HAL open science

A Multi-agent System for Autonomous Control of Game Parameters

Luc Pons, Carole Bernon

▶ **To cite this version:**

Luc Pons, Carole Bernon. A Multi-agent System for Autonomous Control of Game Parameters. IEEE International Conference on Systems, Man and Cybernetics (SMC 2013), IEEE, Oct 2013, Manchester, United Kingdom. pp.583-588, 10.1109/SMC.2013.105 . hal-01147327

HAL Id: hal-01147327

https://hal.science/hal-01147327v1

Submitted on 30 Apr 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Open Archive TOULOUSE Archive Ouverte (OATAO)
OATAO is an open access repository that collects the work of Toulouse researchers and makes it freely available over the web where possible.

This is an author-deposited version published in : http://oatao.univ-toulouse.fr/
Eprints ID : 12835

To link to this article : DOI :10.1109/SMC.2013.105
URL : http://dx.doi.org/10.1109/SMC.2013.105

To cite this version : Pons, Luc and Bernon, Carole *A Multi-agent System for Autonomous Control of Game Parameters*. (2013) In: IEEE International Conference on Systems, Man and Cybernetics - SMC 2013, 13 October 2013 - 16 October 2013 (Manchester, United Kingdom).

Any correspondance concerning this service should be sent to the repository administrator: staff-oatao@listes-diff.inp-toulouse.fr

A Multi-Agent System for Autonomous Control of Game Parameters

Luc Pons, Carole Bernon
IRIT, University of Toulouse
Toulouse, France
{Luc.Pons,Carole.Bernon}@irit.fr

***Abstract*—Control of game parameters to reach domain-related objectives cannot be easily handled with classic control theory approaches. Given the dynamics and complexity of modern game engines, diversity of human players and their constantly changing nature, this paper advocates for means to tune game parameters in real time, with no use of game or users models. The proposed approach, based on a multi-agent system, is used to control two dynamic systems before analyzing the results.**

I. Introduction

In recent years, video games engines have become more and more complex. They can now aggregate many different concepts, as well as many rules defining what the game-play consists in, and what are the tasks players have to perform. Mechanisms and concepts of game engines are parametrized, and an important part of the designer's job is to get from the game-play idea to actual parameter values. The increasing amount of possibilities offered by game engines makes it difficult to properly implement a game that will match ideas of designers [1].

Additionally, the ability of a game to adapt autonomously to players becomes a requirement. This is especially true in games having a more serious aspect, such as military training or professional education, where the experience should constantly match players' skills and abilities to be optimally efficient [2], [3], [4].

Automated parameter adjustment is a candidate solution for answering these points [5]. In the field of control, proportional-integral-derivative (PID) controllers have been used with success to determine appropriate policies on system inputs to match objectives on their outputs [6]. However, PID controllers generally connect one input to one output in a linear fashion, making them inefficient in the control of more complex systems.

Model Predictive Control (MPC) has then been proposed to deal with non-linear systems, incorporating time delays, and high order dynamics. A model of the system to be controlled is needed in order to anticipate the modification to apply on the control policy.

However there are several major challenges to overcome before offering a satisfactory solution. Video games, just like simulations, can present a high degree of complexity. With many autonomous interacting entities, as well as time-delayed interactions and multi-objective tasks for players to perform, it may be difficult, if not impossible, to predict the consequences of an arbitrary change in the parameters, and thus, to create an appropriate model.

In such cases, another possible approach is to use the knowledge of domain experts to build an appropriate control policy. Expert systems consist of a rule base, that associates actions with specific situations, as well as mechanisms observing the current state of the controlled system [7]. Conveniently, this approach does not necessitate a model of the controlled system, nor a deep understanding of its inner dynamics. Various improvements have been made over the years, such as the addition of fuzzy predicates in the rule base [8]. This however requires a complete knowledge of the repairing actions when the system does not satisfy objectives. Such knowledge is generally not available in complex interactive systems like games, especially since a human is constantly interacting with the game.

Interaction makes controlling the game engine even more difficult. Human beings and conditions they are in cannot be efficiently modeled, and what seems difficult for someone may be simple for someone else [9]. Modification of parameters' values can have distinct consequences depending on which human is interacting with the game. Besides, humans are by nature changing, and when they learn something new, a new strategy must be adopted in order to efficiently control the parameters of the game engine.

