Needs, Pains, and Motivations in Autonomous Agents

Janusz A. Starzyk, *Life Member, IEEE*, James Graham, *Member, IEEE*, and Leszek Puzio

***Abstract*— This paper presents the development of a motivated learning (ML) agent with symbolic I/O. Our earlier work on the ML agent was enhanced, giving it autonomy for interaction with other agents. Specifically, we equipped the agent with drives and pains that establish its motivations to learn how to respond to desired and undesired events and create related abstract goals. The purpose of this paper is to explore the autonomous development of motivations and memory in agents within a simulated environment. The ML agent has been implemented in a virtual environment created within the NeoAxis game engine. Additionally, to illustrate the benefits of an ML-based agent, we compared the performance of our algorithm against various reinforcement learning (RL) algorithms in a dynamic test scenario, and demonstrated that our ML agent learns better than any of the tested RL agents.**

***Index Terms*— Cognitive architectures, embodied intelligence, motivated learning (ML), simulation, virtual agent.**

I. Introduction

A SIGNIFICANT challenge in robotics is to develop autonomous systems that can reason and perform missions in dynamic, uncertain, and uncontrolled environments [1]. Therefore, recent research efforts are directed toward developing autonomous cognitive systems. Reinforcement learning (RL) methods have made significant progress in this direction [2]–[5], and the topic is actively researched in laboratories around the world.

Current cognitive architectures, such as Soar [6], Adaptive Control of Thought—Rational (ACT-R) [7], [8], Icarus [9], [10], Learning Intelligent Distribution Agent (LIDA) [11], Polyscheme [12], Nonaxiomatic Reasoning System (NARS) [13], and Connectionist Learning with Adaptive Rule Induction Online (CLARION) [14], either have to rely on predefined goals [without self-motivated learning (ML)] or predefined rules (without autonomous reasoning). Due to their reliance on predefined scripts and heuristic rules, current robotic systems lack autonomy, self-adaptability, and reasoning capabilities either to accomplish complex missions or to handle ever-changing missions in uncontrolled environments.

One of the best known cognitive architectures, Soar, consists of many useful functional blocks like semantic, procedural, and episodic memory, working memory, symbolic and subsymbolic processing of sensory input data. Semantic, procedural, and episodic memories interact in Soar during each decision cycle when the memory is searched for knowledge relevant to related goals and rewards. Soar uses production rules that help to fetch data from the long-term memory and select an appropriate action. If it has insufficient information to handle a problem, it generates subgoals. However, it lacks grounding (intrinsic understanding of symbols) and its intrinsic motivations are limited to set goals. Attention switching in Soar is limited to the perceptual stage, hence it does not contain real-time regulation of its processes and assumes that it has enough processing power to compute everything that it needs to decide about its action.

ACT-R is part of a family of architectures that is used to model human cognitive processes. However, these architectures are focused primarily on modeling and are not designed toward emulating full intelligence, but rather to test cognitive concepts. A newer version of this architecture, ACT-R/Embodied (ACT-R/E) [8], tries to faithfully model people's behavior as they perceive, think about, and act on the world around them. ACT-R/E hopes that by closely modeling human processing, it will aid in developing agents that are more capable of interacting with humans in an assistive manner. However, ACT-R/E has no mechanism to develop internal motivations.

Icarus, is an adaptive architecture for intelligent physical agents. It shares several similarities with Soar and ACT-R, such as symbolic representation of knowledge, pattern matching, a recognize-act cycle, and incremental learning. It also possesses short and long-term memory structures. Icarus focuses on perception and action but its lack of focus on more complex cognitive capabilities, such as mental attention switching, significantly limits its use.

The LIDA architecture utilizes a "Global Workspace" (based on Baar's Global Workspace Theory) for cognitive processing. LIDA is a comprehensive architecture that models biological cognition from low-level action to high-level reasoning. However, its reliance on somewhat ambiguous "atoms" and "attention codelets" makes it more difficult to implement

Manuscript received January 22, 2015; revised February 14, 2016; accepted July 21, 2016. Date of publication August 17, 2016; date of current version October 16, 2017. This work was supported by the National Science Centre under Grant DEC-2011/03/B/ST7/02518. *(Corresponding author: James Graham.)*

J. A. Starzyk is with the Russ College of Electrical Engineering and Computer Science, Ohio University, Athens, OH 45701 USA, and also with the School of Computer Science and Management, University of Information Technology and Management, Rzeszów 35-225, Poland (e-mail: starzykj@ohio.edu).

J. Graham is with the Russ College of Electrical Engineering and Computer Science, Ohio University, Athens, OH 45701 USA (e-mail: james.graham.1@ohio.edu).

L. Puzio is with the School of Computer Science and Management, University of Information Technology and Management, Rzeszów 35-225, Poland (e-mail: lpuzio@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2016.2596787

than used in this work mechanism based on motivations and mental saccades [15].

Polyscheme tries to integrate multiple inference and representation schemes into a single system capable of cognitive thought. By combining several modules, Polyscheme is a fairly flexible system, capable of choosing its own focus. However, the system is overspecialized. Its modular architecture is good, but the individual modules are overly restricted in their operation, which limits Polyscheme's learning capabilities.

NARS is an embodied situated system that grounds its knowledge in its past experience, uses fuzzy reasoning and fuzzy learning, and operates in real-time. Its tasks are prioritized based on task urgency depending on the system's internal state and task durability. Competing tasks, as well as resource management and resource allocation, depend on the state of the environment. In NARS, intrinsic goals are derived and prioritized by the system. NARS can generate new goals using its decision-making procedure. The decision-making procedure in NARS requires a formal language (with a grammar and semantics) and a set of inference rules.

CLARION combines sensory perception with symbolic representation through neural networks, RL, and higher level declarative knowledge. It uses a motivational system for perception, cognition, and action and uses inductive analysis of its learned knowledge to discover rules. Higher level motivations in CLARION are pretrained using a back propagation neural network. Sun [14] suggests that motivations may be derived through RL. However, no specific mechanism or a practical example illustrating how the learning agent derives its motivations was provided. CLARION tries to capture the interaction between explicit and implicit processes and appears to be focused primarily on the modeling of psychological models.

Embodied cognition is a trend in philosophy that challenges the traditional view that the body has a marginal effect on understanding of cognition and mind [16], [17]. This parallels developments in robotics that claim that intelligence has to be embodied, situated in its environment, and cannot develop without embodiment [18], [19]. Embodied cognitive systems often use RL to develop their abilities to interact with their environment [5], [20]–[23].

Intelligent systems should have some degree of autonomy, and are described in [24] as systems that perform tasks in a diverse environment through learning and adaptation to accomplish high-level goals. Thus, in this paper, we focus on the development of autonomous intelligent systems that implement their goals in a dynamic environment using limited resources. We stress the importance of the system's intrinsic motivations and development and management of internal cognitive goals.

In open-ended tasks that an autonomous intelligent system must perform, resource management is often pointed to as a core problem. In our paper, we provide a motivational development mechanism that can engage such a core problem.

Starzyk *et al.* [25], [26] and Graham *et al.* [27] focused on the idea of motivation as the underlying drive behind the operation of a cognitive agent. The agent has designer specified "primitive" needs and a general motivation to satisfy its needs. As the agent learns how to fulfill its "primitive" needs, it creates new "abstract" needs related to the actions and/or resources needed to fulfill them.

This paper presents a significant extension of the agent's motivation and goal creation scheme by providing the agent with needs to respond to both desired and undesired actions by other agents. We developed a learning mechanism to either encourage or discourage such actions. We also unified bias calculations for desired and undesired resources and actions.

The obtained generalized ML agent is better equipped to recognize desired and undesired situations, and learn how to respond. We illustrate the development of such systems in dynamic, changing environments using virtual agents with symbolic I/O in a simulation environment within the NeoAxis game engine [28] (see Section IV).

Most current autonomous robot systems [29]–[32] aim at developing local behavior control algorithms under heuristic rules, and then seek to emerge global behaviors. However, no clear path is provided that tells the agent how to generalize from local to global behavior. In contrast, our ML approach provides such generalization via a mechanism that creates higher level goals and motivations that are intrinsic to the agent.

Adding the intrinsic motivations developed in ML improves the agent's skills and facilitates learning in dynamically changing environments. To test this hypothesis, we constructed a black-box framework for autonomous learning systems and demonstrated that our ML agent learns better than any of the tested RL agents.

