Maciej KUTYŁO, Marcin PLUCIŃSKI, Magdalena LASKOWSKA

Faculty of Computer Science and Information Technology, West Pomeranian University of Technology

doi:10.15199/48.2015.02.33

Application of the reinforcement learning for selecting fuzzy rules representing the behavior policy of units in RTS-type games

Abstract. *The aim of the presented research was to prove the feasibility of the fuzzy modeling employing in combination with the reinforcement learning, in the process of designing an artificial intelligence that effectively controls the behavior of agents in the RTS-type computer game. It was achieved by implementing a testing environment for "StarCraft", a widely popular RTS game. The testing environment was focused on a single test-scenario, which was used to explore the behavior of the fuzzy logic-based AI. The fuzzy model's parameters were adjustable, and a Q-learning algorithm was applied to perform such adjustments in each learning cycle.*

Streszczenie. *W artykule przedstawiono badania możliwości połączenia modelowania rozmytego z uczeniem ze wzmocnieniem w procesie projektowania inteligentnego algorytmu, który będzie efektywnie kontrolował zachowanie agentów w grze typu RTS. Aby osiągnąć założony cel, zaimplementowano testowe środowisko w popularnej grze RTS „StarCraft". W środowisku tym realizowano jeden założony scenariusz gry, w którym badano zachowanie opracowanego algorytmu rozmytego. Parametry modelu rozmytego były modyfikowane za pomocą metody Q-learning.* ***(Zastosowanie uczenia ze wzmocnieniem w doborze reguł rozmytych reprezentujących strategię zachowań jednostek w grach typu RTS)***

Keywords: reinforcement learning, Q-learning, fuzzy model, RTS game
Słowa kluczowe: uczenie ze wzmocnieniem, algorytm Q-learning, model rozmyty, gra RTS

Introduction

Fuzzy logic is currently being used in many different fields of science and everyday life, ranging from relatively simple industrial applications to sophisticated research models. Fuzzy logic makes possible the approximation of human reasoning, leading to the creation of increasingly sophisticated expert systems, and enables complex modeling of the human decision making-process [1, 2].

The real-time decision-making under dynamically changing conditions seems to be an especially interesting aspect of fuzzy logic. Classic approach to modeling is not efficient for more complex systems due to the immense size of data required to properly identify the system, and often due to the impossibility of creating a detailed model of the system. In some cases, where a priori designing of a suitable decision model is not possible, learning systems can be used to obtain "knowledge" from their interactions with the system's environment. The reinforcement learning is an example of such an approach. In this case, the learning system is not provided with any training examples; the result of its actions is evaluated instead [2, 3].

The combination of the reinforcement learning with the fuzzy modeling is not quite a new approach, of course. In the case of continuous environment state, the fuzzy model can be a very good alternative to the popular table model which is used in policy representation [4, 5]. Examples of such an approach can be found, first of all, in such areas as automation and robotics [6, 7, 8, 9, 10].

One of the many possible applications of the aforementioned concepts is developing artificial intelligence of computer opponents in video games. Developers of entertainment software compete to provide the players with increasingly challenging computer adversaries, based on more and more sophisticated artificial intelligence systems, and the players themselves are constantly expecting greater challenges to tackle.

The presented work was intended to develop a fuzzy logic system able to represent the behavior policy of agents in an RTS[1]-type computer game. Selection of proper adaptation rules for the system was also necessary to obtain an appropriate efficiency of the agents' as a result of the reinforcement learning [3].

The above task is concerned with two complementary concepts: macromanagement ("macro") and micromanagement ("micro"). Macromanagement comprises proper allocation of acquired resources, technology development and creation of new structures and units – it can be understood as corresponding to the strategic level of decision making. Micromanagement, on the other hand, involves controlling individual units or small groups of units – corresponding to decisions on the tactical level.