Moreover, the inner dynamics of video games change over time and an observed property of the game system may not remain true at some other point in time.

Other artificial intelligence techniques have been used to overcome these limitations. Neural networks have been advocated when no model of the controlled system is available, and when no efficient control policy is known in advance.

Neural networks (NN) can approximate any non linear system, given that they receive appropriate training feedback. When controlling a system S with an input I, its output can be given to a neural network. The network can then be trained to find the previously given input I. In other words, the NN is trained to find the inverse model of S [10], [11]. Another possibility is to train the NN to find a predictive model of the control system. But whatever the use of the NN, a training phase with appropriate data is always needed. Data need to be available in a sufficient quantity, and should be representative enough so the NN can generalize to novel situations. Given the diversity of human players, such data may not be available and the cost of setting up a training phase for the NN may

prevent one from applying this technique for controlling game engines.

From these considerations, we advocate the need for a system able to control parameters of a game in a real-time and autonomous fashion, without making use of a model of the game being controlled, nor of users playing the game.

The remainder of this paper is structured as follows. The concepts used are first defined before detailing why and how the control system is designed as a self-organizing multi-agent system. Sections IV and V then apply the proposed control on two different kinds of application: a prey-predator system and a video game. Finally, an analysis concludes the paper.

II. Problem Definition

In order to apply a control policy on input parameters of systems like video games, the parameters manipulated by this control system, as well as the objectives that need to be satisfied, have to be defined first. This section also describes the terminology used throughout the paper, as well as the concepts that need to be understood by any domain expert wishing to apply the approach.

A. Parameters and Measures

When building a parameter adaptation system, the first step consists in identifying all the parameters it is possible to adjust. Each of these parameters needs to be uniquely identified and associated with a range defining the values allowed for this specific parameter. The set of all manipulable parameters *IN* constitutes the input of the system to be controlled; it is defined as follows:

$$IN = \{p_1, ..., p_m\}; p_i \in [p_{i_{min}}, p_{i_{max}}] \subset \mathbb{R}$$

Parameter adjustment aims at influencing the state of the game engine in certain directions. In addition to the input parameters previously defined, such a state is characterized by observations. Observations can either be emerging properties of the game engine, or they can result from interactions between human players and the game engine. In either case, they cannot be directly inferred from the values of the input parameters. Moreover, some games may have some stochastic aspects, making observations mandatory to determine their current state. Each atomic observation of the game must be identified and associated with a range. The set of these observable values is defined as the output set *OUT* as follows:

$$OUT = \{o_1, ..., o_n\}; o_i \in [o_{i_{min}}, o_{i_{max}}]$$

B. Objectives and Constraints

The objective of the control system then needs to be defined. Objectives of game designers are often multiple and contradictory. Instead of defining a general state of the game as an objective, several independent objectives are here considered, each one related to an atomic observation of the game engine.

Human requirements on artificial systems can be hard to express. Humans often have a general idea of their expectations that cannot be easily expressed in a numerical form, more suitable for an intelligent control system. Solutions with fuzzy predicates have been proposed [12]. Fuzzy predicates allow the definition of properties that are not necessarily true or false, but can take instead any intermediate value.

This paper proposes the use of similar mechanisms. An objective involving a specific measure on the game engine should not be necessarily satisfied or unsatisfied. For that purpose, satisfaction functions are introduced. A satisfaction function takes the value of an output, and yields a satisfaction value, arbitrarily chosen within the $[0, 100]$ interval, 100 being the state of complete objective satisfaction.

Depending on the "shape" of the function, an objective can be strict, with a high satisfaction only on a single value, or it can be loose, with a satisfaction slowly decreasing as the value strays from the objective.

The set of all objectives *OBJ* is defined as follows:

$$OBJ = \{obj_1, ..., obj_n\}; obj_i\colon [o_{i_{min}}, o_{i_{max}}] \rightarrow [0, 100]$$

In addition to the objectives to be satisfied, a control system should consider a set of constraints on the inputs. Each input has a range of validity for its value, but there can be some additional constraints, reflecting specific requirements from the domain. In this paper, constraints on input parameters are expressed similarly to objectives, that is, with the use of satisfaction functions.