The rest of this paper is organized as follows. In Section II, we discuss the memory organization of an ML agent. Section III describes a generalization of the ML agent capable of dealing with desirable and undesirable resources while managing its own and the environment's actions. We discuss how it learns proper behavior to minimize its pains. Following this, in Section IV, we present an implementation of the ML agent, which includes the improvements specified in this paper. The ML agent is integrated into the NeoAxis simulation environment and operates in an example testing scenario. Section V compares our ML agent with other learning agents in a simplified scenario using our black-box testing approach. In Section VI, we discuss our current results, and present our plans to further advance the ML agents and their virtual simulation tool.

II. Motivated Learning Agent Memory Organization

A higher level goal for the ML system, addressed in our work, is survival—-this includes avoiding hazardous situations in the environment (like highly toxic substances), but also adverse actions by other agents that can hurt the system.

ML systems use artificial curiosity to explore, which is similar to curiosity-based learning used in RL [3], [4], [33]. However, in addition to curiosity-based motivations, ML systems are able to create abstract goals and new motivations to learn efficiently and purposefully. It was argued in the ML

literature [34], [35] that the designer cannot predetermine all goals, and we agree with this argument. However, in these papers, motivations that drive learning and the creation of new goals are specified by the designer. We suggest that an intelligent system cannot have all motivations predetermined by the designer either. A simple argument is that complex motivations may require understanding of complex concepts and the machine must first learn such complex concepts before they are used to trigger complex motivations. Thus, concept learning, goal creation, and motivations are mutually dependent and have to be gradually developed in learning agents.

The ML agent interacts with the virtual environment, changing it by its actions and receiving rewards (external and internal) for its actions. In the current implementation of the ML agent, we assume that both sensory inputs and motor outputs are symbolic, and that they provide an interface to the virtual environment. While this symbolic form of perceptions and actions is not necessary for ML, it simplifies the description of the agent's motivations, goals, and action control.

We assume that the ML agent learns in a hostile environment, where there is either a limited amount of renewable resources that the agent needs or there are hostile characters that may harm the agent, unless it learns to defend itself. An ML agent has a number of predefined basic needs, e.g., desired energy level, comfortable temperature, or acceptable pressure. The agent can satisfy these needs by proper actions. If a need is not satisfied (e.g., energy level is below threshold) the agent feels a pain signal. Thus a pain signal related to basic needs can be predefined as a deviation from desired levels. When the agent learns how to satisfy its need, then an abstract need is introduced related to the requirements for satisfying the need. Unsatisfied need manifests itself by a pain signal and the agent is motivated to reduce this pain.

Needs in our ML system are similar to drives used in principles of synthetic theory (PSI) agents [36]. However, drives in [36] were prespecified by the designer, while in ML the agent develops its own needs and motivations, which makes it more autonomous and a better fit to face various challenges in an open environment than PSI agents. In order to clarify our discussion, let us first define some critical concepts used in our ML.

Definitions: An agent has predefined *needs* (for instance need for shelter, food, or energy). A *primitive pain* is related to each predefined need and is defined as a measure that reflects how far the agent is from satisfying its need. The pain is larger if the degree of satisfaction of a need is lower. The agent acts on its need only if the pain is greater than a prespecified threshold. An agent's *motivations* are to satisfy its needs, therefore, the agent must reduce the associated pains below threshold. Pain reduction in ML is similar to a reward in RL.

We use pain rather than reward for two reasons. Pains lead to efficient management of goals, as dominant pains will be considered first, and the system easily switches between goals.

The second reason is that a system that uses pain reduction acts only when it is needed, while a reward-based system always tries to maximize the reward. We found this concept more convenient for the development of a need-based system, than when a reward is used to stimulate all actions.

To be consistent with this terminology, we convert all drives that the agent may have into pain signals. For instance, if *curiosity* is used to learn, we introduce a curiosity pain signal that generates a low-grade pain that remains until the agent has explored its environment and the actions it can take.

A. Neural Network Implementation of ML

A simple implementation of our ML schema uses a neural network where each sensory neuron represents an object and each motor neuron represents an action. Such symbolic representations have to be learned through a sensory–motor (S–M) approach to object recognition and action learning (known also as symbol grounding), but full discussion of symbol grounding is beyond the scope of this paper.

The ML system's neural network, in addition to sensory S and motor M neurons, contains pain neurons P that register the pain signals, and goal neurons G responsible for pain reduction. Selected pain neurons are connected to the external reward/punishment signals. These neurons receive primitive pain signals that directly increase or decrease their activation level. In ML, abstract pain centers are created through the goal creation mechanism [25], [37] and are activated via an interpretation of sensory inputs through bias signals. All abstract pain neurons have a bias input B that depends on the state of the environment and the preference (bias) of the agent for or against a certain resource or action performed by other actors, referred to here as nonagent characters (NACs).

Definition: A *bias signal* reflects the state of the environment related to a specific resource or event and the agent preference for or against such resource or event.

Definitions: A *goal* is an intended action that involves an S–M pair. To implement a goal the agent acts on the observed object or interacts with another agent. An *abstract pain* is a product of the bias signal and w_{bp} weights.

Reduction of the bias signal reduces the associated abstract pain P and triggers learning, increasing the interconnection weight between corresponding P and G neurons. However, if the dominant pain increases as a result of the selected action, then the interconnection weight between the corresponding P and G neurons is reduced. Learning goals through S–M pairs has been used in robotics in recent years [38].

All pain neurons are initially connected to goal neurons with random interconnection weights. In simple ML, all goal neurons and pain neurons are subject to the winner-take-all (WTA) competition. Once the highest pain above threshold is determined, the winning goal is established by evaluating w_{pg} weights for a given pain. Opportunistic ML (OML) agents [27] use a simple heuristic algorithm rather than a WTA mechanism to determine which goals they will implement. In the symbolic representation, each neuron represents a single symbol, pain, goal, or action. In Fig. 1, pain neurons P with connections to their sensory S, bias B, goal G, and motor M neurons are shown.

A single curiosity neuron is added to indicate the machine's desire to explore. It is triggered by a constant pain value equal

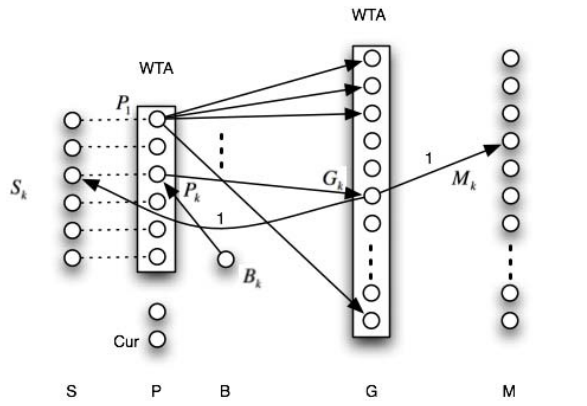

Fig. 1. Connections between sensory, motor, bias, pain, and goal neurons.

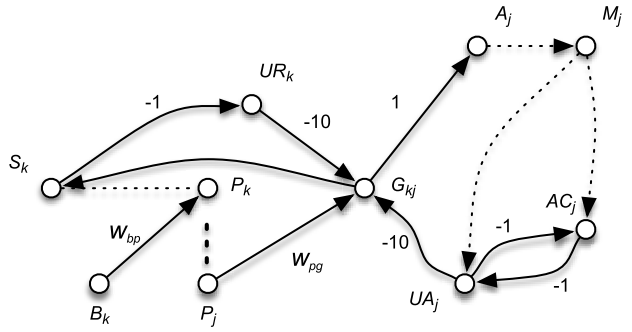

Fig. 2. Trainable connections between pain, bias, and goal neurons.

to a specified threshold. The curiosity pain neuron is linked to all the goal neurons with randomly set weights. As the machine learns, the weights from the curiosity pain to various goals decrease so the curiosity exploration steadily decreases.

G neurons are connected via untrainable weights equal to 1 to corresponding S and M neurons, P and G neurons are fully connected with trainable w_{pg} weights. There are no direct connections from the pain center neurons P to the motor neurons M.

Fig. 2 shows trainable connections between bias, pain, and goal neurons as well as additional inhibitory neurons [*unavailable resource* (UR) and *unsuccessful action* (UA) neurons] that fire depending on the environment conditions. A UR neuron inhibits goal selection if a resource required to perform the action is not observed on the sensory input, while a UA neuron inhibits goal selection when a desired action could not be completed (due to motor malfunction or adverse environment conditions). UA neurons are normally inhibited by an inhibitory link from an *action completed* (AC) neuron, indicating that a motor function can be performed if needed. The AC neuron forms a bistable pair with a UA neuron and only one of them is active at any given time. When a desired action (A) cannot be completed, a UA neuron is activated.