When developing an artificial intelligence of a computer opponent, more emphasis is traditionally put on macromanagement – a sophisticated artificial intelligence on the tactical level is still a rare sight. In fact, the computer player often cheats by having access to more data that would normally be available to a human player. This is a major flaw from the perspective of comparing a human player with their computer counterpart. Developing an efficient system for learning behavior patterns on the micromanagement level would make it possible to increase the competitiveness of the computer player without using cheats [3].

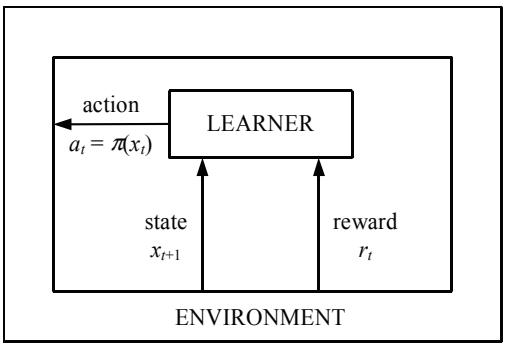
Reinforcement learning

The reinforcement learning (RL) is based on gaining a procedural knowledge (skill) during intermediate interactions with an environment in which this skill will be used to perform a given task. A learning system/agent doesn't need any a priori knowledge about an environment and it even doesn't need to know explicitly the task that it learns to perform. During interactions with an environment an agent receives only a scalar "reward" or "reinforcement" feedback signal indicating how good or bad its action was and on that basis it tries to adapt its future action policy to receive better rewards [5, 11].

The RL is a very universal approach but it is often very hard in a practical realization. It is the reason why other learning methods, like supervised learning for example, are much more popular. One of the main difficulties is the delay (sometimes very large) of an environment reward. Another problem is frequent stochastic and nonstationary character of an environment.

The RL is very close to human learning, because a man obtains a lot of his skills by a trial and error method, so RL is potentially one of the best approaches to creating really

[1] Real Time Strategy – a type of a computer war game in which players position and maneuver the units and structures under their control in order to gain and maintain control over the battlefield and to destroy their opponents' assets.




```
for all learning steps t
    observe an environment state x_t
    choose an action a_t to be taken
       in a state x_t (according to
       current policy)
    perform an action a_t
    observe a reinforcement r_t
    observe a next state x_t+1
    on the base of an experience
       <x_t, a_t, r_t, x_t+1> improve a policy
end
```

Fig. 1. A schematic diagram and the basic algorithm of the RL [18].

intelligent systems. The RL was successfully used for many practical applications from a policy learning in board games [5, 12, 13] or exploration tasks [11] to learning given behaviours of mobile robots [4, 8, 14, 15, 16, 17].

Generally, we can say that during the RL we want to find an optimal action policy in an unknown environment to solve a given task. A learner can observe a state of the environment and on the base of its value it chooses its action according to its current policy. At the beginning, a policy is taken arbitrarily, so it is very typical for a learning process that learner makes a lot of wrong actions. The learner makes errors and receives a reinforcement/reward feedback signal from the environment. On the base of this information it tries to improve its policy.

A policy depends only on the environment state, so during learning a learner is to find an optimal mapping from perceived states to actions to be taken when in those states:

$$\pi : X \to A,$$

where: π – a policy, X – a set of environment states, A – a set of possible learner actions [5, 18]. A schematic diagram of the RL is presented in Fig. 1.

For a current learner policy π, we can define a value function that is a mapping from states to the total amount of reward an agent can expect to accumulate over the future, starting from the state x:

$$V^{\pi}(x) = E_{\pi}\left[\sum_{t=0}^{\infty} \gamma^t \cdot r_t \mid x_0 = x\right], \tag{1}$$

where: E_{π} – expected (for the policy π) sum of future rewards r_t, $\gamma \in (0, 1]$ – discount rate, which determines that rewards received in the future are less worth for the state value.

