REUSABLE REINFORCEMENT LEARNING FOR MODULAR SELF MOTIVATED AGENTS

Jaroslav Vítků, Pavel Nahodil
Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics
Technicka 2, 16627, Prague 6, Czech Rep.
Email: vitkujar@fel.cvut.cz, nahodil@fel.cvut.cz

KEYWORDS

Agent, Architecture, Artificial Life, Creature, Behaviour, Hybrid, Neural Networks, Evolution.

ABSTRACT

Presented topic is from the research fields called Artificial Life and Artificial Intelligence (AI). In this paper, there is presented novel approach to designing agent architectures with its requirements. The approach is inspired by inherited modularity of biological brains and agent architectures are represented here as set of given reusable modules connected into a particular topology. This paper presents design of two particular modules for future use in more complex architectures. The modules are used for implementing model-free motivation-driven Reinforcement Learning (RL). First, the novel framework for these architectures is described together with a used simulator. Then, the design of two new reusable domain-independent components of agent architectures is described. Finally, experimental validation of these new components and their future use is mentioned.

INTRODUCTION

This paper deals with design of agent architectures in the domain of Artificial Life (ALife), which is often inspired in *ethology*. In ethology, the emphasis is put on agents behaviour. Observing agents behaviour and determining its origins (decomposition of problem) is one possible source of inspiration for architecture design (called "top-down" approach). The other possible approach (called "bottom-up") is in connecting small systems (capable of simple behaviour) into larger architectures. This way of designing intelligent systems is often called *connectionism*. Such connectionist models may have promising future with new, more detailed models of neurons Maass (1996); Izhikevich (2003) together with emerging specialized hardware Thomas and Luk (2009) for them. Each approach has own advantages and drawbacks. Our focus is aimed more towards combining the two above together into new, hybrid systems. These hybrid architectures partly employ ethological principles and partly connectionist ones. Our approach focuses on reusability of particular components of agent architectures. This paper describes two reusable modules, which can be freely used in variety of modular architectures.

The first chapter of this paper describes our novel framework for representation and design of hybrid agent architectures, its goals and requirements. The second chapter describes theory and implementation of two new modules which implement domain-independent and model-free Reinforcement Learning (RL). The chapter Experiments describes experiments simple architecture used for verification of these modules. Finally, results of experiments are evaluated and future use of these modules in automatic design of agent architectures is mentioned.

HYBRID ARTIFICIAL NEURAL NETWORK SYSTEMS FRAMEWORK

Rather than designing one particular architecture suitable for a particular task, our research focuses on modular systems Auda and Kamel (1999) where each module can be reused in various architectures. An example of typical reusable domain independent module can be seen the Categorizing and Learning Module (CALM) Murre et al. (1989).

Neural Module

For this purpose, the framework called Hybrid Artificial Neural Network Systems (HANNS) is described. Its main goal is in unification of representation of particular modules, so that these modules can be seamlessly connected into bigger systems. Particular sub-systems use for communication the same methods as ANNs and are defined as "Neural Modules". Each Neural Module can have Multiple Inputs/Multiple Outputs (MIMO), either real-valued or spiking type. Neural Module can implement theoretically any component of agent architecture: sensory systems, decision-making modules or actuators. Scheme of Neural Module can be seen in the Fig.1, where particular components are explained in this section in more detail.

***Prosperity Measure**:* The other main goal of this framework is enabling automatic design of agent architectures by means of Evolutionary Algorithms (EAs). Here, similar methods to neuro-evolution Fekiac et al. (2011) can be used. While omitting multi-objective

Proceedings 28th European Conference on Modelling and Simulation ©ECMS Flaminio Squazzoni, Fabio Baronio, Claudia Archetti, Marco Castellani (Editors)
ISBN: 978-0-9564944-8-1 / ISBN: 978-0-9564944-9-8 (CD)