For each input parameter, a satisfaction function can be defined, mapping all possible values to a degree of satisfaction. The set of all constraints *CONS* is defined as follows:

$$CONS = \{cons_1, ..., cons_m\}; cons_i\colon [p_{i_{min}}, p_{i_{max}}] \rightarrow [0, 100]$$

The aim of the parameter adjustment is to find the values of *IN* that maximize all the time the satisfaction of both *OBJ* and *CONS*. In other words, it is to tune all the parameters so all the constraints and objectives are satisfied.

C. Correlations between Parameters

To dynamically control the target system, designers of the game engine have to express the knowledge they possess about its dynamics. More precisely, they are asked to list the correlations that may exist between input parameters and output measures. A correlation is defined between an input I and an output O if a variation of I is likely to trigger a variation of O. A correlation is said to be positive if an increase (resp. decrease) of I is likely to trigger an increase (resp. decrease) of O; otherwise, it is said negative.

This definition only considers monotonic correlations. Depending on the game engine, there may be non-monotonic correlations between outputs and inputs. In such cases, these non-monotonic correlations are neglected, and the approach is solely based on identified monotonic correlations. The implications of this choice are discussed in more details in section VI.

III. Solution Design

The control system proposed here is based on a Multi-Agent System (MAS) which aims at dynamically tuning all the parameters' values of the system controlled.

An agent is an autonomous entity which evolves in an environment from which it has only a local and incomplete perception. An agent possesses skills that enable it to carry on a behavior which is generally dictated by a local objective.

A Multi-Agent System is a system composed of a set of interacting agents [13]. Two levels can then be distinguished: the lower level – or agent level – involving agents taken individually, and the global level, where the MAS is considered as a whole, a collective. The behavior of this collective is generally something more than the individual behaviors. Thanks to its logical distribution, a MAS can deal with problems composed of a large number of interacting sub-problems and allows a simpler modeling of the domain. Furthermore, interactions between agents can give birth to emergent phenomena (e.g., patterns, organizations, behaviors) at the global level which make MAS interesting for dealing with problems for which no algorithmic solutions can be given in advance and therefore have to be designed in a bottom-up way.

This property of emergence is used here since we consider the control of a dynamic and open system as a complex problem for which no specific solution exists yet. The control system presented in this paper is therefore based on Adaptive Multi-Agent Systems (or AMAS [14]) in which self-organization principles are used [15] to make the collective behavior emerge from local ones. The engine of this self-organization is the ability an agent has to always try and help the agent it considers as being the least satisfied in the system (which maybe itself in some cases); in this sense, agents in an AMAS are said to have a "cooperative attitude".

This section presents the agents involved in the control system, their interactions and how they deal with time delays that have to be taken into account during the control process.

A. Agent Design

The relevant concepts identified in section II which represent both the nature of entities composing the system, on an abstract level, and the raw knowledge on its dynamics are mapped onto several agents:

- Input-agents represent the input parameters of the target system $p_i \in IN$. An input-agent is aware of the bounds $\{p_{i_{min}}, p_{i_{max}}\}$ of the parameter it is modeling, and possesses the ability to effectively change its value.
- Output-agents represent the outputs of the target system $o_i \in OUT$, i.e. single observations of the interactions of a human user with the interactive system. An output-agent is given means to update its value while the system activity is taking place.
- Objective-agents are finally introduced to represent requirements expressed by domain experts, i.e. the set of objectives OBJ, expressed with satisfaction functions. The expression of the global solution of the global problem (i.e. an appropriate control) is here distributed among all the objective-agents. Therefore, one agent expresses a single requirement, involving only a small part of the target system, regardless of the state of the rest of it.

To solve the problem in a distributed fashion, not only agents need to have appropriate behaviors, but they also need to have appropriate communication skills. For that purpose, all agents can make use of a message mechanism, allowing them to send messages to agents they are aware of. Next section details how agents behave.

B. Solving Process Principle

In the design of the global function of a multi-agent system, the focus is being set on isolated parts of the problem. Situations in which agents are going to be are considered, and an appropriate behavior is designed so a simpler problem is solved among a few selected agents. The solving process is then distributed among all the agents modeling the problem. Therefore, one key aspect of the proposed solution is the definition of the agent behaviors, described hereafter.