As it is often convenient to analyse state-action pairs, an action value function can be defined as follows:

$$Q^{\pi}(x, a) = E_{\pi}\left[r(x, a) + \sum_{t=1}^{\infty} \gamma^t \cdot r_t \mid x_0 = x\,,\ a_0 = a\right]. \tag{2}$$

The action value function assigns each state-action pair $\langle x, a\rangle$ an expected (for the policy π) value of the immediate reward $r(x, a)$ and the discounted sum of future rewards assuming that the agent will start its activity in the state x with the action a.

During the RL, a learner looks for an optimal policy π^* – the policy for which it will always receive the best rewards from an environment. For such a policy, the value function $V^{\pi^*}(x)$ is always biggest or equal to the value function $V^{\pi}(x)$ for any policy π.

If the optimal value function $V^{\pi^*}(x)$ is known, we can easily find an optimal action policy π^* as a greedy policy to $V^{\pi^*}(x)$. A greedy policy to $V(x)$ always chooses its action to maximize an immediate reward and a discounted next state value of a value function $V(x)$. If the action value function is determined, then the greedy policy can be defined as:

$$\pi(x) = \arg\max_{a} Q(x, a)\,. \tag{3}$$

In the research, the Q-learning algorithm [5, 18] was used to find the optimal policy. As the name of the algorithm implies, it enables learning of the action value function and, after learning, the optimal policy can be defined as greedy to the determined action value function Q. One of the advantages of this algorithm is its independence from the policy that is learnt. It means that during learning an agent can use any arbitrary policy. This feature allows for good exploration of the state space and enables non-zero probability of executing each action. More specifically, the algorithm is described in the next section.

Method of the fuzzy model learning

StarCraft[2] was chosen as an epitome of an RTS game. Its longevity on the computer entertainment market, the forming of professional teams and competitions, along with a large fanbase, resulted in the refinement of macromanagement strategies (termed "build orders"), which are by nature strictly determined schemes.

In the case of the micromanagement, however, in most cases there are no strict behavior schemes, just fuzzy rules governing the desired course of an action. This is the reason why the micromanagement was chosen as the key concept for implementing the fuzzy system described by the presented work.

During the development of the fuzzy system, choosing its adaptation method and testing its performance, the following tools were used: Microsoft Visual Studio 2008, BWAPI[3], ChaosLauncher[4] and StarCraft Campaign Editor[5]. Use of the aforementioned tools made it possible to extract information from the game environment, analyze it with the created program and then give feedback influencing the behavior of units, such as attack or move orders.

For the purposes of the presented work, there was a need of creating a proper testing environment which would allow for the needed repeatability of testing conditions. As

[2] Published in 1998 by Blizzard Entertainment, the RTS game StarCraft was, at the time of writing, still experiencing unwavering interest and support from its fanbase; in South Korea, StarCraft was cited as the epitome of "eSport" (electronic sport), and professional StarCraft leagues were established, with matches being transmitted by three different television channels and teams being sponsored by major Korean companies.

[3] BWAPI (Brood War API) is an Application Programming Interface, distributed under GNU license, developed to be used with StarCraft. It enables communication with the game environment by executing code contained in dll libraries created by the user [19].

[4] A tool for injecting dll libraries into StarCraft's code.

[5] An editor for StarCraft with which the training map for the fuzzy logic system was created.

stated before, the emphasis was put on the micromanagement aspect of the game. The created testing map enabled the repeated replaying of a given micromanagement scenario – 3 Vulture units versus 12 Zergling units. Under normal conditions, without micromanagement from the player, the Vulture units would be doomed to lose. With proper micromanagement techniques, however, it is possible for them to win the engagement. The goal of the developed system was to learn how to emerge victorious from engagements such as described above.

Reasoning was being carried out for each individual unit and took into account all enemy units. On the input side of the fuzzification block, there were values of the unit's health points (*hp*), enemy unit's health points (*ehp*) and the distance between the units (*dist*). Membership functions for all values were triangular-shaped (Fig. 2). Each of values could belong to three fuzzy sets: L (low value), M (medium value), and H (high value).