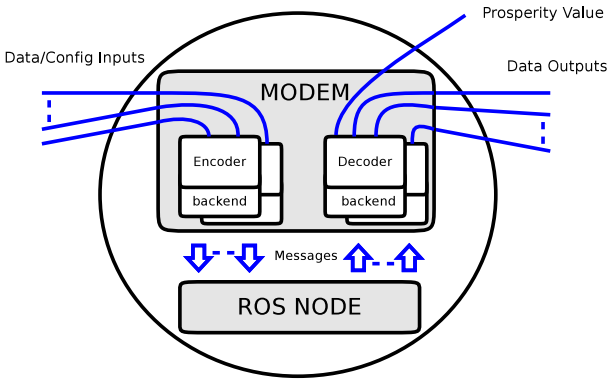

Fig. 1. Scheme of Neural Module with three data and configuration inputs, three outputs, one "prosperity" output and arbitrary inner structure. An encapsulated algorithm can be implemented by means of Robotic Operating System (ROS). The prosperity output represents subjective heuristics telling how well the algorithm performs.

optimization techniques Deb (2011), the evolutionary approach requires single measure of quality of a solution (architecture) represented by an individual. This measure is called the *Fitness*.

Often, the quality of a solution as a whole is evaluated. However architectures represented by HANNS can be composed of highly heterogeneous modules (from simple functions to complex algorithms). In order to give some insight whether a particular module is used efficiently in a given architecture, it is suitable to have some measure representing this. Therefore this framework defines the *Prosperity*. The Prosperity is similar to the Fitness, but represents heuristics which tells *"how well the particular Neural Module probably works"*. The value should be from the interval $\langle 0, 1 \rangle$, where 0 represents the worst, and 1 the best performance. This value is *"subjective"* and is provided by the Module itself during the simulation.

Data Connections and Configuration Connections: Neural Module is purposed to implement various algorithms. Despite the fact that such algorithms should be domain-independent, some configuration is needed in many cases. In most cases, the values of parameters are constant during the simulation. Therefore the HANNS framework distinguishes data and configuration connections. This can be used by specialized EA algorithms for separate searching for connections and parameters (such as Kordík (2006)) or to use predefined parameters and search only for data connections.

The Simulator NengoROS

In order to create and use as reusable modules as possible, the simulator NengoROS (available at url-http://nengoros.wordpress.com/) was created. It combines simulator of large-scale neural networks (based on Neural Engineering Framweork (NEF) Eliasmith and Anderson (2003)) called Nengo (available at nengo.ca) and the Robotic Operating System (available at http://www.ros.org/). The ROS is decentralized infrastructure based on nodes Quigley et al. (2009), which communicate by means of messages over the TCP/IP protocol. In the ROS, each node is separated process (several programming languages supported so far), by connecting several nodes together, a network-like structure can be cerated. Particular algorithms can be used (or implemented) as ROS Nodes, where each ROS node has own Jython interface which connects it into the NengoROS simulator. The Jython interface defines modem, a ROS node which translates ROS messages into the Nengo data, see Fig.1.

THEORETICAL BACKGROUND

When compared to the knowledge-based AI and to connectionism approaches, several types of RL algorithms have several advantages. Compared to supervised-learning ANNs, these algorithms do not require learning by examples. And compared to planning systems, they do not require even a model of the environment. RL is based only on rewards received as a result of some action executed. This makes RL algorithms suitable for unknown environments and also usable in the HANNS framework. For the integration, the type of RL, called Q-Learning was chosen.

Q-Learning Algorithm

The Q-Learning algorithm is suitable for online learning without need of environment model - it is model-free approach. During the learning, the algorithm updates the action-value function Q, which represents mapping set of agent's actions A and set of all admissible environment states S to real values according to the equation (1).

$$Q : A \times S \rightarrow \mathbb{R} \tag{1}$$

Values in the matrix $Q(s, a)$ then define the benefit of each action in a given actual state. When exploiting the knowledge learned the by Q-Learning algorithm, the best action (with the highest value in the matrix) can be selected at each step for obtaining best known policy in a given situation. At each step of the algorithm, values in the $Q(s, a)$ matrix are updated according to the equation (2):

$$Q(s_t, a_t) = Q(s_t, a_t) + \alpha\big[r_t + \gamma \max_a Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\big]. \tag{2}$$

Here, the $s_t \in S$ is a previous state of the environment, $a_t \in A$ is action which was just executed, r_t is reward received at a result to the action a_t, the current time step t, $\max_a Q(s_{t+1}, a_{t+1})$ is the action with the highest utility value in the current state. There are the following algorithm parameters: $\gamma \in \langle 0; 1)$ is a forgetting factor and $\alpha \in (0; 1\rangle$ is a learning rate, for more information see Vítků (2011).

