A Simple Language for Describing Autonomous Agent Behaviors

Philippe Codognet
Japanese-French Laboratory for Informatics (JFLI)
CNRS /UPMC / University of Tokyo,
Tokyo, Japan
E-mail: codognet@jfli.itc.u-tokyo.ac.jp

***Abstract*— we present a simple and declarative language for describing autonomous behaviors in the paradigm of multi-agent systems. This framework is based on the notion of "goal constraints" and the satisfaction of these goals is done by iterative improvement, with the help of "repair functions" which will partially fulfill them, step by step. We can thus define both deterministic and non-deterministic behaviors. This framework is aimed at describing autonomous systems where the context is changing or unknown, thus goals can be re-evaluated or solved in different ways when changes occur. We will illustrate it with a simple example of life-like navigation behaviors for agents in unknown environments.**

***Keywords*— multi-agent systems, emergence, behavior languages, constraints, optimization.**

I. INTRODUCTION

For more than a decade, multi-agent systems have become popular and widely used for various types of simulation applications [39]. Agents with autonomous behavior are now popular for simulating crowds in urban settings or emergency situations [13]. Many programming languages and frameworks have been proposed [4] from very high-level cognitive agents to low-level reactive agents. In the domain of computer graphics, where agents are used to represent autonomous characters populating a virtual world and interacting with the user, several high-level formalisms have been proposed [18,19], and other agent models have been proposed in a game-theoretic [33]. But there also exist also some lower-level frameworks such as the ABL language [20] which makes it possible to assign goals and subgoals to an agent in a procedural way and to define in such a way agent behaviors. However for autonomous agents evolving in an unknown or changing context, more abstract and declarative formulations are sometimes needed, and we will thus propose in this paper a framework for describing autonomous agent behaviors based on a declarative programming paradigm. We propose to use the formalism of Constraint Satisfaction Problems (CSP) as a general behavior description language. Constraints are used to state goals, or more exactly partial goals, that the agent has to achieve. However, the agent does not know in a deterministic way what action to apply in order to achieve a goal but maybe just have a (non-deterministic) procedure to partially fulfill it; procedure that could be applied repeatedly until the complete satisfaction of his goal or set of goals. Our approach is thus based on the idea of iterative improvement with a heuristic function, as popularized in the domain of search and optimization by the family of local search methods. It is worth noticing that the idea of using simple heuristics to guide the behavior of humans or animals has been recently proposed in many cases by both psychologists and biologists [12].

A key feature of our framework is that behaviors will not be described in a *procedural* way (i.e. stating explicitly the sequence of action needed to achieve the goal), but in a *declarative* way (i.e. stating conditions, and alternatives), by a heuristic function stating how much a goal is achieved. Then alternative configurations will be generated and the best one (i.e. the one achieving the best partial satisfaction of the goal) will be chosen for the next action, yielding thus an iterative improvement process. We believe that a declarative, nondeterministic formalism such as that of goal constraint is more powerful and easier to use than a procedural one.

As an application example, we will consider the problem of autonomous navigation of virtual creatures in a virtual world and show how algorithms derived from biologically-inspired models of navigation can be defined in our approach in order to produce life-like and robust behaviors. In the recent years, there have been a growing interest in both the animal behavior communities in considering non-trivial navigation problems, where animals or virtual agents does not know in advance the location of the goal but rather have a to explore the environment towards it, guided by a stimulus (e.g. light or smell) towards the goal (e.g. food), or just has to search some given area in an efficient way to locate an unknown goal (e.g. a prey). Our approach for defining a description language for autonomous Agent behaviors is indeed rooted in the observation that autonomous navigation can cast it as an optimization problem and we will propose a framework derived from local search techniques to efficiently obtain optimal or near-optimal trajectories. More generally, our framework can be used as a motivation architecture for virtual creatures, by considering variables for denoting internal states (e.g. energy, thirst, etc) and goal constraints for defining internal needs (e.g. the energy should stay above a certain level), routine behaviors (if the energy falls below some level, go for food), or external desired properties (e.g. stay away from predators).

The rest of the paper is organized as follows. Section 2 describes the agent model and Section 3 presents the basic ideas of constraint-based local search. Then, Section 4 defines concretely the declarative language for describing agent behaviors and Section 5 introduces the application example for autonomous navigation of agents. A short conclusion and perspectives end the paper.