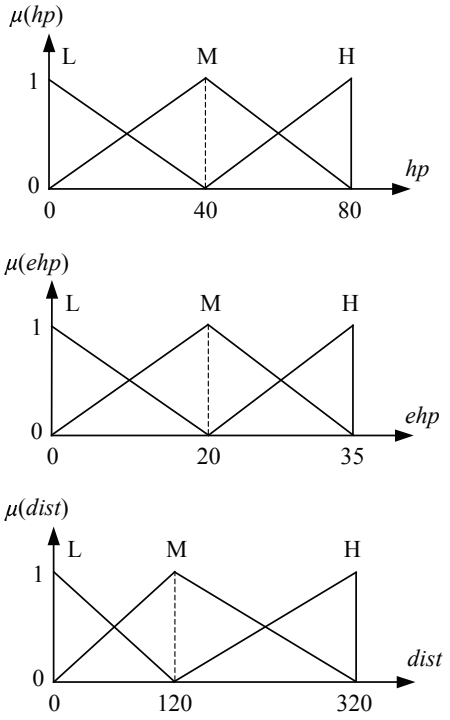


Fig. 2. Membership functions of the input values.

The model's inference block comprised a rule base taking into account all the possible input fuzzy sets as well as output fuzzy sets – that is, a linguistically complete rule base. A MAX-MIN inference mechanism was used, and the aggregation operation was carried out using a PROD operator. Activation level of each rule's conclusions was calculated using Mamdani implication[6]. Accumulation was based on the MAX operator. The use of this operator was substantiated by the simplicity of the model's output – a binary function (A – attack enemy unit, R – retreat away from enemy unit). The defuzzification block determined the output value of the model based on the resulting output membership function, calculated in the process of accumulation. A simplified defuzzification method was used – either A or R was the output value, one for each of the enemy units (whereas the attack or retreat point was always in relation to the closest enemy unit). The highest values of the conclusions' membership functions for both output values (A and R) were compared, while the biggest value so far was stored along with the identifier of the enemy unit it was pertaining to. After comparing the values for all enemy units, the highest value for A was compared to the highest value for R. The order given to the unit (that is, the output value of the model) was the one with the higher membership function value.

The above cycle was repeated for each friendly unit, for every animation frame of the game – it could be therefore accurately assumed that the decisions concerning the orders to be given to the units were being made constantly, in real time.

The created training map made it possible to play out a chosen number of single skirmishes, with the number of player's victories being stored and updated constantly, so that the effectiveness of the player's actions could be determined after all the skirmishes had been finished. This data was crucial for the system's adaptive process. The training map was tailored to the needs of the learning process by eliminating the need for manual interference between learning episodes.

The developed fuzzy logic system was adapting by modifying the core values of the fuzzy sets M for each of the input values (*hp*, *ehp*, *dist*). Adequate limits to the core values were also set, so that they covered only the possible values of the corresponding parameters.

Q-learning was used as the learning algorithm, and the learning process was epoch-based. Every 30 skirmishes was treated as one episode and accounted for one of the iterations (steps) of the training map. At the start of each iteration, the number of victories in the previous episode was retrieved – based on this number, the current state was determined (Table 1). Then, an action to perform was selected (Table 2).

Table 1. Definition of model states.

State	% of victories
0	$0 \leq x < 20$
1	$20 \leq x < 40$
2	$40 \leq x < 60$
3	$60 \leq x < 80$
4	$80 \leq x \leq 100$

Table 2. Actions available in each learning episode.