The scheme of Q-Learning system and the principle of it's function is depicted in the figure 2. The Stochastic

Return Predictor (SRP) is composed of Q-Learning algorithm and Action Selection Method (ASM). The ASM selects the action and executes it, RL algorithm observes the reinforcements received and updates the value of the Q function for the previous state according to the eq. (2).

The algorithm can be further improved by the *Eligibility Traces*. When using the Eligibility traces, the algorithm updates values of several state-action pairs at one step Vítků (2011). The parameter λ defines how much are previous states updated. Correct estimation of the λ can greatly improve the speed of learning, but also can cause oscillations in learning. This modification is also called *Q-Lambda* algorithm.

***Action Selection Method**:* Actions to be executed by the agent are selected by the ϵ-Greedy Action Selection Method (ASM). The ϵ parameter defines amount of randomization: with the probability of ϵ, a random action is selected and with the probability of $1-\epsilon$ the Greedy action is taken. This helps the agent escape from the local extreme and encourages exploration of new states.

Motivation Source

As seen in the previous chapter, the amount of randomization (exploration of the state-space) can be chosen by varying the ϵ value. This can be taken further and enable the agent architecture to set the ϵ parameter dynamically during the simulation, based on the current situation. If there is a free time, living animals tend to play/explore the surrounding environment and therefore gain new knowledge. On the other hand, if the situation requires fast exploitation of current knowledge, it is not suitable to explore. In many systems, there is need to choose good tradeoff between *exploration* and knowledge *exploitation*.

In the past, our research team solved this for example by defining agents physiology Kadleček (2008); Kadleček and Nahodil (2001); Kadlecek and Nahodil (2008). The physiological variable can represent e.g. need for water. In time, as the value of variable (e.g. amount of water in body) decreases, the need for correcting this state (drinking) increases. After drinking, the variable is returned towards the optimal condition and the motivation decreases.

Exactly this purpose has the *Motivation Source*. The simplest case can include one linearly decaying physiological variable, where the amount of motivation inversely depends on a value of the physiological variable. Here, a more natural definition is used: the physiological variable decays linearly each time step t with predefined $decay$:

$$V_{t+1} = V_t - decay, \tag{3}$$

but the amount of motivation is determined by applying the sigmoid to the inverse value of physiological variable V. The resulting amount of Motivation M at time t is:

$$M_t = \frac{1}{1 + e^{min+(max-min)*(1-Vt)}} , \tag{4}$$

where min and max parameters are chosen, so that value of the variable $V_t = 0$ roughly corresponds to the motivation of $M_t = 1$.

DESIGN OF NEURAL MODULES

This section describes the design of two reusable Neural Module that can be used for model-free learning in modular agent architectures. This section briefly describes design and implementation of these modules. The first module implements the *Q-Lambda* algorithm together with the ASM. The second Module implements the *Physiological State Space*, which can serve as a source of motivation in agent architecture Kadleček (2008).

Q-Learning Module Design

Several design requirements have to be met in order to successfully implement the Q-Lambda algorithm in the Neural Module. First, the typical use-case and main requirements for such a Neural Module in the HANNS framework will be described. The requirements for integration of the Q-Lambda algorithm into the Neural Module are described in the following sections.

***Representing the Inputs and Outputs**:* Neural Modules in the HANNS communicates by vector of real values on the interval $\langle 0, 1 \rangle$. Since the module should be as compatible with classical ANN paradigms as possible, the encoding of input/output values is selected *1ofN*. In case of actions, only the currently selected one has non-zero value on its output. Compared to this, array of input values represent array of state variables. Each discrete state variable is sampled with predefined step form the interval $\langle 0, 1 \rangle$.

***Operation in non-Episodical Experiments**:* The Q-Learning belongs into the group of algorithms which learn episodically. At the beginning of each episode, the SRP should start to operate from randomly chosen state of the environment. This ensures that the algorithm learns efficiently in the entire state-space. However, in real-life experiments this cannot be provided often. The successful learning in continuous experiments is accomplished by connecting to the own motivation source. By dynamically adjusting the *exploration* vs. *exploitation* tradeoff, the learning can be efficient.