1) Objective-agents: Objective-agents are the ones that trigger the activity in the whole system. Their responsibility is to compute a satisfaction value for either an input or an output of the target system (represented by an output-agent, called "relative" agent hereafter). An objective-agent has a rather simple behavior: it observes its "relative" agent and uses its own satisfaction function to compute a satisfaction value.

Its goal is to reach a state in which an optimal satisfaction value is computed. To do so, it sends a message to its "relative" agent, requesting this latter to modify its own value in a proper way; this message contains only the current satisfaction value of the sender and the requested variation sign (either positive or negative).

2) Output-agents: Output-agents monitor the value of the outputs of the target system. An output-agent may receive one or several requests from objective-agents. When several requests are received, it adopts a cooperative attitude by tending to help the least satisfied agent it is aware of. Therefore it selects the request with the lowest satisfaction value, and aims at helping its sender.

Since an output-agent has no control on its value, it cannot modify it freely and it uses the information about the domain that experts expressed through a set of correlations (see section II-C) for selecting all the agents it is correlated to. According to the sign of their correlation, it sends them a message and therefore propagates the need for help of the initial request sender.

3) Input-agents: Input-agents are at the bottom of the chain and act when they receive a request from other agents. An input-agent uses received requests to determine which is the least satisfied agent among all the requests senders, and in which way its own value should be modified.

Being cooperative, an input-agent modifies its value in order to help the agent it believes to be the least satisfied. For efficiently exploring the potential search space for this value, an input-agent uses a component named Adaptive Value Tracker (AVT) [16].

C. Delayed Perception Management

One major problem in the application of this solving process among agents, is the potential delay in the effects of parameter modification. When an agent R receives a request from an agent S, it may act accordingly, in order to help the request sender, but the consequences of the action of agent R may not be directly perceived, resulting in agent S keeping on sending requests to R even though appropriate actions have already been taken. This may result in oscillations around the optimal value.

In order to prevent the delay problem, we suggest that each agent R receiving a request builds a representation of its sender S composed of:

- an optimal value O
- a delay value Δ, representing the time needed for consequences of the action of R to be perceived by S.

This representation is constantly updated as new requests are sent, and may not reflect the reality. It is just used to enhance the behavior of requests receivers by making them independant of the delays between each other.

The representation of the optimal value O and the delay Δ allow the input agent to prevent oscillations. The two next sections detail how Δ is used, and how it is measured.

1) Delay utilization to find the optimal value: When a receiver R receives a request $r1$ from a sender S, it first examines the expected variation sign $v1$ contained in the request: either *positive* or *negative* (positive in the case of figure 1). According to $v1$, O is increased/decreased to a new value (dotted line on figure 1).

The agent value (solid line in figure 1) is then modified in the next time steps towards O. When R's actual value reaches O, a timer is triggered. The reason for this is that if the value O is correct, and if the delay Δ is correct, then once the value O is reached, a contradictory request should be received after the delay Δ. Therefore, once the timer is triggered, all requests coming from S with an expected variation equals to $v1$ are ignored, as they are supposed to be caused by some delay.

Once the delay has expired, if requests containing $v1$ are still received (rn in figure 1), then the optimal value O is updated again, and the process is repeated.

2) Delay measurement: Before agents are stabilized, they alternate between increasing and decreasing phases, since their perceptions may not be appropriate right away. When an agent R is in an increasing phase, its value goes up, and it reaches a maximum $m1$ at some point t_{max} in time, before the decision process triggers a decreasing phase (see figure 2). $m1$ is considered to be the worst value of the *increasing phase* since all the agent knows is that it has gone too high.

In the decreasing phase, requests are received with a *negative* expected variation. These requests may, or may not be taken into account, as described in III-C1. Even though the agent is decreasing its value, the satisfaction level contained in the received requests may not be decreasing, due to the delay in the sender perceptions. At some point $t_{c_{max}}$ in time, a request with a minimal satisfaction is received. A link is then made between the maximum value in increasing phase, and the request with the minimum satisfaction in the subsequent decreasing phase.