In ML, each time a goal based on selected S–M pair results in the decrease of a dominant pain P_j, there is an increase in the connection weights between this pain neuron and the selected goal neuron G_{kj} (w_{pg} in Fig. 2). In addition, the bias link strength w_{bp} of the abstract pain neuron P_k associated with the desired sensory input S_k is increased. In a similar way, the bias link strength w_{bp} of the abstract pain neuron P_k associated with the desired action A_j is increased. These weight increases gradually establish the need for resources and actions that have helped the agent to reach its goals. At the same time, the lack of these resources or an inability to perform needed actions will become abstract needs (pains) of the learning agent.

The only exception to this is in a curiosity triggered action (when no other pain exceeds the pain threshold). Is this situation, the w_{pg} connection weight is always decreased signifying a decrease in curiosity for this goal. Furthermore, the weights for the other pains are adjusted depending on the effect of the curiosity-based action on these pains.

Section III presents a generalized ML agent, which includes internal motivations to learn how to respond to NAC actions.

III. Generalized ML Agent

The previously implemented ML agent learns how to survive in an environment where resources are limited and can be renewed by the agent's actions. In general, we must consider various situations the agent may face in a hostile environment. Thus, we introduced rules that govern desired and undesired events. In this section, we discuss the desirability of resources and actions, the creation of pains and goals, bias signals, weights, and associated pains, and learning how to respond. All of these rules are general enough to be applied to various environment conditions, and are implemented in the agent simulation algorithm presented in Section IV-A.

A. Desired and Undesired Events

The resources in the environment can be used either by the agent itself to its benefit, or by other agents. In the first case, the agent must learn how to restore these resources, while in the second case the agent may need to learn how to prevent the competing agents from using the resources it needs. The resources can be either desirable, in which case the agent needs to protect or restore them, undesirable, in which case the agent needs to remove or destroy them, or neutral to the agent.

Initially, all resources and NAC actions are unknown to the agent, so their desirability is unknown.

Definition: An agent finds a resource *undesirable* when the resource increases the agent's pain. A resource is found *desirable* once its use reduces the agent's pain.

A more general approach is to consider both desired and undesired events caused either by the agent itself or by other agents. We want the agent to learn how to avoid undesirable events and encourage those that benefit it.

Typically, an event desired by the agent requires it to act on a resource. Any inability to do this is a source of pain to the intelligent agent. One such situation is a lack of the desired resource, but equally important is the inability to perform a desired action. Suppose the agent needs to bring coal to heat its home and has found a huge coal slab. Although it has a desired resource, if it cannot transport it home, its need will not be satisfied. A similar situation is when the agent wants to remove an undesired resource from its environment—it must identify the undesired resource and have the ability to remove it.

TABLE I
AGENT'S LEARNING AND GOAL CREATION PRINCIPLES

Relations	Pain changes	Resource	Abstract pain	Agent's goal
The agent acts on the environment	Reduced	Desirable	Low resource amount or inability to act on the resource	Increase the resource amount
	Increased	Desirable	None	Do not perform this action
		Undesirable	High resource amount	Reduce the resource amount or do not perform this action
	No change	Unknown	None	None
The environment acts on the agent	Reduced	Desirable	Low resource amount or no desired NAC's action	Increase the resource amount or learn how to encourage NAC's action
	Increased	Desirable	Inability to perform defensive action	Learn to prevent the NAC's action
		Undesirable	High resource amount or inability to perform defensive action	Reduce the resource amount or learn to prevent NAC's action
	No change	Unknown	None	None

A slightly different situation is when a NAC instigates an undesired event. In this case, the ML agent can either remove the resource that the NAC agent needs to perform the undesired action (for instance remove his weapon), or prevent the undesired action.

An interesting case is when both desired and undesired events are related to the same resource (or NAC). Consider a beehive. It is desired to have more bees, so the agent can get more honey; on the other hand, the more bees an agent has around the more likely it is that they will sting it, increasing its pain. Removal of bees from the environment will reduce this pain, but this may increase an abstract pain related to the lack of honey. A practical solution, in this case, is to prevent bees from stinging the agent by wearing protective gear.

The agent generates an internal pain signal when it has too few desired resources or too many undesired ones. It also generates an internal pain signal if it cannot perform a desired action or cannot stop an undesired one. Such simple rules for generating intrinsic pain signals are general enough for the agent to develop many abstract goals from those that are needed for its survival—fight its enemies, make friends, or obey a teacher. Table I illustrates the agent–environment interactions and various effects they have on the agent's pains and goals.

Next we describe setting bias signals and adjusting weights associated with pain reduction.

B. Bias Signals, Weights, and Associated Pains

The ML agent is motivated to learn how to minimize its internal pains. We differentiate three types of internal pains: 1) pain based on availability of resources; 2) action-related pain; and 3) pain caused by the inability to perform a desired action.

A bias signal triggers an abstract pain and depends on the perceived situation. Introduction and proper use of bias signals is perhaps the most significant differentiating factor between RL and ML schemes.

1) Resource-Related Pains: If the autonomous agent needs to maintain a resource at a certain level, falling below this level triggers a resource-related pain. To generate this pain, a bias signal that reflects how difficult it is to obtain this resource, is used. The agent must first use the resource to determine if the resource is desired or undesired. The resource bias signal is calculated from the amount of the resource and its desired/undesired status as follows:

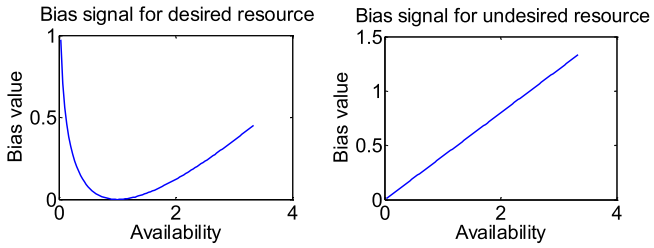

Fig. 3. Bias signal for (a) desired resource and (b) undesired resource.

$$B(s_i) = -(1+\delta_i) * (lnA(s_i) + 1) + 2 * A(s_i) \tag{1}$$

where

$$A(s_i) = \frac{\varepsilon + R_c(s_i)}{\varepsilon + R_d(s_i)} \tag{2}$$

represents availability of the resource s_i, R_c is a current resource amount, and R_d is a desired resource amount. If the resource is undesired R_d is the limit value of the resource perceived as painful to the agent (for instance a painful amount of pollution). ε is a small positive number to prevent numerical overflow, and $\delta_i = 1$ when the resource is *desired*, $\delta_i = -1$ when it is *not desired*, and $\delta_i = 0$ otherwise (when the character of the resource is unknown).

R_d is used as a normalizing factor for the resource level. If $R_c(s_i) = R_d(s_i)$ the corresponding resource pain is zero for a desirable resource, and for an undesirable resource the smaller the $R_c(s_i)$, the smaller the resource pain. Pain reaches significant levels when the agent is out of a desired resource or has plenty of an undesired one.

Fig. 3 shows changes in the bias signal values for desired and undesired resources. Bias value also increases when a desired resource exceeds the desired level. This is a useful feature of the assumed bias function as it penalizes hording.

In special cases, when the resource was found both desired and undesired, we first set $\delta_i = 1$ and learn how to prevent it from harming the agent by proper action. When this fails, we set $\delta_i = -1$.

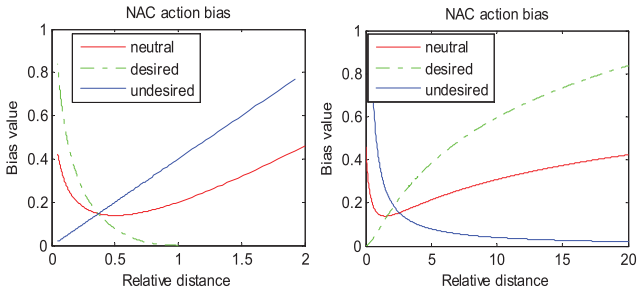

Fig. 4. Bias signal as a function of (a) availability and (b) relative distance.

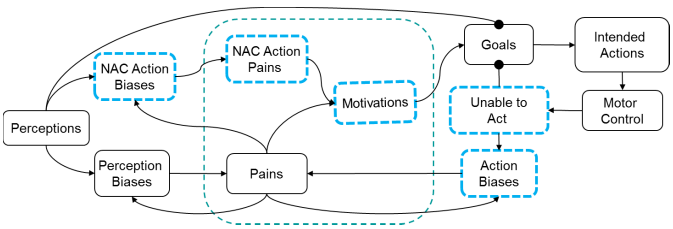

Fig. 5. Block dependencies between perceptions, biases, pains, motivations, goals, and actions.

While other functions can be used to compute biases, we found that function (1) properly correlates availability and desirability of resources and actions, and can be conveniently applied in all situations, as discussed next.