II. Agent Model

Simple but interesting life-like behaviors with emergent properties, such as for instance those described in [5], should be easily implemented and tested. We will thus consider a simple model of reactive agents, who can sense their environment through a set of sensors perceiving external stimuli and who can perform actions on their environment through a set of effectors. In between some decision process will decide what behavior to perform, based on some internal state in which the agent is currently.

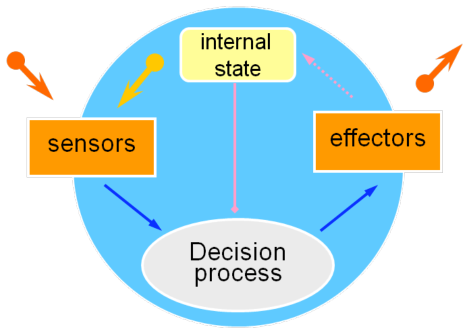

Figure 1. Simple agent model

We will however consider s slightly refined framework and detail the decision process. The agent behavior is defined by a set of *goals*, which will be specific logical relations over both the *internal variables* of the agent (defining the internal state) and the *external variables* (i.e. values of some stimulus perceived through the sensors), and thus decisions are made by a *reasoning engine*, which will try to satisfy the goals. This reasoning engine will be an iterative improvement algorithm that will be detailed in the following. Each goal will propose a repair mechanism in order to reduce the error between the current situation of the agent and a situation satisfying the goal. As an agent can have several goals to satisfy simultaneously, all the repair mechanism have to be aggregated, this will define the next action performed by the agent. This iterative step-by-step mechanism continues until the agent has satisfied all its goals. Obviously, this model can be considered as a specific instance of the Belief-Desire-Intention (BDI) agent model [25], but it is more specialized and operational.

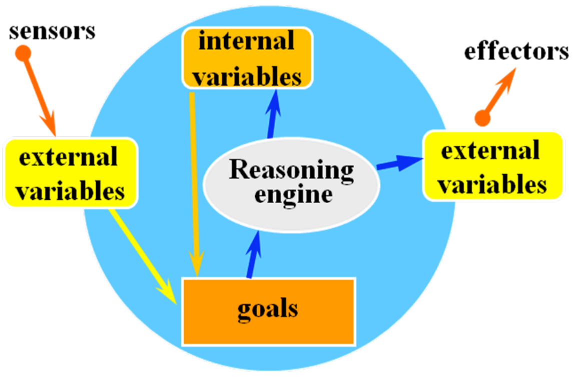

Figure 2. Refined agent model

In computer graphics and animation systems, the most common formalism for representing behaviors of high-level agents, such as virtual humans is some extension of finite state automaton (FSA) [22,40] or more complex hierarchical models [16,32]. For low-level agents, such as the swarm agents in flocks or herds and reactive agents, two basic approaches are classically used:

1. Steering behaviors, where the different low-level goals (such as grouping or escaping) are stated as forces that are then added to produce the actual behavior of the agent in a time-step manner. This approach has been pioneered by Reynolds in the late 80's [30,31], but is still active now and various extensions have been proposed [23,32,9,11, 35].
2. Particle systems [36] or potential fields [14] treating the swarm as a complex physical system.

The second approach might be interesting for efficiently simulating million of agents, but it is not flexible at all and cannot be the basis of a general behavior description framework. Indeed, elegant mathematical models of swarming behaviors have been proposed [7] but they rely on idealistic assumptions and cannot be extended to more complex models that the entomologist have nevertheless pointed out from real observations.

We are obviously closer to the first approach above, but we propose to use the formalism of *constraints* as a general behavior description language. Constraints are used to state goals, or more exactly partial goals, that the agent has to achieve. This can be seen as an extension of the steering behavior approach where constraints are solved logically instead of forces added numerically. One interesting point however is that the constraint formalism is naturally nondeterministic, as opposed to any force-based formalism such as steering behaviors, which is intrinsically deterministic. Indeed we find here again the classical dichotomy between declarative and procedural languages.