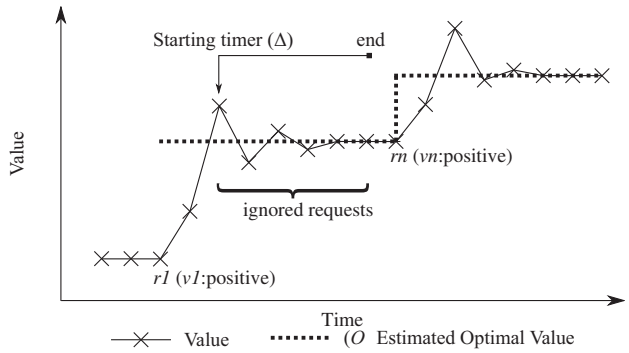

Fig. 1. Delay utilization from the request receiver point of view

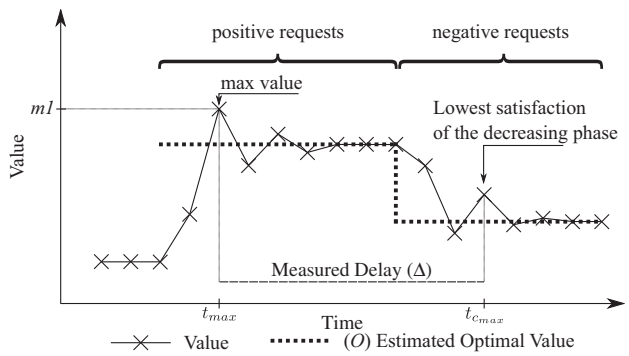

Fig. 2. Delay measurement from the request receiver point of view

Therefore, the delay Δ is obtained with the formula $t_{c_{max}} - t_{max}$ (see figure 2).

IV. Application on a Prey-Predator System

The prey-predator system is an interesting test-bed for the proposed approach. It consists of a dynamical non-linear system modeled by two differential equations, known as the Lotka-Volterra equations. The equations model the evolution of two populations evolving in a common environment: preys and predators. Predators need to consume preys to survive, and preys spontaneously reproduce. The Lotka-Volterra model involves four parameters:

- α : preys reproduction rate
- β : preys death rate due to predators
- δ : predators death rate in absence of preys
- γ : predators reproduction rate according to consumed preys

The population evolution is given by these two equations:

$$\frac{dx(t)}{dt} = x(t)(\alpha - \beta y(t))$$

$$\frac{dy(t)}{dt} = -y(t)(\delta - \gamma x(t))$$

where $x(t)$ is the prey population at time t and $y(t)$ is the predator population at time t.

The description of the system enables us to determine correlations between observables (populations) and parameters (death and reproduction rates) which are given in table I where $+$ means a positive correlation and $-$ means a negative one.

TABLE I. CORRELATIONS BETWEEN OBSERVABLES AND PARAMETERS

	Prey Population	Pred. Population
α	+	
β	+	
δ		-
γ		+

This correlation matrix is a good example of how correlations do not consider side effects. The predator population is positively correlated to the predator reproduction rate, even though a high reproduction rate may lead to a sudden increase in population, and consequently, to prey extinction, and thus, to predator extinction. Such side effects are deliberately neglected, and only direct correlations are listed in the matrix.

The populations are initially set to 10 prey and 10 predator. The aim of the multi-agent system is to dynamically tune the parameters in order so satisfy objectives of both populations: 150 preys and 50 predators. As seen in figure 3, this control first allows both populations to grow. When the number of predators reaches its objective, the population is stabilized and oscillates around 50, while the quantity of preys continues to grow. When the prey population reaches its objective, it stops growing and oscillates around 150. Before step 4000, the model is stable and both objectives are satisfied.