2) Action-Related Pains: Action-related pains are of two kinds—the agent's own action, or the action of a NAC. When the pain increase is a result of the agent's own action, such action must be avoided. When the pain increase is a result of a NAC's action—the agent must learn how to mitigate such an action. The ML agent may need to discourage or encourage other agents to act according to its perception of how useful such actions are. It can do this by introducing biases related to NAC agents' actions. The bias signal is activated whenever the environment conditions are similar to those that caused the pain in the past.

"NAC Action Pain" is first learned when the NAC's action is observed and is correlated to an increased or decreased pain. Subsequently it is triggered by the "NAC Action Bias." Biases that resulted from pain reduction due to a NAC's action are indicated by the link from "Pain" to "NAC Action Biases" and create a need to encourage the NAC to act in the desired way. Bias activation strength is proportional to the reward or punishment received by the agent.

The "NAC Action Bias" is calculated from (1) with $A(s_i)$ representing action availability computed from

$$A(s_i) = \frac{1}{\frac{d_c}{d_d} + \frac{1+\delta_i}{2}} \tag{3}$$

where d_c is current and d_d is a desired (a comfortable) distance to another agent and $\delta_i = 1$ when the NAC action is *desired*, $\delta_i = -1$ when it is *not desired*, and $\delta_i = 0$ otherwise. Fig. 4 shows changes in the bias signal values for desired, undesired, and neutral NAC actions, both as a function of availability and a relative distance d_c/d_d.

Setting the "NAC Action Pain" is important since this motivates the agent to properly respond to a NAC action. This provides the agent with the ability to beneficially interact with other agents. The biases related to a NAC action (shown in Fig. 4 as "NAC Action Biases") are activated through the link from the likelihood of the specific action.

Distance-based availability, computed from (3), is a poor estimate of the likelihood of a NAC action. We used this in our simulation to show that even when using such a simple measure, the agent learns the correct responses to NAC actions. A more elaborate approach would replace $A(s_i)$ with the likelihood that the NAC agent will perform its action.

3) Inability to Act Pains: The third kind of pain—the inability to perform a desired action is generated after the agent learns that an action is useful, but it is unable to perform it in spite of the available resources. This pain may be caused by a motor malfunction or a restriction on the agent's movements. In both cases it produces an abstract pain, which grows in proportion to the need to perform the action. Thus, the third kind of pain depends on the first two, and is triggered by them. However, once it is triggered it does not depend on a current need to obtain a resource or to perform the action.

The bias signal in this case is obtained from

$$B = \gamma * \overline{P(s_i)} \tag{4}$$

where γ is a constant value less than one, and $\overline{P(s_i)}$ is the mean value of the lower level pain that requires the given action to be performed in order to reduce this pain.

Pain biases calculated in this section were set using arbitrary functions. They were introduced to illustrate setting of internal pain signals that represent the motivational state of the agent. Other approaches to compute the internal state of the agent are possible. Fig. 5 schematically shows block dependencies between perceptions, biases, pains motivations, goals, and actions.

Links in Fig. 5 that end with a black dot are inhibitory, while those that end with an arrow are excitatory.

Blocks marked with dashed lines extend the original motivated agent that was only able to create resource-related pains [25], [37]. The new blocks contain perception based NAC Action Bias that correlates the observed action by a NAC and the pain it inflicted on the ML agent. Subsequently, an observed attack will trigger the NAC Action Pain related to the action by the NAC. This, in turn, triggers a motivation block to prevent or stop the attack and the proper action that accomplishes this is learned. "Unable to Act" and "Action Biases" are new functional blocks related to an agent's inability to perform useful actions either to restore/destroy resources or to act defensively.

4) Bias Weight Adjustment: The agent learns the importance of the observed resources or events by adjusting bias weights w_{bp} (shown in Fig. 2) that relate biases and pains. Initially, all bias weights w_{bp} are set to 0. Thus, the machine initially responds only to the primitive pain signals P directly received from the environment. Each time a specific pain P is reduced, the weight w_{bp} of the $B_k - P_k$ bias link increases. However, if the goal activated by the pain center P was completed and did not result in the reduction of pain P,

then the $B_k - P_k$ weights w_{bp} are reduced. Since the bias weight indicates how useful it is to have access to a desired resource S or to be able to perform the selected motor function M, a bias weight adjustment parameter Δ_b must be properly selected to reflect the rate of stimuli applied to a higher order pain center. This rate reflects how often a given abstract pain center P_k was used to reduce the lower order pain signal P.

When a specific goal is not invoked for a long period of time its importance in satisfying a lower level pain is gradually reduced. This requires a reduction of the w_{pg} weight to this goal from all the pain centers. A similar reduction of the w_{bp} weights indicates a gradual decline in importance of an abstract pain. Without such a decay mechanism, the machine can set higher level goals even if they are no longer required to support its lower level needs.

To implement these concepts, the bias weight w_{bp} is computed incrementally based on pain change signals that resulted from the action taken as follows:

$$w_{\text{bp}} = \begin{cases} w_{\text{bp}} + \Delta_{b+} * (\alpha_b - w_{\text{bp}}) \\ \qquad \text{if associated pain changed} \\ w_{\text{bp}} * (1 - \Delta_{b+}) \\ \qquad \text{if there was no change in pain} \\ w_{\text{bp}} * (1 - \Delta_{b-}) \\ \qquad \text{if associated percept was not used} \end{cases} \tag{5}$$

where $\alpha_b = 0.5$, sets the ceiling for w_{bp}; $\Delta_{b-} = 0.0001$, sets the rate of decline for w_{bp} weights; $\Delta_{b+} = 0.08$, sets the rate of increase for w_{bp} weights. These parameters are set experimentally and while they affect the learning speed, the agent behavior is not critically sensitive to these parameters. The bias weights are limited to the $(0, \alpha_b)$ interval.

Using the bias signal, the pain value is estimated from

$$P(s_i) = B(s_i) * w_{\text{bp}}(s_i) \tag{6}$$

where w_{bp} is a bias to pain weight for a given pain center.

C. Learning Recommendations

A motivated agent learns when its actions satisfy one of its needs, which translates to a reduction in one of its pain signals. In previous implementations of ML [26], [27], pain reduction was related to the case of a passive environment in which the agent only used resources to its advantage. Here, we extend this to an active environment, where other NACs may act on the ML agent affecting its pains.

A correct action may not result in the reduction of pain, but instead it may stop the pain's increase. Thus, a successful action cannot be measured by pain reduction alone.

In the general case of an active environment, learning based solely on the agent's own actions may not suffice. In this case, the agent learns how to respond depending on the outcome of interactions with NACs as described in Table II.

In Table II, we have nine distinct situations depending on the action taken and its outcome. We consider three types of actions: agent acts δ_a, NAC acts δ_n, or both agent and NAC act $\delta_a\delta_n$. The outcome of each action can be pain reduction (-1 in pain column), neutral (0 in pain column), and pain increase (1 in pain column).

TABLE II
DETERMINATION OF w_{pg} WEIGHT ADJUSTMENTS

Pain	δ_a	δ_n	$\delta_a\delta_n$
-1	(1,0)	(0,1)	(1,1)
0	(-1,0)	(0,0)	(1,-1)
1	(-1,0)	(0,-1)	(-1,-1)

Pairs (x, y) in Table II indicate *learning recommendations* related to reinforcement and desirability. If $x = 1$ the action taken by the agent should be reinforced by increasing w_{pg} weights, if $x = 0$ no change in w_{pg} weights is recommended, and when $x = -1$ then w_{pg} weights should be reduced.

Desirability of NAC actions is determined based on y values. When $y = 1$ this action is desirable and should be encouraged by the agent, when $y = 0$ the action is neutral and the agent should not care about it, and when $y = -1$ this action is undesirable and should be discouraged.

Recommendations related to NAC actions mean that the agent must learn how to influence the NAC's behavior. This means that any action by the ML agent that results in the desired behavior by NAC will be reinforced.

An interesting case is the second row in Table II, when the pain does not change. If the pain did not change after the agent's own action, such action should be discouraged, since it is unnecessary. If the agent acted in response to an action by a NAC, this action is deemed useful since it is possible that it stopped an "attack" on the agent. This is a special case in which the agent's own action should be encouraged but that of the NAC agent should be discouraged. The reason is that even though the action by the NAC agent would not do any harm, it requires a defensive action by the learning agent, possibly taking away precious time and resources.

D. Fight or Flight

If the NAC acts and "hurts" the agent, the agent must respond to protect itself or its resources. The correct response depends on the agent's ability to observe the NAC's action and prevent it from harming the agent. Thus, detecting an action by the NAC and determining if it is a desired or undesired action is critical.