III. Goal Constraints and local search

In recent years, the interest for the family of Local Search methods and Metaheuristics for solving large combinatorial problems has been increasing and has attracted much attention from both the Operations Research and the Artificial Intelligence communities for solving real-life problems [1,10]. Local Search (i.e. non-complete) methods have been widely used in Combinatorial Optimization for finding optimal or near-optimal solutions for more than a decade [arts, gonzales], and efficient general-purpose systems for local search now exist, and Simulated Annealing, Genetic Algorithms, Tabu Search, neighboring search, Swarm Optimization, Ant-Colony optimization, etc, are all different kinds of metaheuristics that can be applied to different sets of problems ranging from resource allocation, scheduling, packing, layout design, frequency allocation, etc, in order to produce task-specific local search algorithms. These methods usually start from one random configuration (or a set of random configurations) and try to improve this configuration, little by little, through small changes in the values of the variables. Hence the term "local search" as, at each time step, only new configurations that are "neighbors" of the current configuration are explored. The definition of what constitutes a neighborhood will of course be problem-dependent, but basically it consists in changing the value of a few variables only (usually one or two, even for a problem with hundreds or thousands of variables). The advantage of Local Search methods is that they will usually quickly converge towards a solution (if the optimality criterion and the notion of neighborhood are defined correctly...) and not exhaustively explore the entire search space.

We will use the notion of *constraints* to represent at a declarative level goals that agent have to achieve. Constraints are logical relations, specific relation from a limited vocabulary, for instance artihmetic relations (e.g. =, ≤, etc) or specific symbolic relations (e.g. *all_different)* for which there exist some efficient solving algorithms. Thus achieving (logically) the goals is then reduced to solving (numerically) the constraints. We have developed in previous work [6] a framework for autonomous navigation of agents in virtual worlds based on a constraint-based combinatorial optimization algorithm named "adaptive search" and applied it to path-finding. The core ideas are:

1. To consider for each constraint a heuristic function that is able to compute an approximated degree of satisfaction of the goals (the current "error" on the constraint);
2. To aggregate constraints on each variable and project error on variables thus trying to repair first the variable with worst "error"; and
3. To update and refine these heuristic functions as the agent explores the environment and discovers new information.
4. To use some sort of "Tabu list" in order to give the agent a short-term memory and avoid having it trapped in loops and local minima.

This constraint-based local search method gave us the inspiration for the behavior description language for autonomous agents, as we would like now to rephrase this approach in a more abstract manner and generalize it to any type of behaviors, as a generic goal-based motivation architecture for autonomous agents.

IV. A Behavior Description Language

We will consider that each variable v_i in the model is either an internal variable of some agent or an external variable of the environment (also called a *stimulus*) and is ranging over a discrete or real-valued domains D_i. Each variable has thus a possibly distinct domain. A *configuration* over the domains $D_1 ,..., D_n$ is a vector $(d_1,...,d_n)$ of values with $d_i \in D_i$. When the domains are clear from the context, we will simply speak of a confiruation of size *n*.

An ***agent*** is defined by:

- A set of internal variables $V=\{ v_1, ..., v_k\}$ (describing the internal state of the agent) with associated domains
- A set of external variables $V'=\{ v'_1, ..., v'_j\}$ (describing the *stimuli* perceived by the agent) with associated domains
- A set of behavior goals: $G=\{ g_1, ..., g_p\}$ (defining the intended behaviors of the agent)
- An aggregation operator $\sum$ (used to combine behaviors and decide which action to perform)

The aggregation operator will be used to "sum up" the repair actions proposed to partially solve each of the *p* behavior goals of the agent (which could at some point be contradictory) and thereof to produce a single action to be performed by the agent. This operator is defined when programming the agent, and takes a set of *p* configurations to produce a single result configuration. It could be for instance a (component-wise-) sum, average, max, or any other type of function. In order to give more importance to some goals w.r.t. others, aggregation operators can be weighted sums or hierarchical (e.g. lexicographic ordering choosing to satisfy some first-ranking goal first and then the second-ranking goal only when the first one is satisfied, and so on so forth). Thus complex schemes, including for instance sequential satisfaction of goals or opportunistic behaviors, can be encoded with specific aggregation operators. As each goal g_i of the agent might involve a different set V_i of variables with $V_i \subset V$, the tuple resulting from the repair action of goal g_i has to be extended to a complete configuration of size *k*. This is done by completing the result vector with corresponding current values of the internal variables of the agent. Then aggregation operator ∑ is applied over configuration of size *k* only.