Action	Name	Effect
0	*hp* $-$ 4	Decrease the core value of set M for value *hp* by 4
1	*hp* $+$ 4	Increase the core value of set M for value *hp* by 4
2	*ehp* $-$ 5	Decrease the core value of set M for value *ehp* by 5
3	*ehp* $+$ 5	Increase the core value of set M for value *ehp* by 5
4	*dist* $-$ 15	Decrease the core value of set M for value *dist* by 15
5	*dist* $+$ 15	Increase the core value of set M for value *dist* by 15
6	–	No action

Updating of the values of the matrix Q was done after each episode, for which the selected action was performed, according to formula (4), where: $Q^t(x_t, a_t)$ – is the action value function in the episode t. Matrix R (5), containing the reward values for changing (or maintaining) states, was used in the updating process. An additional feature of the system was the inclusion of a number of factors that influenced the adaptation process, such as: step size (β), discount factor

[6] In the case of singleton outputs, Mamdani implication simplifies to comparing the aggregation result to 1 (MIN(μ_R(*hp*,*ehp*,*dist*),1). This step was omitted, because using the PROD operator during the aggregation step guarantees that the result will be less than or equal to 1.

(γ), and exploration factor (ϵ) [5, 18].

$$(4)\quad Q^{t+1}(x_t, a_t) = Q^t(x_t, a_t) + \beta\left(R(x_t, x_{t+1}) + \gamma \max_a Q^t(x_{t+1}, a_t) - Q^t(x_t, a_t)\right)$$

$$(5)\quad R = \begin{bmatrix} -10 & 10 & 40 & 70 & 100 \\ -20 & -10 & 40 & 70 & 100 \\ -60 & -20 & -10 & 70 & 100 \\ -90 & -60 & -20 & -10 & 100 \\ -150 & -150 & -150 & -150 & 100 \end{bmatrix}$$

Initial values of the aforementioned matrices and factors, types of employed fuzzy operators, as well as initial values of core values M, were chosen based on the author's knowledge and were subject to updates during research.

Results of experiments

Initial testing allowed for the proper adjustment of the factors, matrices and values mentioned above, so that convergence of the learning process could be achieved. An illustration of the achieved results is presented in Table 3 and Figure 3.

Table 3. Research #24 results.

Number of steps: 50	*hp*(M) = 40
Step size factor: $\beta = 0.9$	*ehp*(M) = 25
Discount factor: $\gamma = 0.99$	*dist*(M) = 255
Exploration factor: $e = 0.98^t$	

Skirmishes won: 29
Maximum number of victories: 29
Minimum number of victories: 5
Matrix Q:

35.99	42.92	0.00	0.00	0.00	0.00	0.00
92.69	35.09	0.00	0.00	0.00	97.82	49.67
52.12	186.79	52.12	0.00	53.48	57.97	0.00
27.19	31.67	0.00	−9.00	0.00	196.77	25.82
0.00	0.00	0.00	−94.32	29.32	0.00	726.81

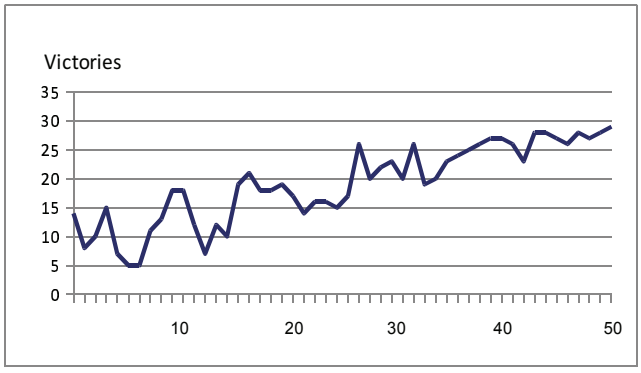


Fig. 3. Convergence of the learning process for research #24.

Achieving of the convergence was required to step forward to the last phase of the research – running the program with constant values of matrix Q, determined in one of the previous steps with clearly visible convergence – final values of matrix Q from the last ($m = 50$) step of research #24 (Table 3) were used. The core values M were left modifiable. Results of the experiment are presented in Table 4 and Figure 4.

The final step was to test the efficiency of the agents' behavior for the core values M determined during the learning process of the previous research (Table 5). Test results are shown in Figure 5.

Table 4. Research #29 results.