When the agent detects a NAC action, it activates a potential pain based on the bias signal (1) for the NAC action. This potential pain competes for the agent's attention and motivates it to act defensively. At the same time, the potential pain is not registered as a pain increase, since it is only the motivating factor and not the real pain. Likewise, if the agent acts defensively and stops the attack, it learns the proper action by observing that the real pain did not increase. This lack of pain increase is sufficient reason to reinforce such behavior. However, if the real pain increased, this indicates that the agent did not use a proper response to the attack, and the corresponding w_{pg} weight must go down.

This action-related potential pain signal is removed as the observed NAC action stops. If it stopped as a result of the

agent action, a reinforced learning takes place. If it stopped without the agent's action (for instance when the attacker walks away), the learning agent gradually reduces the potential pain signal.

If the agent is attacked, it can resolve to either defend itself by attacking the enemy or running away from it. The agent can learn which action is a better choice, depending on its experience. The cognitive agent will be able to estimate its chance of running away from the enemy based on its understanding of its own and the enemy's embodiment and limits on its ability to prevent the enemy's action. If, in its own estimate, the agent cannot outrun the NAC, it will fight.

E. Setting Goals

The ML agent sets its goals based on the pain signals and its ability to control them. In a simple neural network implementation of ML, the goal is selected based on the strength of interconnection weights w_{pg} shown in Fig. 2. A given pain signal P is multiplied by w_{pg} weights and a goal neuron with the strongest activation is selected.

To control w_{pg} weights we established the *pain reduction parameter* δ_p based on the reduction or increase of pain and actions performed. δ_p is negative when pain increases, and is positive if the pain decreases. The pain reduction parameter δ_p represents the learning recommendations from Table II and is obtained using the formula

$$\delta_p = 2 * x + y \tag{7}$$

where (x, y) are obtained from Table II. δ_p is between -3 and $+3$. Larger weight is given to the agent's own actions, represented by x, since the agent controls them better than NAC actions.

Initial weights between P–G neurons are randomly selected in the $0-\alpha_g$ interval (a good setting is between 0.49 and 0.51 of α_g for faster learning). Assume that the weights are adjusted upward or downward by a maximum amount μ_g. In order to keep the interconnection weights within prespecified limits $(0 < w_{\text{pg}} < \alpha_g)$, the value of the actual weight adjustment applied can be less than μ_g and is computed as

$$\Delta_a = \mu_g * \operatorname{atan}\left(\frac{ds}{d\hat{s}}\right) * \min(|\alpha_g - w_{\text{pg}}|, w_{\text{pg}}) \text{ where } \alpha_g \le 1 \tag{8}$$

$(ds)/(d\hat{s})$ is the rate of change of lower level resource s as a result of using higher level resource $\hat{s}$ observed by the agent, and

$$\mu_g = \mu_0\left(1 - \frac{2}{\pi}\operatorname{atan}(10/B)\right) \quad \text{where } \mu_0 = 0.3. \tag{9}$$

Using (8) produces weights that slowly saturate toward 0 or α_g (for quick learning set $\mu_0 = \alpha_g/2$) and μ_g gradually changes from μ_0 toward 0 when B changes from a large value toward 0.

If, as a result of the action taken, the pain that triggered this action increased (as determined by pain reduction parameter δ_p), then the w_{pg} weight is decreased, and if the pain decreased, then the w_{pg} weight is increased as follows:

$$w_{\text{pg}} = w_{\text{pg}} + \delta_p * \Delta_a. \tag{10}$$

δ_p is computed from (7) depending on the (x, y) value in Table II. A new pain signal is created to either discourage or encourage the NAC action depending on the y value. The agent learns how to reduce NAC action pain, adjusting corresponding w_{pg} weights.

Additionally, the curiosity weight associated with the action is reduced as

$$w_{\text{pg}_C} = w_{\text{pg}_C} - \Delta_{\text{ac}}. \tag{11}$$

Curiosity helps the machine to explore the environment and learn when not performing any specific pain-based actions.

In addition to decreasing curiosity w_{pg} weight, there is another factor affecting the curiosity goal. We refer to it as "certainty." Certainty C_G is a measure indicating how certain the agent is about a particular goal. For example, if any of the w_{pg} weights associated with a specific goal G approaches 1, then this goal's certainty approaches 1. Conversely, if all the values of w_{pg} weights for a particular goal approach zero, we can say that the certainty for that goal also approaches 1.

When calculating the goal's curiosity value we multiply the w_{pg_C} value by $(1 - C_G)$, since if we are certain about a particular goal, there is no reason to be curious about it.

In sum, certainty C_G is determined from

$$C_G = 1 - \min(\max_P(w_{\text{pg}}), 1 - \max_P(w_{\text{pg}})) \tag{12}$$

and the curiosity-related goal activation strength with consideration of certainty is determined using

$$G_s = P_c * w_{\text{pg}_C} * \min(\max_P(w_{\text{pg}}), 1 - \max_P(w_{\text{pg}})) \tag{13}$$

where P_c is the curiosity pain and w_{pg_C} is curiosity to goal weight.

F. Action Value Determination for OML Agent

The ML agent must choose which of the currently activated goals it must implement first. A simple ML agent chooses the goal with the highest pain value and the highest chance to reduce this pain based on w_{pg} weights. While this simple approach gives the agent autonomy over its goals, it does not yield the optimum results as demonstrated in [27]. In an OML agent the "best action" is determined by the linear heuristic OML model [27], using action "Value" V_i

$$V_i = \frac{P_i + (\Delta P * (t_{\text{mot}} + t_{\text{dist}}))}{(t_{\text{mot}} + t_{\text{dist}})^2} \tag{14}$$

where ΔP is the estimated change in pain. P_i is the pain associated with the action under consideration, t_{mot} is the required motor time to complete the action, and t_{dist} is the travel time to the location where the action will be performed. We generally assume that pains will not change over the course of the action. The action with the highest value of V_i is the one chosen by the OML agent.

IV. Simulation Algorithm of ML Agent in NeoAxis

We use a simulated environment with a virtual robot to develop and test our generalized ML agent and to compare it with other learning agents. First, we created a test bed within the NeoAxis engine [28], generated a virtual environment,

and set its rules. The rules describe how the environment will change when acted upon by the agent or a NAC. Next, we embedded the ML agent in the created environment and provided it with means to observe the environment and to interact with it. The agent receives information about the resources in the environment and NAC actions. The agent has certain needs that are based on the state of the environment. It can satisfy these needs by proper actions that it must learn and use when needed.

The NeoAxis environment can simulate agents in their virtual world. Various types of agents can be developed in the 3ds Max tool by modeling the body, constructing a skeleton and defining agent constraints and then imported into NeoAxis. By using different agent types, we are able to choose the best type of agent for a given task.

The second reason why we utilize NeoAxis is its capable physics engine. Using it we can obtain realistic representation of objects by assigning different properties, static and dynamic parameters like friction, mass, bounciness, hardness, etc. In addition, objects can be attached to one another to create complex structures. It is also possible to create environment rules, i.e., a tree may produce apples at set times, a ball will roll when kicked, or a resource will disappear when used up.

The simulation can host various types of NACs; some of them could be friendly and some hostile, like wild animals. All these result in the creation of a complex, dynamic simulation environment, which reflects realistic situations for an intelligent agent to handle. The ML agent is tested on how well it can respond to various scenarios, and the performance measure we use is either the average pain level or total reward received (when we compare with RL agents).

A. Agent's Simulation in NeoAxis Environment

The basic steps of ML agent simulation algorithm in NeoAxis are as follows.

After initialization, the algorithm performs successive iterations. Each iteration, occurring once per frame (or roughly every 33 ms) consists of the Agent Phase, where the agent observes the Environment, updates its internal state, and generates motor outputs, and the Environment Phase, where the environment performs the agent's actions and updates itself accordingly.

1) Agent phase includes the following.
   a) Relevant environment information is passed to the agent. This information consists of resource object, the resource levels, and their distance from the agent, and information about other NACs, with their goals, distances to resources and to the ML agent.
   b) The bias signals are updated by taking new sensory data into account.
   c) Current pain levels are compared with the pains from the previous cycle in order to determine the pain reduction parameter δ_p.
   d) The bias weights w_{bp} are adjusted according to (5). If the action was curiosity based, no adjustment of the w_{bp} weights is performed.
   e) Pain to goal weights w_{pg} are adjusted using (10) and (11) and are limited to the $(0, \alpha_{\text{g}})$ interval.
   f) All other w_{pgi} $(i \neq 0)$ weights from the selected pain center P to other goals are reduced by
   $$\Delta_i = \Delta_a w_{\text{pgi}} / \sum w_{\text{pgi}}. \tag{15}$$
   g) The system recalculates pains and determines the new goal using (14).
   h) The algorithm determines if the winning goal value is above threshold. If it is not, no action is taken in the next cycle.
2) Environment update phase includes the following.
   a) The current goal involving an S–M pair is implemented. This requires moving the agent to the target and performing the selected motor action.
   b) The environment is changed as a result of this action (e.g., one resource is replenished, while the other, typically the one that the agent used, is depleted).
   c) NACs respond to the agent's action according to their AI rules (for instance a NAC will run away when kicked by the agent).
   d) The sensory data about the state of the environment and NACs actions to be passed to the agent in the next iteration are updated.