A ***behavior goal***, or simply a ***behavior***, is defined by:

- A set of *variables* $X \subset (V \cup V')$
- A *goal constraint c* over X
  (logical relation stating when the goal is achieved)
- A real-valued *error function* f
  (giving an heuristic value on how much the goal is unsatisfied)
- A *repair mechanism* r over X
  (to be applied when the goal is not satisfied)

Consider an *n*-ary behavior goal $g(X_1, \ldots, X_n)$ and associated variable domains $D_1, \ldots, D_n$. An error function f_g associated with the behavior goal *g* is a real-valued function from $D_1 \times \ldots \times D_n$ such that $f_g(X_1, \ldots, X_n)$ has value zero if $c_g(X_1, \ldots, X_n)$ is satisfied. Observe that it could be possible for f_g to have value zero when c_g is not satisfied, as f_g is only an approximation heuristic function representing the degree of non-satisfaction of the goal constraint. This function is intended to give an indication on how much the constraint is violated. For instance in path-planning applications and spatial goal constraints, the error function can be seen as (an approximation of) the distance between the current configuration (i.e. position) and the closest satisfiable region of the constraint domain, e.g. the air-distance. Since the error is only used to heuristically guide the agent, we can use any simple approximation when the exact distance is difficult (or even impossible) to compute.

A ***repair mechanism*** is defined by:

- A set of variables $X = \{ X_1, \ldots, X_n \}$
- A *repair generator RG*
  (generating a set of alternative configurations for variables in X)
- An *escape generator EG*
  (generating stochastic values for variables in X)

The *repair generator* will generate alternative values only for the internal variables of the agent, as external variables are considered as constants by the agent, in the hope that some of them will better satisfy the goal, that is, will produce a configuration with error less than that of the current one w.r.t. error function of the goal. Then the best one (i.e. with smallest error) will be selected as next possible action for the agent. However, it might happen no alternatives generated by the repair mechanism are improving the error with respect to the goal constraint. Then the *escape generator* will be used in order to propose a stochastic action. In that case the escape generator is used to produce a random value for the internal variables, which will be used for the next action. The range in which this stochastic choice should be applied is of course dependent on the behavior, this is why the escape generator is part of the repair mechanism of each given behavior. Observe that a simple and conservative (although a bit stupid…) escape strategy is to have the identity as escape generator, i.e. to keep the current values of the variables. Therefore, the agent will freeze and do nothing, hoping that the environment will change and allow for some action at a later time. On the other hand a more complex escape generator (e.g., a stochastic procedure based on Perlin noise or Levy flight) will provide a non-deterministic behavior.

As a very simple example, the behavior of an agent which should go to a given position (already known) can be modeled by a goal constraint on a single internal value, its position:

$$agent.position = target.position$$

The error function could be the air-distance:

$$|agent.position - target.position|$$

The repair mechanism can generate several alternative positions in front of the agent for the next step, as there might be some obstacles and the direct way (straight line) might be infeasible. In any case the agent will select the best remaining position (closest to the target).

Observe that if all the repair generators are deterministic (i.e. generate a single alternative configuration), then the overall behavior of the agent will be deterministic. For example the framework of Steering behaviors [30,31], is based on the notion of steering forces that are added to produce the overall behavior of the agent. In the case of so-called *boids* with flocking behaviors steering forces are three distinct separation, alignment and cohesion forces that are computed and then added component-wise, thus deterministically. Indeed, comparing to the steering behavior paradigm, we propose here non-deterministic behaviors and we do not try to define a combination of forces that will bring the agent to the desired position but just check possible alternative configurations and choose the best one, which can be done efficiently in the local search approach. Also observe that we can cope with dynamically changing priorities between behaviors (e.g. in the definition of the aggregation operator of the agent), because the error functions are re-evaluated and combined at each time step.