Number of steps: 100	*hp*(M) = 76
Step size factor: 0	*ehp*(M) = 20
Discount factor: 0	*dist*(M) = 195
Exploration factor: 0	

Skirmishes won: 30
Maximum number of victories: 30
Minimum number of victories: 12
Matrix Q:

35.99	42.92	0.00	0.00	0.00	0.00	0.00
92.69	35.09	0.00	0.00	0.00	97.82	49.67
52.12	186.79	52.12	0.00	53.48	57.97	0.00
27.19	31.67	0.00	−9.00	0.00	196.77	25.82
0.00	0.00	0.00	−94.32	29.32	0.00	726.81

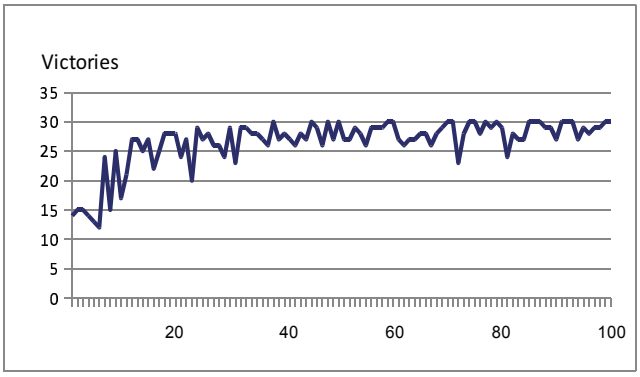


Fig. 4. Efficiency for research #29.

Conclusions

The achieved results are very satisfactory and support the thesis of viability of using fuzzy modeling and reinforcement learning for developing artificial intelligence for efficient agent control in an RTS game environment.

Additional tests were carried out in which the aggregation operator PROD had been replaced with the MIN operator. Those tests, however, yielded very unsatisfactory results. This was a consequence of too large dependence of the conclusions' activation levels on the lowest values of their premises' membership functions.

Another result that is worth mentioning is the agents' behavior without the developed system's influence – based only on the behavior patterns initially present in the game's environment (default AI). In the above example, the number of skirmishes won was 0 for each of the 50 steps, which solidifies the conclusion about the efficiency of the developed system.

The research carried out on the course of the presented work confirmed the efficiency of the chosen approach. However, many possibilities of further development of the posited implementation exist. First of all, the problem of the learning algorithm's dependency on the initial core values M, that was encountered during testing, should be addressed. The above dependency had a strong influence on efficiency (0 – 1 skirmishes won for theoretically worst possible starting core values M, 24 – 30 for theoretically best).

Having in mind the efficiency of controlling agents in a

Table 5. Core values M determined during the model's learning process.

Set	Core value
hp(M)	76
ehp(M)	20
dist(M)	195

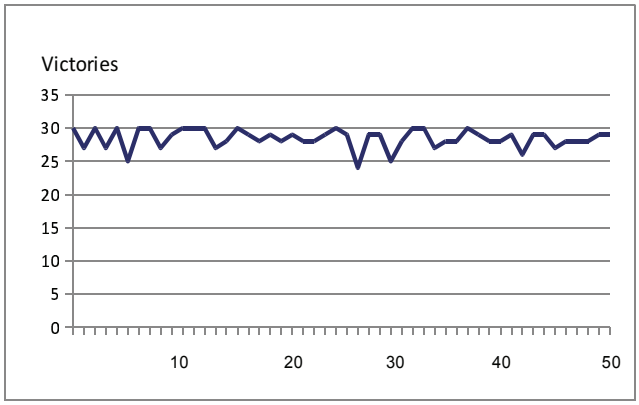


Fig. 5. Results of the final agent control efficiency test.

real game environment, the presented implementation could be further enhanced by enabling control over different types of units – which would require a proper generalization of the model, and perhaps some extent of modification to the rules or an increased number of factors to be considered by the system while making a decision. A real game environment would also pose additional limitations, such as terrain configuration on the map, which would require a more advanced decision process. Integration with other artificial intelligence applications would also be recommended, especially those responsible for macromanagement, as it would enable to further boost the competitiveness of the computer player.