In the following section, we discuss how we design and simulate the NeoAxis environment. The performance test is designed to compare autonomous agents' learning in virtual environments.

B. NeoAxis Implementation

We implemented the basic infrastructure of the ML agent in NeoAxis describing the motivated agent functionality in C++ and in C#. The virtual environment for the ML agents built in NeoAxis is a 3-D simulated world governed by realistic physics to present the agents with a dynamic, challenging world. This simulation environment can be separated into two major components. The first component is the animation controller that handles display tasks and transitions the agent from one action to another. The second component processes the agent's behaviors implementing potentially sophisticated rules governing the virtual world in which the agent lives. The agent working in the created environment discovers these rules and learns to use them to its advantage.

In this environment, we created resources that the agent could use (presented in Table III), and we endowed the agent with the ability to act on these resources (agent's motor actions are listed in column 1 of Table III). The agent's actions are driven by pains listed in column 3. Only the pains listed in Table III as primitive pains are predefined. They are hunger, thirst, and sweet tooth that increase over time, bee sting that results from the Bee NAC action, and curiosity. Curiosity pain causes the agent to explore the environment when no other pain is detected and it lasts until all useful actions are

TABLE III
LIST OF RESOURCES, USEFUL RESOURCE-MOTOR PAIRS, AND THEIR OUTCOME

Motor action	Resource name	Agent's pains	Outcome		
			Increase	Decrease	Pain reduced
Eat food from	Bowl	Lack of Bowls		Bowls	Hunger
Drink water from	Cup	Lack of Cups		Cups	Thirst
Eat honey from	Honeycomb	Lack of Honeycombs		Honeycombs	Sweet tooth
Smoke	Cigar	Lack of Cigars		Cigars	Bee sting
Take food from	Fridge	Lack of Fridges	Bowls	Fridges	Lack of Bowls
Pour water from	Bucket	Lack of Buckets	Cups	Buckets	Lack of Cups
Plant	Flowers	Lack of Flowers	Honeycombs	Flowers	Lack of Honeycombs
Buy food with	Money	Lack of Money	Fridges	Money	Lack of Fridges
Pull water from	Well	-	Buckets	-	Lack of Buckets
Buy flowers with	Money	Lack of Money	Flowers	Money	Lack of Flowers
Buy cigars with	Money	Lack of Money	Cigars	Money	Lack of Cigars
Work for money with	Tools	Lack of Tools	Money	Tools	Lack of Money
Study for job with	Book	Lack of Books	Tools	Books	Lack of Tools
Play for joy with	Beach ball		Books		Lack of Books
Kick	Bug	-		Likelihood	Bug eating food
		Hunger - primitive	pain		
		Thirst - primitive	pain		
		Sweet tooth - primitive	pain		
		Bee sting - primitive	pain		
Any		Curiosity - primitive	pain		Curiosity

Fig. 6. Main simulation view with displayed simulation state in windows.

learned. The agent observes which resources it needs and introduces the need to have them according to the goal creation methodology.

We also defined the world rules (listed in Table III as outcome), which describe what the results of the agent's actions are. The agent's actions result in various outcomes like increasing and decreasing resources quantities, as well as in reducing abstract pains. To visualize the resource quantities, the current task, pains levels, and the agent's memory, we added simulation windows, which display the current state of the environment and the ML agent, as shown in Fig. 6. When a pain level is above threshold, it is displayed in red. In this screenshot the agent action is driven by NAC action pain.

The agent tries to learn useful actions and their outcomes. Sometimes the agent performs a nonsense action like "Play for joy with hammer," but even such "useless" actions are used to learn. The memory window on the left part of Fig. 6 displays the memory state of various S–M pairs. When the color is gray, then it means that the agent did not learn the value of the corresponding S–M pair. When the color is white, it means that action is useful for the agent, and if the color is black, the action is known to be useless.

The environment parameters must be properly set to give the ML agent a chance to learn proper behavior. When resources are sparse, the agent does not learn all useful actions because it runs out of resources to test new actions. On the other hand, when resources, are plentiful, the agent does not bother to learn anything new since its pain is under control. Also, when action times are too long, the agent cannot satisfy all of the pains. When proper simulation parameters are selected, the ML agent proves to learn correctly even in a complex and changing environment. We have run multiple simulations where we modified resource quantities and motor action times. By using a human controlled character, we tried to disturb the ML agent by getting in its way or moving resources to different locations. In all of these simulations, the ML agent learned to manage the changes in the environment and minimize its average pain.

C. Simulation Results

Fig. 7 illustrates changes at various pain levels, with particular emphasis on the pain triggered by a NAC action labeled by the solid line as "NAC Action" pain. This pain is related to the consumption of food by the NAC. Food is directly related to the primitive hunger pain (not shown in Fig. 7), so any pain related to food reduction is discovered quickly. Initially, the pain remains below threshold. However, as the NAC consumes food, it causes a buildup of bias against the NAC action (1) and related pain. Eventually the pain passes threshold, and the agent learns to scare the NAC away. As time progresses, Fig. 7 illustrates that the agent correctly interrupts an undesirable

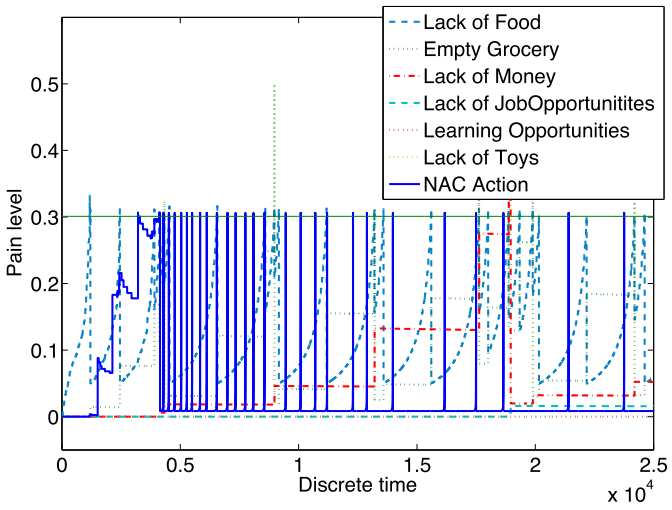

Fig. 7. NAC pain behavior.

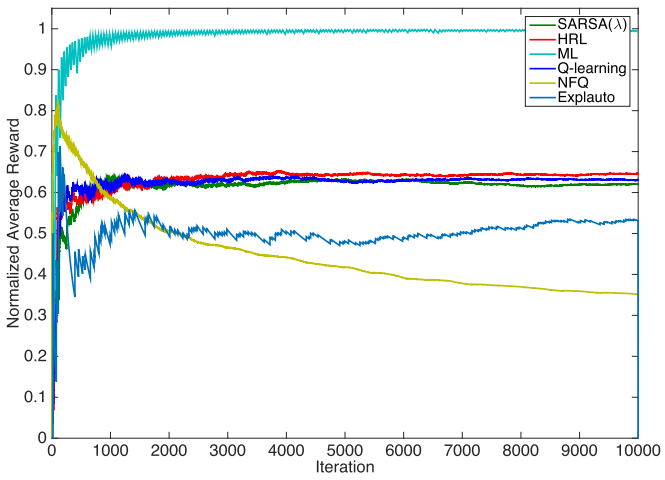

Fig. 8. Comparison of RL algorithm average reward performance with ML.

NAC behavior, and reduces the pain associated with the said behavior by forcing the NAC away.

V. Comparison With RL Agents

An important question that must be answered is whether the proposed ML scheme with goal creation is any better that other schemes derived from RL? Some may argue that RL with subgoal creation will perform equally well as ML, where a subgoal is created when it is needed to complete a designer set goal. However, it is not the same as internal motivations to perform a specific goal. An ML agent will act on its internally set goals independently from the goals set by designer, and in a dynamic environment this results in better performance of the ML agent over the RL agent.