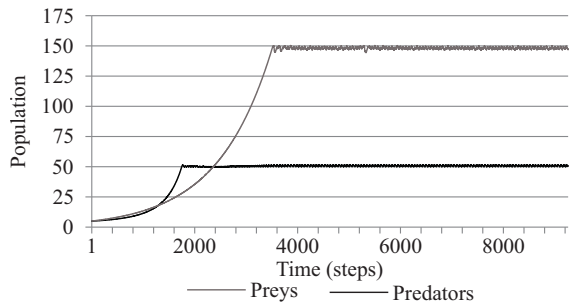

Fig. 3. Evolution of populations through time

V. APPLICATION ON A VIDEO-GAME

In order to demonstrate that this approach also allows to dynamically control actual interactive systems like video games, which usually nowadays often consist in over a hundred parameters to tune, we have applied it on a tower-defense game named ASD - Tower Defense (www.asd-td.com). This typical tower defense game consists of a set of two-dimensional maps, that computer-controlled entities seek to cross. The goal of the player is to place autonomous defenses to prevent entities from crossing the map. Many parameters are involved in this type of game (see table II). There are three types of entities (A, B and C), each having a speed and robustness and each can be released at different paces. Three types of defenses exist (D, E, F) and each has a frequency, a range and a strength. No universal parameter combination can be found to fit the needs and abilities of all players, since humans can exhibit different strategies and behaviors, as well as different learning paces.

TABLE II.

			Score
Entity Type	A	Speed	-
		Robustness	-
	B	Speed	-
		Robustness	-
	C	Speed	-
		Robustness	-
Defense Type	D	Range	+
		Effectiveness	+
		Frequency	+
	E	Range	+
		Effectiveness	+
		Frequency	+
	F	Range	+
		Effectiveness	+
		Frequency	+
Wave Characteristics		Quantity of A	-
		Quantity of B	-
		Quantity of C	-
		Spawning Frequency	-

Domain-related measures are identified as objectives to be reached by any player. For the sake of clarity, only one objective is used in the experiment presented: the percentage of computer-controlled entities that a player has successfully prevented from crossing the map; in other words, the score.

The objective arbitrarily set on the score expresses that each wave should end with an average ratio of 0.75 entities destroyed by the player defenses. Between each wave, the player increases the amount of defenses on the map. The control system is then in charge of restoring the satisfaction of the objective by adjusting the relevant parameters.

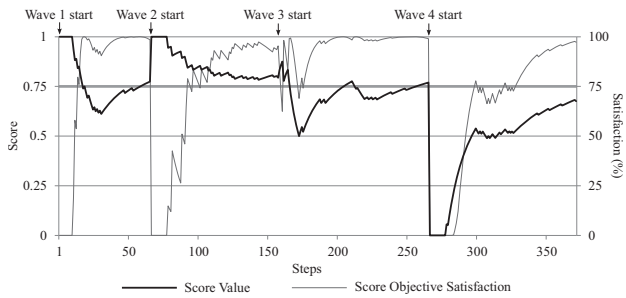

Fig. 4. Evolution of the score during the game session.

Figure 4 shows the evolution of the score in real time, as the game is played during four waves. Each update of the score corresponds to one step within the control system, and to one modification of relevant parameters.

At the beginning of the game, the player places a set of defenses on the map. When the first wave starts, the game is too easy, as shown on the chart: the score is 1, and therefore, the satisfaction is 0. The parameters are adjusted by the MAS, and consequently, the score starts to decrease, to the point, around step 25, where it is below the objective. The tuning of the parameters continues, and the score finally reaches its objective by the end of the first wave.

Before the beginning of the second wave, the player adds more defenses on the map. Consequently, the score at the beginning of the second wave is too high. Parameters are

modified by the MAS, and the objective is satisfied by the end of the second wave. A similar dynamics is observed in the third wave.

On the other hand, before the beginning of the third wave, a large quantity of defenses is removed. Consequently, a reversed dynamics is observed: the score starts with a low value, and parameters are adjusted to make the game easier this time. The score increases during the most part of the wave, to reach 0.7 by the end.

VI. Conclusion

The two experiments presented before show that the proposed approach is able to tune parameters of dynamical systems in order to reach numerical objectives. The objectives reach various satisfaction levels depending on their nature. Deterministic systems, such as the prey-predator model (see section IV), can reach high levels of satisfaction and stability. By contrast, highly stochastic systems, specially because of the involvement of a human being, present a lot a variations over time. However, the real-time aspect of the control system allows quick compensations of sudden changes on the game engine measurements, as shown in section V when the player strategy changes between the waves.