In a particular architecture, the RL module represents some behaviour. We introduce the parameter called **Importance**. Increasing of this parameter affects two following components in the Neural Module:

- Causes **decrease of ϵ parameter** in the ϵ-Greedy ASM. Therefore, when the behaviour represented by the module has high importance, the exploration is suppressed.
- Causes **increase of value of the selected action**. This ensures that in competition against other RL modules (or other action-selecting sub-systems) has higher chance to win.

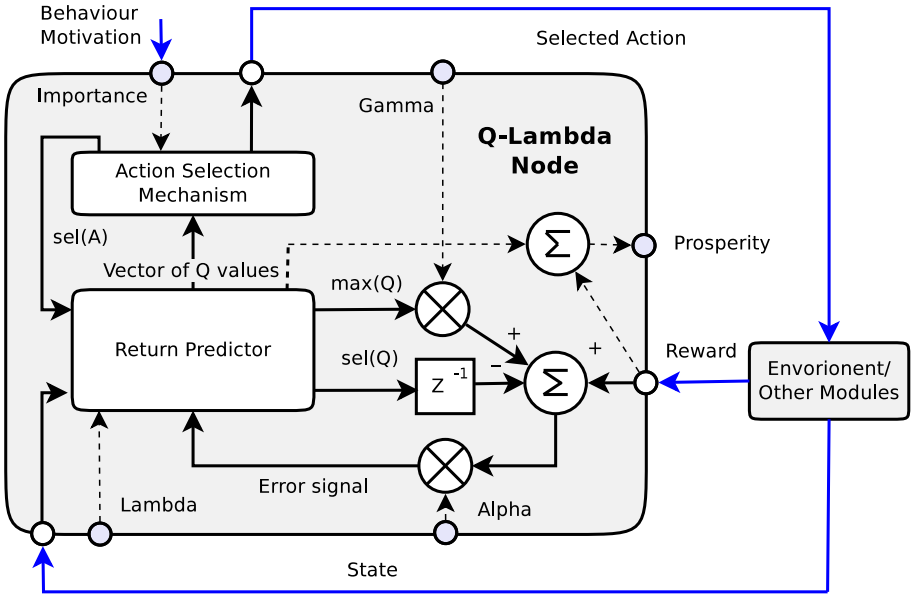

Fig. 2. Scheme of the Q-Learning system. The line labeled "max(Q)" is the prediction of return for the best action. The "Sel(Q)" is the Q actually taken, it is combined with return prediction and reinforcement received from the environment r_i through unit the delay z^{-1}, γ is the discount factor and α is the learning rage. The predictor predicts action values in a current state, based on this information the ASM selects action to be executed. The corresponding Neural Module will have the following configuration inputs: α,γ,$lambda$, data inputs will represent states and reward and data outputs will represent actions to be executed.

Defining Prosperity of Q-Learning Neural Module: The single value of Prosperity for this module can be difficult to choose. In our implementation, the prosperity of the module is composed of two values as follows:

$$P_t = \frac{Cover_t + MCR_t}{2}, \tag{5}$$

Where, Mean Cumulative Reward (MCR) is defined as mean reward (R) received during the simulation:

$$MCR_t = \frac{\sum_i R_i}{i} \ \forall i \in 0..t, \tag{6}$$

and the $Cover_t$ represents how many states has been visited (by the RL module) during the simulation.

Motivation Source Module Design

The only change in the motivation source implementation (compared to the theory) is that the Module produces two data outputs: **The motivation** for the behaviour and **the reward** received on its input. The reward output serves mainly for simpler connecting of modules in the simulator.

The prosperity of this module should correspond to the value of physiological variable V, defined in the equation 3. The *limbo* represents the optimal conditions of agents physiology. If the physiological state space is in the limbo area, no motivation is produced Kadleček (2008). Here the limbo area is in $V = 1$. The *Mean State Distance* (MSD) to optimal conditions (limbo area) is defined as:

$$MSD_t = \frac{\sum_i SD_i}{i} \ \forall i \in 0..t, \tag{7}$$

where SD_i is state distance from the optimal conditions.

$$P_t = 1 - MSD_t. \tag{8}$$

The more time spent near the optimal conditions, the better agents behaviour probably is. Therefore the Prosperity of the Motivation source is defined in the equation (8).