To prove this, we tested several RL agents operating in the same environment using a "black box" approach. The RL agents were given the same information about the environment as the ML agent and could perform the same actions. Agent actions resulted in changes in the environment that corresponded to their actions. Since RL agents optimize the reward received, we computed the rewards received by various agents and normalized them to the maximum reward that could be obtained in this environment at any given time.

The tested environment presents a simple eight-level hierarchy of resources similar in structure to the basic scenario presented in [26], thus it was much simpler than the scenario we implemented in NeoAxis and illustrated in Table III. In this scenario, there is a single primitive need, which can be resolved by the correct action, which consumes a specific resource. This specific resource gets depleted over time and needs a proper action to restore, and so on up to the "top" level of the hierarchy. The top resource level is unlimited.

The settings and basic descriptions of the RL algorithms used to control the agent are as follows.

Q-learning is one of the traditional RL algorithms and can be used to acquire optimal control strategies from delayed rewards. We implemented the Q-learning algorithm from [39] with the discount factor $\gamma = 0.9$ and $\alpha = 0.7$.

State-action-reward-state-action [SARSA(λ)] is a typical temporal difference learning algorithm. It considers transitions from state-action pair to state-action pair. It updates the estimates based on the other learned estimates, without waiting for a final outcome. We implemented this SARSA(λ) algorithm from [21] and set the parameters as $\gamma = 0.9$, $\alpha = 0.4$, and $\lambda = 0.9$.

Maximum Q (MAXQ) is adopted to implement the hierarchical RL from [40], where subgoals and subtasks are defined. By doing this, the agent constructs a set of policies that need to be considered during RL. The MAXQ value function is assumed to represent the value function of any given hierarchy. The parameters are set as $\gamma = 0.9$ and $\alpha = 0.4$.

Neural fitted Q iteration (NFQ) is a variant of the batch RL learning fitted Q iteration algorithm [41], [42]. Defining the learning problem and appropriately adjusting all relevant parameters is often a tedious task. Hence, NFQ implements a dynamic scaling heuristic that can be seamlessly integrated into neural batch RL algorithms, which use a fixed set of *a priori*-known transition samples, e.g., offline learning.

Explauto [43] is a Python library/interface framewor for the implementation of active and online sensorimotor learning algorithms. In our tests with Explauto we used its built-in DiscreteProgress interest model configured to match our environment.

The "black box" scenario regulates the amount of reward that a system can receive using the following equation:

$$R_p(i) = 1 - S_{\text{cp}}/S_{\text{inp}} \tag{16}$$

where S_{inp} is the initial (restoration) level of the primitive resource p and S_{cp} is the resource level at the time the agent performs a valid action on it. By using this approach, the agent will receive a reward ranging from 0 to 1 whenever it performs the correct action, receiving higher values when the need for the resource is greater. At the end of the simulation, results are generated using (17) by averaging the runs and normalizing them, such that the maximum reward any agent can receive

is 1.0

$$R_{\text{Norm}}(i) = \frac{\frac{1}{i}\sum_{p=1}^{n}\sum_{k=1}^{i} R_p(k)}{\sum_{p=1}^{n}\frac{S_{\text{rp}}}{S_{\text{inp}}}} \tag{17}$$

where R_{Norm} is the normalized reward, i indicates the current time step, n is the number of primitive resources, and S_{rp} is the rate at which the rewarding resource p is depleted.

Fig. 8 compares our ML algorithm against the specified RL algorithms. Each algorithm was run 25 times and the results were averaged. The 95% confidence intervals were less than $\pm 2.6\%$ per each line. As expected, hierarchical reinforcement learning (MAXQ) performs better than either Q-learning or SARSA(λ). However, ML clearly outperforms all of them. ML is simply better suited to operate in the hierarchically structured, dynamically changing environments.

These results show that the ML algorithm performs favorably against several common RL algorithms. RL is only using information about the environment state and a "reward" signal, while the ML agent's decision also depends on the internal state of the agent and its intrinsic rewards. A more thorough treatment of our "black-box" testing was given in [44]. The Black box scenario is open to anyone wishing to conduct their own experiments at: http://ncn.wsiz.rzeszow.pl/autonomous-learning-challenge/.

Please feel free to run the experiment yourself and let us know if your results are better than what we reported.

VI. Conclusion

In this paper, we have presented a generalized ML agent. This generalization includes a significant extension of the agent's motivation and goal creation scheme by introducing needs to respond to both desired and undesired actions by NACs. We introduced several modifications to the ML algorithm, including introduction of resource and action desirability and calculations of bias signals and w_{pg} weights. We unified bias calculations for desired and undesired resources and actions. We also presented simple learning recommendations that depend on interactions between the agent, NACs, and the environment. This includes the calculation of pain reduction parameter δ_p. By adding these new features we have improved the agent's ability to handle its environment as well as our own ability to implement complex and interesting environments for our agents.

We simulated the ML agent in a virtual environment using the NeoAxis game engine with a focus to illustrate the new features. The currently implemented agent has several basic primitive motor procedures, which can be linked together to form more complex operations. By rendering low-level motor actions such as grasping or walking into symbolic motor control, we avoid the agent's need to learn these elementary motions. Furthermore, by simulating the agent's environment, we both improve and simplify our control over the learning process. As a result, there are fewer adaptive learning variables and learning takes less time than it would in a real world environment.

It is important to emphasize that our ML agent learns and acts in a real time within its virtual environment. This means that it learns and makes all its decisions within the time frame permitted between iterative updates of the environment (less than 33 ms).

This paper represents a major overhaul of the ML agent, redefining its needs, pains, and motivations to act, which are fundamental to its learning principles. The simulation results of the ML agent in the virtual 3-D environment in NeoAxis prove that our theoretical assumptions for ML agent memory organization, determination of bias signals, weights, and associated pain calculations, were useful. The ML agent was able to learn all environment rules, and keep the agent's pains under control.

Comparison with several RL agents demonstrated the validity of our approach in dynamic environments.

References

[1] H. Hirukawa *et al.*, "Humanoid robotics platforms developed in HRP," *Robot. Auto. Syst.*, vol. 48, no. 4, pp. 165–175, Oct. 2004.

[2] B. Bakker and J. Schmidhuber, "Hierarchical reinforcement learning based on subgoal discovery and subpolicy specialization," in *Proc. 8th Conf. Intell. Auto. Syst.*, Amsterdam, The Netherlands, 2004, pp. 438–445.

[3] P.-Y. Oudeyer, A. Baranes, and F. Kaplan, "Intrinsically motivated exploration for developmental and active sensorimotor learning," in *From Motor Learning to Interaction Learning in Robots*, vol. 264, O. Sigaud and J. Peters, Eds. Berlin, Germany: Springer, 2010, pp. 107–146.

[4] S. Ro, G.-J. M. Kruijff, and H. Jacobsson, "Curiosity-driven acquisition of sensorimotor concepts using memory-based active learning," in *Proc. IEEE Int. Conf. Robot. Biometrics*, Feb. 2009, pp. 665–670.

[5] S. Singh, A. G. Barto, and N. Chentanez, "Intrinsically motivated learning of hierarchical collections of skills," in *Proc. 3rd Int. Conf. Develop. Learn.*, San Diego, CA, USA, 2004, pp. 112–119.

[6] J. E. Laird, "Extending the soar cognitive architecture," in *Proc. Conf. Artif. General Intell.*, Memphis, TN, USA, 2008, pp. 224–235.

[7] J. R. Anderson, D. Bothell, M. D. Byrne, S. Douglass, C. Lebiere, and Y. Qin, "An integrated theory of the mind," *Psychol. Rev.*, vol. 111, no. 4, pp. 1036–1060, Oct. 2004.

[8] J. G. Trafton, L. Hiatt, A. Harrison, F. Tamborello, S. Khemlani, and A. Schultz, "ACT-R/E: An embodied cognitive architecture for human-robot interaction," *J. Human-Robot Interact.*, vol. 2, no. 1, pp. 30–55, 2013.

[9] P. Langley, "An adaptive architecture for physical agents," in *Proc. IEEE/WIC/ACM Int. Conf. Intell. Robot Technol.*, Compiègne, France, Sep. 2005, pp. 18–25.

[10] P. Langley and D. Choi, "A unified cognitive architecture for physical robots," in *Proc. 21st Nat. Conf. Artif. Intell.*, Boston, MA, USA, 2006, pp. 1469–1474.

[11] B. J. Baars and S. Franklin, "Consciousness is computational: The LIDA model of global workspace theory," *Int. J. Mach. Consciousness*, vol. 1, no. 1, pp. 23–32, Jun. 2009.