The overall generic computation model for the agents can thus be simply defined as follows:

```
For each (agent A in the world) {
  For each (goal G of agent A){
    Evaluate current error $F_G$
    Evaluate satisfaction condition $C_G$
    If (not $C_G$) {
      Generate a set of alternatives R
      For each (alternative S in R){
        Evaluate error function $F_G$ on S}
      Select best alternative config. S'
            (i.e. with smallest error)
      If (error of S' > $F_G$) {
        S' = escape generator for G}
    repair action $R_G$ = S' extended to $V_A$
    }
  }
  Aggregate all repair actions with ∑
  Perform resulting action for agent A
```

}

It is clear that the combination of several goals might produce quite complex behaviors. For instance if the agent should go towards some object (goal constraint: *agent.position = target.position*), and stay at some distance of it (goal constraint: *agent.position ≥ target.position + d*), then a following behavior (at distance *d*) will be simply obtained if the target is moving.

V. EXAMPLE: NAVIGATION STRATEGIES

For the last two decades, the observation and modeling of animal motion and navigation strategies by zoologists and entomologists have inspired researchers in Artificial Intelligence and Artificial Life for the design of simulation models for crowds and autonomous agents. In particular, Nature-inspired simulations based on the metaphor of *swarm intelligence*, such as the foraging simulation of ant-colonies [3] or collective motion such as flocks schools and herds [30,8]. More recently, [37] presents a comprehensive review the similarities between collective behaviors of different types of agents: bacterial colonies, cells, insects, fishes, birds, mammals, humans, etc. Besides these collective motion and cognition, researchers have also been investigating movements and navigation strategies of single animals, especially the navigation patterns of predators looking for a prey in an unknown environment. It appears that similar patterns and probably search strategies are put at use by very different animal (e.g. bees, flies, moths, peacocks, albatross, and can be modeled by so-called "Levy flights" [29]. This method has been defined as an optimal search behavior (e.g. better than random walk or so-called spiral search) for random search in an unknown environment, especially when the targets are sparse [38] [2]. Even more interestingly [27] showed that Levy flight navigation strategies could be an emergent property resulting from a simple gradient-following strategy for chemotaxis (reaction to some chemical stimulus). This is indeed the behavior of simple living creatures such as the small soil nematode *Caenorhabditis elegans* [21,24]. Very recently [15,28] showed that a simple gradient-following strategy such as chemotaxis can be used to navigate in complex mazes and showed that such simple chemotaxis finds in several examples the shortest route, although this method does not guarantee to always find an optimal path.

When considering the domain of adaptive behaviors [34], an exploration process guided by a stimulus (e.g. light or smell) towards a goal (e.g. food) of unknown location, two different methods are usually defined: temporal difference or spatial difference. The temporal difference method consists in considering a single sensor (e.g. the nose) and checking at every time-point the intensity of the stimulus. If the stimulus is increasing, then the agent continues in the same direction, otherwise the direction is changed randomly and so on so forth. This is exemplified by the chemotaxis of the *Caenorabditis elegans*. With this method, the agent eventually reaches the goal, but might wander in some irrelevant regions of the environment in between [21]. A more efficient strategy is possible by using the spatial differences method [17]. It requires to have (at least) two identical sensing organs, placed at slightly different positions on the agent (e.g. the two ears). The basic idea is to favor, at any time-point, motion in the direction of the sensor which receives the most important stimulus. If none of the sensors perceives an increasing of the stimulus, then a random move is performed. This behavior gives very good results and the agents goes most of the time directly towards the goal. Also when the goal is moved away, e.g. because the prey is moving, the agent reacts instantly (as the stimulus is checked iteratively at every time-point) and moves towards the new location.

Both of these models fit quite well within the general framework for describing agent behaviors presented in the previous section. The temporal difference method consists in having a single internal variable for storing the value of the stimulus at the previous time point and if the current value of the stimulus is not increasing then a random turn (within some given limits, e.g. plus or minus 30 degrees) will be performed. The spatial difference method consists in having two internal variables for checking the values of the stimulus with two sensors in different positions and then making a move in the direction of the sensor with highest stimulus. Observe that one can have more than two sensors and thus check the intensity of the stimulus at various alternative points, but it seems than it does not improve the search process significantly [17], explaining thus maybe why animals and humans only need two symmetric sensors (e.g. eyes or ears).