EXPERIMENTS

The resulting modules were tested in an architecture composed of one Module with motivation source and one Q-lambda module. The importance input and the reward input of the Q-lambda module was connected to the Motivation source module. Connecting the motivation to the importance input causes that the **action selection is dynamically weighted between the greedy and randomized** one.

Experiment Description

The architecture was tested on discrete grid map of size 20×20 with obstacles and one attractor, the environment is described in the Fig.3. The agent was equipped with 4 actions (moving in four directions) and the reward was received after reaching the position containing the reward.

Concluded simulations are made as non-episodical, this means that the agent starts on initial position (center of the map) and is let to interact with the environment freely for given number of steps.

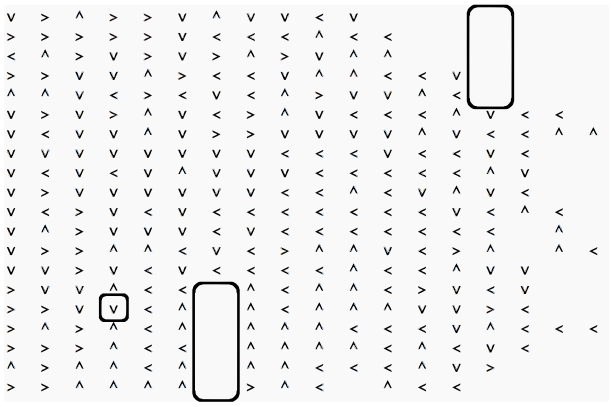

Fig. 3. Simple 2D environment representation. Agent has ability to execute four actions (moving in four directions). Each position in the map contains symbol representing action with the highest $Q(s,a)$ value learned. When the agent follows the greedy policy (Greedy ASM), these actions will be used. In the map, there are two obstacles and one attractor.

Validating Functionality of Learning Algorithm

The presented values of prosperity are presented from 10 non-episodical experiments, each started from the same initial state (center of the map) and lasted 100000 discrete steps. The RL algorithm was configured with the following empirically-estimated constant parameters: $\alpha = 0.5$, $\gamma = 0.3$, $\lambda = 0.04$.

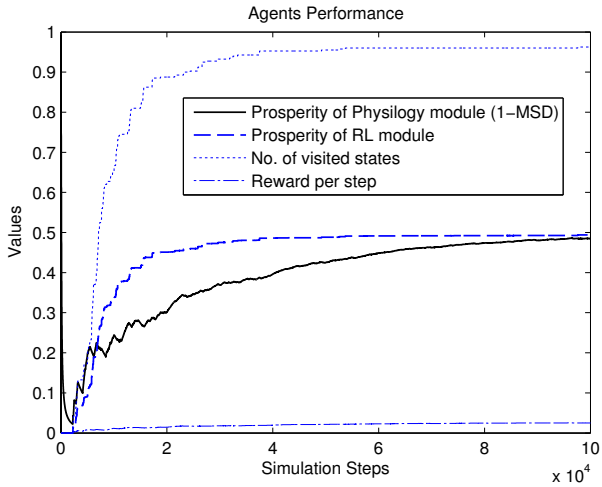

Fig. 4. Typical course of Prosperities of particular Neural Modules in the architecture. The complete Prosperity value is composed of prosperities of the Q-Lamda Module and the Physiology Module. The value should represent how suitable is overall architecture design for a given task. In this case, the value also should represent the convergence of learning. We can see that despite the value of Reward/step is small, the agent learns to maintain its physiology in reasonable distance from optimal conditions. It can be also seen that the agent explores the environment efficiently and it has visited almost all states (except obstacles).