[12] N. L. Cassimatis, "Polyscheme: A cognitive architecture for integrating multiple representation and inference schemes," Ph.D. dissertation, Dept. Archit. Program Media Arts Sci., Massachusetts Inst. Technol., Cambridge, MA, USA, 2002.

[13] P. Wang, *Non-Axiomatic Logic: A Model of Intelligent Reasoning*. Singapore: World Scientific, 2013.

[14] R. Sun, "The importance of cognitive architectures: An analysis based on CLARION," *J. Experim. Theoretical Artif. Intell.*, vol. 19, no. 2, pp. 159–193, 2007.

[15] J. A. Starzyk and J. Graham, "MLECOG: Motivated learning embodied cognitive architecture," *IEEE Syst. J.*, vol. PP, no. 99, pp. 1–12, 2015, doi: 10.1109/JSYST.2015.2442995.

[16] F. Adams, "Embodied cognition," *Phenomenol. Cognit. Sci.*, vol. 9, no. 4, pp. 619–628, Dec. 2010.

[17] K. Aizawa, "Understanding the embodiment of perception," *J. Philos.*, vol. 104, no. 1, pp. 5–25, Jan. 2007.

[18] R. A. Brooks, "Intelligence without reason," in *Proc. 12th Int. Conf. Artif. Intell.*, Sydney, NSW, Australia, 1991, pp. 569–595.

[19] R. Pfeifer and J. Bongard, *How the Body Shapes the Way We Think: A New View of Intelligence*. Cambridge, MA, USA: MIT Press, 2007.

[20] R. S. Sutton, "Temporal credit assignment in reinforcement learning," Ph.D. dissertation, Dept. Comput. Sci., Univ. Massachusetts, Amherst, MA, USA, 1984.
[21] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 1998.
[22] A. G. Barto and S. Mahadevan, "Recent advances in hierarchical reinforcement learning," *Discrete Event Dyn. Syst.*, vol. 13, no. 4, pp. 341–379, Oct. 2003.
[23] A. G. Barto and M. T. Rosenstein, "Supervised actor-critic reinforcement learning," in *Handbook of Learning and Approximate Dynamic Programming*. Piscataway, NJ, USA: Wiley, 2004, pp. 359–380.
[24] K. R. Thórisson and H. Helgasson, "Cognitive architectures and autonomy: A comparative review," *J. Artif. General Intell.*, vol. 3, no. 2, pp. 1–30, May 2012.
[25] J. A. Starzyk, "Motivation in embodied intelligence," in *Frontiers in Robotics, Automation and Control*. Klagenfurt, Austria: I-Tech, 2008, pp. 83–110.
[26] J. A. Starzyk, J. T. Graham, P. Raif, and A.-H. Tan, "Motivated learning for autonomous robots development," *Cognit. Sci. Res.*, vol. 14, no. 1, pp. 10–25, 2012.
[27] J. Graham, J. A. Starzyk, and D. Jachyra, "Opportunistic behavior in motivated learning agents," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 26, no. 8, pp. 1735–1746, Aug. 2015.
[28] *NeoAxis 3D Engine*, accessed on May 5, 2015. [Online]. Available: http://www.neoaxis.com/
[29] J. Weng *et al.*, "Autonomous mental development by robots and animals," *Science*, vol. 291, no. 5504, pp. 599–600, Jan. 2001.
[30] A. Clark, "Reasons, robots and the extended mind," *Mind Lang.*, vol. 16, no. 2, pp. 121–145, 2001.
[31] A. Farinelli, L. Iocchi, and D. Nardi, "Multirobot systems: A classification focused on coordination," *IEEE Trans. Syst., Man, Cybern. B, Cybern.*, vol. 34, no. 5, pp. 2015–2028, Oct. 2004.
[32] K. Lerman, C. Jones, A. Galstyan, and M. J. Mataríc, "Analysis of dynamic task allocation in multi-robot systems," *Int. J. Robot. Res.*, vol. 25, no. 3, pp. 225–241, Mar. 2006.
[33] J. Schmidhuber, "Developmental robotics, optimal artificial curiosity, creativity, music, and the fine arts," *Connection Sci.*, vol. 18, no. 2, pp. 173–187, Jun. 2006.
[34] P.-Y. Oudeyer, F. Kaplan, and V. V. Hafner, "Intrinsic motivation systems for autonomous mental development," *IEEE Trans. Evol. Comput.*, vol. 11, no. 2, pp. 265–286, Apr. 2007.
[35] K. E. Merrick, "Intrinsic motivation and introspection in reinforcement learning," *IEEE Trans. Auto. Mental Develop.*, vol. 4, no. 4, pp. 315–329, Dec. 2012.
[36] J. Bach, *Principles of Synthetic Intelligence PSI: An Architecture of Motivated Cognition*. London, U.K.: Oxford Univ. Press, 2009.
[37] J. A. Starzyk, "Motivated learning for computational intelligence," in *Computational Modeling and Simulation of Intellect: Current State and Future Perspectives*, B. Igelnik, Ed. Hershey, PA, USA: IGI Publishing, 2011, ch. 11, pp. 265–292.
[38] L. Montesano, M. Lopes, A. Bernardino, and J. Santos-Victor, "Learning object affordances: From sensory–motor coordination to imitation," *IEEE Trans. Robot.*, vol. 24, no. 1, pp. 15–26, Feb. 2008.
[39] T. M. Mitchell, *Machine Learning*. New York, NY, USA: McGraw-Hill, 1997.
[40] T. G. Dietterich, "Hierarchical reinforcement learning with the MAXQ value function decomposition," *J. Artif. Intell. Res.*, vol. 13, pp. 227–303, Nov. 2000.
[41] T. Gabel, C. Lutz, and M. Riedmiller, "Improved neural fitted Q iteration applied to a novel computer gaming and learning benchmark," in *Proc. IEEE Symp. Approx. Dyn. Program. Reinforcement Learn.*, Paris, France, Apr. 2011, pp. 279–286.
[42] M. Riedmiller, "Neural fitted Q iteration—First experiences with a data efficient neural reinforcement learning method," in *Proc. Eur. Conf. Mach. Learn.*, Porto, Portugal, Oct. 2005, pp. 317–328.
[43] C. Moulin-Frier, P. Rouanet, and P.-Y. Oudeyer, "Explauto: An open-source Python library to study autonomous exploration in developmental robotics," in *Proc. Int. Conf. Develop. Learn.*, Genoa, Italy, Oct. 2014, pp. 171–172.
[44] J. Graham, J. A. Starzyk, Z. Ni, H. He, T.-H. Teng, and A.-H. Tan, "A comparative study between motivated learning and reinforcement learning," in *Proc. Int. Joint Conf. Neural Netw.*, Killarney, Ireland, Jul. 2015, pp. 1–8.

Janusz A. Starzyk (SM'83-LM'16) received the M.S. degree in applied mathematics and the Ph.D. degree in electrical engineering from the Warsaw University of Technology, Warsaw, Poland, and the habilitation degree in electrical engineering from the Silesian University of Technology, Gliwice, Poland.

He was an Assistant Professor with the Institute of Electronics Fundamentals, Warsaw University of Technology. He spent two years as a Post-Doctoral Fellow and a Research Engineer with McMaster University, Hamilton, ON, Canada. Since 1991, he has been a Professor of Electrical Engineering and Computer Science with Ohio University, Athens, OH, USA, and also the Director of the Embodied Intelligence Laboratories. Since 2007, he has been the Head of the Information Systems Applications, University of Information Technology and Management, Rzeszów, Poland. His current research interests include embodied machine intelligence, motivated goal-driven learning, self-organizing associative spatiotemporal memories, active learning of sensory-motor interactions, machine consciousness, and autonomous robots and avatars.

James Graham (M'09) received the B.S.E.E. degree in electrical engineering from the University of Michigan, Ann Arbor, MI, USA, in 2001, and the M.S.E.E. degree from Ohio University, Athens, OH, USA, in 2007, where he received the Ph.D. degree from the School of Electrical Engineering and Computer Science in 2016.

He is involved in the development of a cognitive architecture based on motivated learning. His current research interests include cognitive intelligence, machine learning, pattern recognition, and tracking.

Leszek Puzio received the M.S. degree in electrical engineering, automatics, computer science and electronics from the AGH University of Science and Technology, Kraków, Poland, in 2002, and the Ph.D. degree in cybernetics from the Military University of Technology, Warsaw, Poland, in 2008.

He has been an Adjunct Professor with the University of Information Technology and Management, Rzeszów, Poland, since 2008. He was a Post-Doctoral Fellow with the University of Manitoba, Winnipeg, MB, Canada, in 2009. His current research interests include image processing, image features detection and near set-based morphological operations, wavelets, computer graphics and simulations, and artificial intelligence.