VI. CONCLUSION

We presented a simple and declarative framework for describing autonomous behaviors in the paradigm of multi-agent systems. This model is a generalization of previous work on navigation models for autonomous agents in virtual worlds [6], and can be used as generic goal-based motivation architecture for autonomous agents. Our language is based on the notion of "goal constraints" and the satisfaction of these goals is done by iterative improvement, with the help of "repair functions" which will partially fulfill them, step by step. We are currently finishing an implementation of this framework using the *processing* language [26].

REFERENCES

[1] E. Aarts and J. K. Lenstra, Eds., *Local Search in Combinatorial Optimization,* Chichester, UK: John Wiley and Sons, 1997.

[2] F. Bartumeus and J. Catalan, "Optimal search behavior and classic foraging theory", *Journal of Physics A*, vol. 42 (2009), 434002.

[3] E. Bonabeau, M. Dorigo and G. Theraulaz, *Swarm Intelligence: From Natural to Artificial Systems,* Oxford University Press 1999.

[4] R. H. Bordini, M. Dastani, Mehdi; A. El Fallah Seghrouchni (Eds.), *Multi-Agent Programming: Languages, Platforms and Applications*, Springer Verlag 2005.

[5] V. Braitenberg, *Vehicles - Experiments in Synthetic Psychology*, MIT Press, 1984.

[6] P. Codognet, "Animating virtual creatures by constraint-based adaptive search" In: *Proc. VSMM2000, 4th Int. Conf. on Virtual Systems and Multimedia*, Gifu, Japan, IOS Press 2000.

[7] F. Cucker and S. Smale, "Emergent Behavior in Flocks", *IEEE Transactions on Automatic Control*, vol. 52, Issue 5 (2007), pp 852 – 862.

[8] I. Couzin, J. Krause, R. James, G. D. Ruxton, and N.R. Franks, "Collective Memory and Spatial Sorting in Animal Groups", *Journal of Theoretical Biology*, vol. 218, no. 1, Pages 1-11.

[9] I. D. Couzin, "Collective cognition in animal groups", *Trends in Cognitive Sciences*, vol. 13 no. 1, 2008, pp 36-43.

[10] T. Gonzalez (Ed.), *Handbook of Approximation Algorithms and Metaheuristics*, Chapman and Hall / CRC, 2007.

[11] C. Hartman and B. Benes, "Autonomous Boids", *Computer Animation and Virtual Worlds*, vol. 17 no. 3-4 (2006), pp 199–206.

[12] Hutchinson, J.M.C., Gigerenzer, G., 2005. "Simple heuristics and rules of thumb, where psychologists and behavioural biologists might meet", *Behavioral Processes*, vol. 69, no. 2, 2005, pp 97–124.

[13] T. Ishida, L. Glasser and H. Nakashima, *Massively Multi-Agent Systems*, LNAI 3446, Springer Verlag 2004.

[14] X. Jin, C. C. L. Wang, S. Huang and J. Xu, "Interactive control of real-time crowd navigation in virtual environment", In: *Proc. 2007 ACM symposium on Virtual Reality Software and Technology*, pp 109-112, ACM Press 2007.

[15] I. Lagzi, S. Soh, P. J. Wesson, K. P. Browne and B. A. Grzybowski, "Maze Solving by chemotactic droplets", *Journal of the American Chemical Society*, vol. 132 no. 4, 2010, pp 1198–1199.

[16] F. Lamarche and S. Donikian, "Crowd of Virtual Humans: a New Approach for Real Time Navigation in Complex and Structured Environments", *Computer Graphics Forum* vol. 23 no. 3, 2004, pp. 509-518.

[17] W. Leow. "Computational Studies of Exploration by Smell", in [34].

[18] P-S. Liew, C-L. Chin and Z. Huang, "Development of a computational cognitive architecture for intelligent virtual character", *Computer Animation and Virtual Worlds*, vol. 20 no. 2-3 (2009), pp 257-266.

[19] L. Luo, S. Zhou, W. Cai, et al., "Agent-based human behavior modeling for crowd simulation", *Computer Animation and Virtual Worlds*, vol. 19 no. 3-4 (2008), pp 271-281.