The prosperity of architecture is defined as a sum of prosperities of all Neural Modules used:

$$P_{architecture} = \frac{\sum_i P_i}{i} \; \forall i \in architecture, \qquad (9)$$

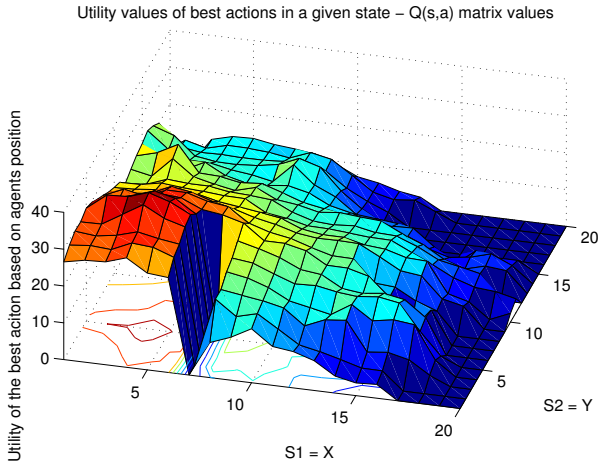

Fig. 5. Value of the highest utility (of the best action a^*) in the $Q(s,a)$ matrix, based on agents X, Y position in the map. The nearer to the reward source, the higher the expected outcome of the best action is. Positions with obstacles have value of $Utility = 0$. Note that the Greedy ASM selects the action with the highest utility in each state. For better visibility, values on the Z axis are rescaled from the interval $\langle 0, 1 \rangle$. These values represent utilities of actions depicted in the Fig.3.

where P_i is the prosperity of one module. Here, the Q-Lambda and the Motivation source Modules were used. The value should represent how suitable is overall architecture design for a given task. In this case, the Prosperity should also represent the convergence of agents learning.

Note that the Prosperity of Q-Lambda module is composed of MCR and $Coverage$ values. Therefore its value represents how often the module (agent) receives the reward as well as how many states it has covered. The value of the Motivation source is the MSD, which represents how "satisfied" is the agent with its behavior.

The Fig.3 is a graphical representation of the best (learned) action in the state (X, Y coordinates). This represents agents autonomously learned knowledge during 10000 simulation steps, this strategy would be followed in case that the Greedy ASM was used. The Fig.5 depicts values in the $Q(s,a)$ matrix. The highest Utility value for each state (X, Y coordinates) is depicted, these values correspond to the actions shown in the Fig.3.

Validating the Purpose of Motivation

During the simulation, the amount of motivation determines how important is learned behaviour. After obtaining the reward, the motivation decreases towards zero and the agent starts to perform the exploration. The value of the motivation should directly correlate to the agents policy. In order to test this, the speed of decay of the physiological variable (see the equation (3)) was altered and resulting agents behaviour was observed.

In case when the decay of the variable is fast, the agent should "switch" to the exploitation of learned knowledge frequently. After satisfying the motivation, the agent returns to the exploration.

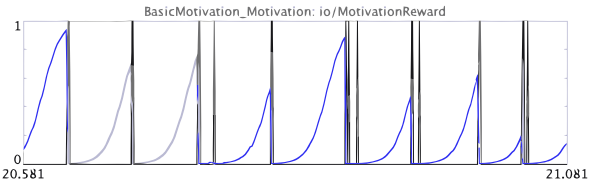

Fig. 6. Motivation value in case when the motivation increases fast. The X axis represents time steps and the Y represents value of particular variable. Spikes in the graph represent binary event of receiving the reward. Continuous value (sigmoidal curves) represents the amount of motivation. It can be seen that the motivation is source of exploitation of behaviour which leads to the reward.

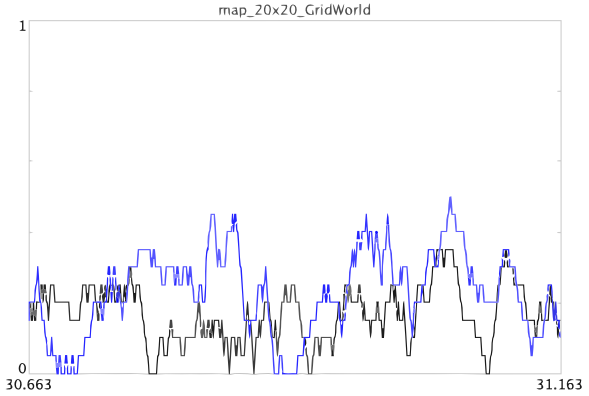

Fig. 7. Agents position in the map in time, which corresponds to the fast increasing motivation. The X axis represents time steps and the Y axis represents agents current position. Two lines represent the X and Y position in the map. Note that values of state variables are sampled from interval of $\langle 0, 1\rangle$ and the source of reward is located on coordinates $\langle 3, 4\rangle$. It can be seen that the agent tends to stay near the reward source most of the time.