Few information is needed to apply the approach on a given system. Correlations between inputs and outputs of the target systems should be listed by domain experts, but this does not imply that an exhaustive understanding of the system dynamics is necessary. Listed correlations only describe obvious and direct consequences of parameter modifications on certain outputs. They are not supposed to model potential side effects of parameters' modification, as these should be handled by the agent interactions. For instance in the prey-predator example, an increase of the birth rate of predators may lead to a diminishing of the prey population (and eventually to the extinction of both populations), but this is not mentioned in the correlation list, as it is an indirect correlation. The control system is still able to deal with these dynamics, and to properly adjust parameters to maintain the satisfaction of the population objectives.

The described concepts are general enough to be applied on a wide variety of systems, as they only rely on the identification of numerical objectives and parameters. The use of satisfaction functions of objectives and constraints allows designers and domain experts to express their requirements with ease since no technical knowledge is required.

Depending on the nature of the system one is trying to control, there may be non-monotonic correlations. The system described in this paper is based on the identification of monotonic correlations. Though we argue that in the field of games, when considering only direct correlations, most of them are identified as monotonic, there is still room for improvement: a current work focuses on means to autonomously determine the correlations between inputs and outputs. Provided that these correlations could be identified in a reasonably short time, there would be no need for a domain expert to list them. Moreover, learned correlations could be used to tune the parameters regardless of their monotonic aspect.

References

[1] J. Schell, *The Art of Game Design – A book of Lenses*. Elsevier/Morgan Kaufmann, 2008, ch. Game Mechanics Must be in Balance.

[2] M. Csikszentmihalyi, *Flow: The Psychology of Optimal Experience*. NY: Harper and Row, 1990.

[3] L. Vygotsky, "Zone of Proximal Development," *Mind in Society: The Development of Higher Psychological Processes*, pp. 52–91, 1987.

[4] J. Chen, "Flow in Games (and everything else)," *Communcations of the ACM*, vol. 50, no. 4, pp. 31–34, April 2007.

[5] R. Hunicke and V. Chapman, "AI for Dynamic Difficulty Adjustment in Games," in *Proc. of the Challenges in Game AI Workshop*, 2004.

[6] K. J. Astrom and T. Hagglund, *PID Controllers : Theory, Design, and Tuning*. Instrument Society of America, 1995.

[7] R. Moore, H. Rosenhof, and G. Stanley, "Process Control using the G2 Real-Time Expert System," in *Record of the IEEE Industry Applications Society Annual Meeting*, 1989, pp. 1452–1456 vol.2.

[8] C.-C. Lee, "Fuzzy Logic in Control Systems: Fuzzy Logic Controller – Part I," *IEEE Trans. on Systems, Man and Cybernetics*, vol. 20, no. 2, pp. 404–418, 1990.

[9] R. Koster, *A Theory of Fun for Game Design*. Paraglyph Press, 2005.

[10] M. Hagan and H. Demuth, "Neural Networks for Control," in *Proc. of the American Control Conference*, vol. 3, 1999, pp. 1642–1656 vol.3.

[11] M. Nrgaard, O. E. Ravn, N. K. Poulsen, and L. K. Hansen, *Neural Networks for Modelling and Control of Dynamic Systems: A Practitioner's Handbook*. Springer-Verlag NY, 2000.

[12] T. Ross, *Fuzzy Logic with Engineering Applications*. John Wiley & Sons, 2004.

[13] M. Wooldridge, *An Introduction to Multi-Agent Systems*. Wiley, 2002.

[14] D. Capera, J.-P. Georgé, M.-P. Gleizes, and P. Glize, "The AMAS Theory for Complex Problem Solving Based on Self-organizing Cooperative Agents," in *12th IEEE Int. Workshops on Enabling Technologies, Infrastructure for Collaborative Enterprises*, 2003, pp. 383–388.

[15] G. Di Marzo Serugendo, M.-P. Gleizes, and A. Karageorgos, Eds., *Self-organising Software*. Springer, 2011.

[16] S. Lemouzy, V. Camps, and P. Glize, "Principles and Properties of a MAS Learning Algorithm: a Comparison with Standard Learning Algorithms Applied to Implicit Feedback Assessment," in *Int. Conf. on Intelligent Agent Technology (IAT)*. CPS, 2011, pp. 228–235.