[20] Michael Mateas and Andrew Stern, "A Behavior Language: Joint Action and Behavioral Idioms", In: *Life-like Characters: Tools, Affective Functions and Applications*, H. Prendinger and M. Ishizuka (Eds.), Springer Verlag 2004.

[21] T. Morse, T. Ferrée, S. Lockery, "Robust Spatial Navigation in a Robot Inspired by Chemotaxis in C. Elegans", in [34].

[22] T. Noma, L. Zhao and N. I. Badler, "Design of a Virtual Human Presenter", *IEEE Computer Graphics and Applications*, vol. 20, no. 4, 2000, pp. 79-85.

[23] C. Pedica and H. Vilhjalmsson , Social Perception and Steering for Online Avatars, *In: Proc. IVA'08, 8th international conference on Intelligent Virtual Agents*, Tokyo, Japan, LNAI 5208, Springer Verlag 2008, pp 104 – 116.

[24] J. T. Pierce-Shimomura, T. M. Morse, and S. R. Lockery, "The fundamental role of pirouettes in Caenorhabditis elegans chemotaxis", *Journal of Neurosciences*, vol. 19 no. 21, pp 9557–9569.

[25] A. S. Rao and M. P. Georgeff, "Modeling rational Agents with a BDI-Architecture", In : *Proc. 2nd Int. Conf. on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann, 1991, pp 473-484.

[26] C. Reas and B. Fry, *Processing*, MIT Press 2007.

[27] A. M. Reynolds, "Deterministic walks with inverse-square power-law are an emergent property of predators that use chemotaxis to locate randomly distributed prey", *Physical Review E*, vol. 78 (2008), 011906

[28] A. M. Reynolds, "Maze-solving by chemotaxis", *Physical Review E*, vol. 81 (2010), 062901.

[29] A. M. Reynolds and C. J. Rhodes, "The Levy flight paradigm: random search patterns and mechanisms", *Ecology*, vol. 90 no. 4, 2009, pp 877-887.

[30] C. W. Reynolds, "Flocks, Herds, and Schools, A Distributed Behavioral Model", *Computer Graphics,* vol. 21 no. 4 (SIGGRAPH '87 Conference Proceedings), 1987, pp. 25-34.

[31] C. W. Reynolds, "Steering behaviors for autonomous characters", In: *Proc. of 1999 Game Developers Conference*, San Francisco, Miller Freeman Group 1999, pp. 763-782.

[32] W. Shao and D. Terzopoulos, "Autonomous pedestrians", *In: Proc. of the ACM SIGGRAPH'2005, Symposium on Computer Animation*, ACM Publishing 2005.

[33] S. Schiffel and M. Thielscher, "A Multiagent Semantics for the Game Description Language", In: *Agents and Artificial Intelligence,* Computer and Information Science series vol. 67*,* Springer Verlag, 2010, pp 44-55.

[34] N. Schmajuck (Ed.), "Special issue on Biologically-inspired models of Navigation", *Adaptive Behavior*, vol. 6, no. 3/4, Winter/Spring 1998.

[35] M. Schuerman, S. Singh, M. Kapadia and P. Faloutsos, "Situation agents: agent-based externalized steering logic", *Computer Animation and Virtual Worlds*, vol. 21 no. 3-4 (2010), pp 267–276.

[36] A. Treuille, S. Cooper and Z. Popovic, "Continuum crowds", *ACM Transactions on Computer Graphics*, vol. 25 no. 3 (SIGGRAPH 2006 conference proceedings*),* 2006, pp. 1160-1168.

[37] T. Vicsek and A. Zafiris, "Collective Motion", preprint available on *arXiv:1010.5017v1 [cond-mat.stat-mech]*.

[38] G. M. Viswanathan, S. V. Buldyre1, S. Havlin, M. G. E. da Luz, E. P. Rapos and H. E. Stanley, "Optimizing the success of random searches", *Nature*, no. 401 (28 October 1999), pp 911-914.

[39] Michael Wooldridge, *An Introduction to MultiAgent Systems*, John Wiley & Sons Ltd, 2002.

[40] H. Xiao and C. He, "Behavioral Animation Model for Real-Time Crowd Simulation", In: *Proc. of 2009 International Conference on Advanced Computer Control*, IEEE Press 2009.