The Fig.6 shows the case when the $decay = 0.01$ causes relatively fast increase of motivation. When the motivation is low, the agent explores the environment. As the motivation increases, gradually less randomization is used until the agent reaches the reward. This sets the motivation back towards zero. The Fig.7 roughly corresponds the the Fig.6 and represents the agents position in map during the simulation. It can be seen that the agent explores only states that are near the motivation source.

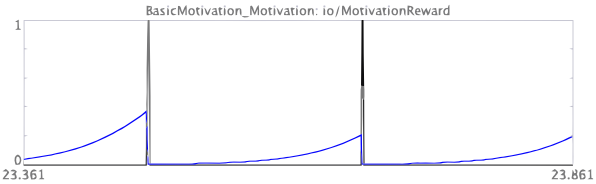

Fig. 8. Motivation value increases slowly in this case, therefore the agent has "more time" for exploration. The physiology is satisfied (by obtaining the reward) only when needed.

In the following experiment, the value of $decay = 0.002$ causes slower increase of motivation. The Figures 8 and 9 show agents behaviour in this case. It can be seen that the agent has less overall motivation to obtain the reward and exploitation of the behaviour is less important. The Fig.9 shows that the agent explored bigger part of the environment while satisfying the motivation when necessary.

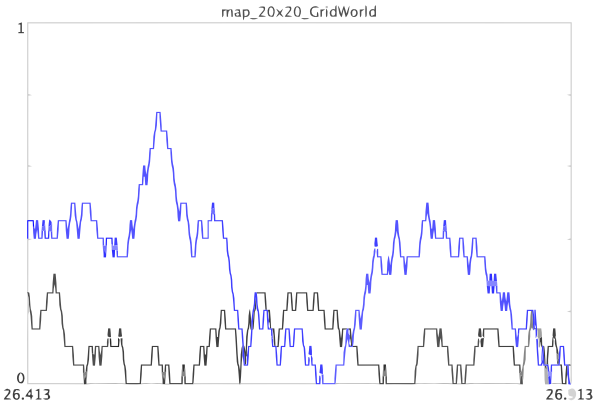

Fig. 9. Agents position in the map in time. Compared to the Fig.7, here the motivation increases slower. In this case, the agent has more time to explore and therefore reaches more distant parts of the environment too.

CONCLUSIONS

Here, the two reusable Neural Modules were presented, one implementing the Q-Learning algorithm and the second implementing the source of motivation. The functionality of these modules was experimentally tested. The results show that the agent "test-architecture" composed of these modules is able to successfully learn in discrete non-episodic experiments (Fig.3,4,5). The expected oscillations between exploration and exploitation behaviour were observed (Fig.6,8). This implies that the architecture was able to dynamically switch between the knowledge exploration and exploitation as needed (Fig.7,9).

These presented modules are compatible with our HANNS framework, but can be also used as stand-alone ROS nodes. The Java implementation of these nodes is freely available online (together with the environment) at https://github.com/jvitku/rl, and the Motivation source available at: https://github.com/jvitku/physiology.

The presented Neural Modules are made as domain-independent as possible and therefore may be directly incorporated in new architectures of autonomous agents, such as those proposed in Kadleček (2008) or Vítků (2011). Furthermore, the optimization techniques can be applied in order to build new architectures for a particular task. By means of techniques similar to the EAs, the modules can be used in searching for entirely new agent architectures.

ACKNOWLEDGEMENT

This research has been funded by the Dept. of Cybernetics, Faculty of Electrical Engineering, Czech Technical University in Prague, under the SGS Project SGS14/144/OHK3/2T/13.

REFERENCES

Auda, G. and Kamel, M. (1999). Modular neural networks: a survey. *Int J Neural Syst*, 9(2):129–151.

Deb, K. (2011). Multi-objective optimisation using evolutionary algorithms: An introduction. In Wang, L., Ng, A. H. C., and Deb, K., editors, *Multi-objective Evolutionary Optimisation for Product Design and Manufacturing*, pages 3–34. Springer London.