From a theoretical standpoint, it would be interesting to test the behavior of a more sophisticated model, taking into account more factors during the decision process. Developing such a system would necessitate employing much more sophisticated behavior mapping, which would positively influence the model's versatility.

REFERENCES

[1] Mendel J.M.: Uncertain rule-based fuzzy logic systems: introduction and new directions, Prentice Hall PTR, 2001.
[2] Piegat A.: Fuzzy modeling and control Physica Verlag, Heidelberg – New York, 2001.
[3] Kutyło M.: Application of the reinforcement learning for the selection of fuzzy rules representing the policy, in the software production process of the RTS-game, Eng. Thesis, West Pomeranian University of Technology, 2011, (in polish).
[4] Pluciński M.: Application of the probabilistic RBF neural network in the reinforcement learning of a mobile robot, *Polish Journal of Environmental Studies*, 16(5B), pp. 32-37, 2007.
[5] Sutton R.S., Barto A.G.: Reinforcement learning: An introduction, The MIT Press, 1998.
[6] Er M.J., Deng C.: Online tuning of fuzzy inference systems using dynamic fuzzy Q-learning, *IEEE Transactions on Systems, Man and Cybernetics – part B: Cybernetics*, 34(3), pp. 1478-1489, 2004.
[7] Glorennec P.Y., Jouffe J.: Fuzzy Q-learning, *Proceedings of the 6th IEEE International Conference on Fuzzy Systems*, vol. 2, pp. 659-662, 1997.
[8] Lin L.J.: Hierarchical learning of robot skills by reinforcement, *Proceedings of the IEEE International Conference on Neural Networks*, pp. 181-186, 1993.
[9] Lin C.K.: A Reinforcement learning adaptive fuzzy controller for robots, *Fuzzy Sets and Systems*, 137(3), pp. 339-352, 2003.
[10] Maeda Y.: Modified Q-learning method with fuzzy state division and adaptive rewards, *Proceedings of the IEEE World Congress on Computational Intelligence*, pp. 1556-1561, 2002.
[11] Pluciński M.: Application of the reinforcement learning in searching of the exploration policy for many vehicles, *Polish Journal of Environmental Studies*, 17(3B), pp. 347-352, 2008.
[12] Sutton R.S.: Learning to predict by the methods of temporal differences, *Machine Learning*, 3, pp. 9-44, 1992.
[13] Tesauro G.: Practical issues in temporal differences learning, *Machine Learning*, 8, pp. 257-277, 1992.
[14] Bekey G.E., Autonomous robots (from biological inspiration to implementation and control), The MIT Press, 2005.
[15] Connell J., Mahadevan S.: Rapid task learning for real robots, *Robot Learning*, Springer US, pp. 105-139, 1993.
[16] Kaelbling L.P., Littman M.L., Moore A.W.: Reinforcement learning: A survey, *Journal of Artificial Intelligence Research*, 4, pp. 237-285, 1996.
[17] Millan J.R.: Rapid, safe, and incremental learning of navigation strategies, *IEEE Transactions on Systems, Man, and Cybernetics, part B.: Cybernetics*, 26(3), pp. 408-420, 1996.
[18] Cichosz P.: Learning systems, Wyd. Naukowo-Techniczne, Warszawa, 2000, (in polish).
[19] BWAPI, Available at: `http://code.google.com/p/bwapi/wiki/BWAPIManual`.

***Authors**: M.Sc. Eng. Maciej Kutyło, e-mail: maciejkutylo@gmail.com, Ph.D. Eng. Marcin Pluciński, e-mail: mplucinski@wi.zut.edu.pl, M.Sc. Eng. Magdalena Laskowska, e-mail: laskowskamag@gmail.com, Faculty of Computer Science and Information Technology, West Pomeranian University of Technology, Żołnierska 49, 71-062 Szczecin, Poland.*