanced Annihilation game show that applying time-based reward shaping is a significant improvement for both general Q-learning, SMDPQ and Q(λ), allowing fast convergence when solving some SMDPs. In addition, the technique allows us to solve SMDP problems with the standard Q-learning algorithm.

We found a way to use reward shaping in Q-learning that reduced the number of runs needed to converge for a restricted group of SMDP problems. It would be interesting to see if this, or some similar approach, could be applied in a more general case. There is already a general concept of reward shaping, which

has been proved to work. But we feel that some more specific concepts about including time usage in reward shaping could really be beneficial. Improvements of this approach could result in loosing the restrictions for the SMDP problems, optimally allowing support for any SMDP problem.

Applying TRS to more advanced problems could also be done, to show whether this approach is too limited. An example of such an advanced scenario could be one with cooperative agents. This is especially relevant in RTS games, where units need to cooperate in order to achieve their goals faster.

References

1. Richard Sutton, Doina Precup and Satinder Singh: *Between MDPs and Semi-MDPs: A Framework for Temporal Abstraction in Reinforcement Learning.* Artificial Intelligence 112, 181–211 (1999).
2. Richard S. Sutton and Andrew G. Barto: *Reinforcement Learning: An Introduction.* MIT press, 1998.
3. Imran Ghory: *Reinforcement learning in board games.* Technical Report, Department of Computer Science, University of Bristol, 2004.
4. Michael L. Littman: *Markov games as a framework for multi-agent reinforcement learning.* In Proceedings of the Eleventh International Conference on Machine Learning, pp. 157-163, 1994.
5. Maja J. Mataric: *Reward Functions for Accelerated Learning.* In Proceedings of the Eleventh International Conference on Machine Learning, pp. 181-189, 1994.
6. Manu Sharma, Michael Holmes, Juan Santamaria, Arya Irani, Charles Isbell, Ashwin Ram: *Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL.* In Proceedings of the 20th international joint conference on Artifical intelligence, pp. 1041-1046, 2007.
7. Bhaskara Marthi, Stuart Russell, David Latham, Carlos Guestrin: *Concurrent Hierarchical Reinforcement Learning.* In Proceedings of the Nineteenth International Joint Conference on Artificial Intelligence, pp. 779-785, 2005.
8. Andrew Y. Ng and Daishi Harada and Stuart Russell: *Policy invariance under reward transformations: Theory and application to reward shaping.* In Proceedings of the Sixteenth International Conference on Machine Learning, pp. 278-287, 1999.
9. Adam Daniel Laud: *Theory and application of reward shaping in reinforcement learning.* University of Illinois at Urbana-Champaign, 2004.
10. Andersen K. T., Y. F. Zeng, D. Tran and D. D. Christensen: *Experiments with Online Reinforcement Learning in Real-Time Strategy Games.* Applied Artificial Intelligence: An International Journal, 23(9): 855-871 (2009).
11. Kaelbling, L. P., M. L. Littman, and A. W. Moore: *Reinforcement learning: a survey.* Journal of Artificial Intelligence Research *4*, 237–285 (1996).
12. Tesauro, G.. *Td-gammon, a self-teaching backgammon program, achieves master-level play.* Neural Comput. *6*(2), 215–219 (1994).
13. Manslow: *Using reinforcement learning to solve ai control problems.* AI Game Programming Wisdom *2*, 591–601 (2004).