Eliasmith, C. and Anderson, C. H. (2003). *Neural Engineering: Computation, Representation, and Dynamics in Neurobiological Systems*. The MIT press, Cambridge, ISBN: 0-262-05071-4.

Fekiac, J., Zelinka, I., and Burguillo, J. C. (2011). A review of methods for encoding neural network topologies in evolutionary computation. In *Proceedings 25th European Conference on Modelling and Simulation ECMS*, pages 410–416. ISBN: 978-0-9564944-2-9.

Izhikevich, E. M. (2003). Simple model of spiking neurons. *IEEE Transactions of Neural Networks*, 14:1569–1572.

Kadlecek, D. and Nahodil, P. (2008). Adopting animal concepts in hierarchical reinforcement learning and control of intelligent agents. In *Proc. 2nd IEEE RAS & EMBS Int. Conf. Biomedical Robotics and Biomechatronics BioRob 2008*, pages 924–929.

Kadleček, D. (2008). *Motivation driven reinforcement learning and automatic creation of behavior hierarchies*. PhD thesis, Czech Technical University in Prague, FEE. Supervised by Pavel Nahodil.

Kadleček, D. and Nahodil, P. (2001). New hybrid architecture in artificial life simulations. In *Advances in Artificial Life. In Lecture Notes in Artificial Intelligence*, volume 2159, pages 143–146. Berlin: Springer Verlag, Berlin, Germany. ISBN: 978-3-540-42567-0.

Kordík, P. (2006). *Fully automated knowledge extraction using group of adaptive models evolution*. PhD thesis, Czech Technical University in Prague, FEE, Dep. of Comp. Sci. and Computers.

Maass, W. (1996). Networks of spiking neurons: The third generation of neural network models. *In Journal Neural Networks*, 10:1659–1671.

Murre, J. M. J., Phaf, R. H., and Wolters, G. (1989). Calm networks: a modular approach to supervised and unsupervised learning. In *Proc. Int Neural Networks IJCNN. Joint Conf*, pages 649–656.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. (2009). Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software*.

Thomas, D. B. and Luk, W. (2009). Fpga accelerated simulation of biologically plausible spiking neural networks. In *In Proc. IEEE Symp. Field-Programmable Custom Computing Machines (FCCM)*.

Vítků, J. (2011). An artificial creature capable of learning from experience in order to fulfill more complex tasks. Diploma thesis, Czech Technical University in Prague, FEE, Dept. of Cybernetics. Supervised by Pavel Nahodil.

AUTHOR BIOGRAPHIES

JAROSLAV VÍTKŮ was born in Prague, Czech Republic. He graduated in 2011 in Czech Technical University in Prague, Faculty of Electrical Engineering in Artificial Intelligence. His diploma thesis "An Artificial Creature Capable of Learning from Experience in Order to Fulfill More Complex Tasks" was awarded by Price of Dean. Currently, he is a PhD student at the Department of Cybernetics, CTU in Prague, still under the guidance of his supervisor Pavel Nahodil. Here elaborates the results of his thesis as an automated design of complex modular systems inspired by Nature. His research interest includes hybrid neural networks, cognitive science, biologically inspired algorithms, behavioral robotics and Artificial Life in common. Now, he is finishing his dissertation: "Autonomous Design of Modular Agent Architectures". Some interesting parts of the thesis are presented in this paper for the first time. His e-mail address is: vitkujar@fel.cvut.cz.

PAVEL NAHODIL was born in Prague, Czech Republic. Since 1970 he has been working at the Department of Cybernetics at the Faculty of Electrical Engineering, Czech Technical University in Prague, where he was also appointed the Professor in Technical Cybernetics (in 1986). He has led and consulted more than 123 diploma thesis here so far. He was also supervisor of about 30 PhD doctoral students till now. His present professional interest includes artificial intelligence, multi-agent systems, on behavior based intelligence robotics (control systems of artificial creatures) and the artificial life design in general. He is (co-)author of several books, university lecture notes, hundreds of scientific papers and large collection of scientific studies. He is also the organizer of some international conferences + reviewer (IPC Member) and a member of many Editorial Boards. His e-mail address is: nahodil@fel.cvut.cz.