ICAT '99

Declarative Behaviors for Virtual Creatures

Philippe Codognet

University of Paris 6 and INRIA
LIP6, case 169, 8, rue du Capitaine Scott, 75 015 PARIS, FRANCE
Philippe.Codognet@lip6.fr

Abstract

We present a high-level language for describing behaviors of autonomous agents in 3D virtual worlds. In order to describe agent behaviors, we have designed VRCC, a concurrent constraint programming language integrated in the VRML (Virtual Reality Modeling Language) environment. The basis of this declarative language is the notion of constraint, and it is based on the Timed Concurrent Constraint framework, which integrates a discrete notion of time adequate for animation systems such as VRML. We illustrate this approach by some simple examples of virtual creatures that can autonomously move in the 3D world, and we describe some simple behaviors derived from biologically-inspired models of navigation.

Key words: Virtual Reality, VRML, Autonomous Behaviors, Artificial Life, Constraints.

1. Introduction

Our aim is to design a high-level language for describing behaviors of autonomous agents in 3D virtual worlds. The basis of this declarative language is the notion of constraint, which can be used to enforce hidden relations between agents and/or between an agent and the environment (e.g. minimal distances, non-collision, etc) or general integrity rules (such as gravity) and thus make sure that the simulated virtual world does not depart too much from our real one.

A constraint is simply a logical relation between several unknowns, these unknowns being variables that should take values in some specific domain of interest. A constraint thus restricts the degrees of freedom (possible values) the unknowns can take; it represents some partial information relating the objects of interest. The whole idea of constraint solving is to start reasoning and computing with partial information, ensuring the overall consistency and reducing as much as possible the domains of the unknowns. This is in sharp contrast to classical programming (either imperative, object-oriented or functional) where one only computes with complete values and has no support for propagating partial information. Constraint Programming has proved to be very successful for Problem Solving and Combinatorial Optimization applications, by combining the declarativity of a high-level language with the efficiency of specialized algorithms for constraint solving, borrowing sometimes techniques from Operations Research and Numerical Analysis [14]. We plan to use the power of constraint solving techniques, for developing complex and efficient planning modules for autonomous agents integrated in 3D virtual environments.

VRML (Virtual Reality Modeling Language) has become since a few years a *de facto* standard for publishing 3D scenes on the Web. It is a very interesting model because of its generality and versatility, see for instance [4] or [7] for a good introduction. Many plug-ins for Web browsers now exist for interpreting the VRML file format, and there is moreover an ISO normalisation of this language. However, VRML is more than a mere specification format for 3D scenes because one can specify complete virtual worlds in which the user can wander at will and where it can interact with 3D objects. More importantly, there is currently a growing interest in developing support for shared virtual worlds, which is putting the (virtual) reality of 3D digital communities at hand.

Nevertheless, most of these shared virtual worlds suffer from an important drawback, namely they are not lively or interactive enough. As interaction is usually limited to chat with other users co-located in the same virtual world, the interest of a site depends more on the number of other people « visiting » it than on the originality or theme of its design. We will therefore consider in this paper the problem of “populating” such worlds with virtual agents representing life-like creatures which could autonomously navigate and react to their environment, and also possibly interact with users. As a first step in that direction, we will consider agents with simple reactive behaviors inspired from research in the field of Artificial Life and robotics. For this purpose, we need to design a language in which such behaviors can be stated, and this language should be both simple, declarative and powerful in order to make it possible to express a great variety of operations. We propose a framework based on VRCC, an integration within the VRML model of a timed Concurrent Constraint language. Moreover, in order to design autonomous, life-

like 3D creatures that can autonomously move in the virtual world, we propose some simple behaviors derived from biologically-inspired models of navigation. A key point here is to consider situated agents, reactive to an unknown and constantly changing environment. The first example is motion planning for a virtual robot in a maze. The idea is to go to a point identified as a goal and to avoid obstacles. The agent is reactive and can thus take into account moving obstacles and goal. Secondly, we investigate exploration guided by a stimulus (e.g. smell) towards a source (e.g. food), location of which is unknown, using either temporal difference or spatial difference methods.

2. A Declarative Language

In computer graphics and animation systems, the formalism which is the most commonly used to represent behaviors is the finite state automaton (FSA). Many variants exist, such as the PatNets of [2], the Hierarchical FSA [6] or the parallel FSA [5]. Our approach rather considers that for representing complex life-like behaviors, one should not be restricted to some extended FSA formalism but indeed needs the power of a complete programming language. The ability to handle states, variables, parameters, or recursion is indeed crucial. We have thus investigated the formalism of Concurrent Constraint Programming (CC) which is bot declarative, high-level and yet simple. Concurrent Constraint Programming (CC) has been proposed a few years ago [12] as a new programming paradigm that can be seen as the merging and generalization of Constraint Logic Programming and Concurrent Logic Languages. It makes it possible to combine both approaches, that is, on the one hand, the ability to reason (symbolically) and to compute (numerically) on specific domains (*constraints*) and, on the other hand, the possibility to have a dynamic data-driven control of the execution flow (*concurrency*). The fundamental idea of Concurrent Constraint Languages is the use of constraints for defining the synchronization and control mechanisms. Therefore, several agents could communicate and synchronize through a global store where all information is added in a monotonic way through the time line. However such a framework, if well-suited for problem-solving or concurrent computing is not suited

2.1 Timed Concurrent Constraint language

The Timed CC (TCC) extension of classical CC languages, described in [13] is the framework needed for integration within a 3D animation system such as VRML. It basically introduces a notion of discrete time that is compatible with that of VRML (time sensors). The basic idea of TCC is that of synchronous programming exemplified by languages such as Esterel [3] : programs run and respond to signals intantaneously. Time is decomposed in discrete time points, generated by clocks outside the system and programs are executed *between* time-points. Program execution takes « zero time », that is, is neglectible w.r.t. the time clocking the overall system. Surprisingly as it seems, this scheme, called the *perfect synchrony hypothesis*, is nevertheless adequate for modeling real-time systems and a language such as Esterel is indeed used to program low-level real-time controllers. In TCC, the concurrent computation of running agents is started at each time point and continues until quiescence of the computation. Control is then given back until another signal (time point) is generated**.**

2.2 Syntax

At each time point, concurrent agents are running simultaneously until quiescence and then the program moves to the next time-point. Basic actions performed by the agents are either posting a constraint to a shared store (Tell operation), suspend until some constraint is entailed by the store (Ask operation), perform some method (Call operation), or post an action to be performed at the next time-point (Next operation). The syntax of these operations, in the classical algebraic CC style, is given below. We will use letters *X,Y,…* to denote program variables, letter *A* to denote an action and letter *c* to denote any constraint.

tell(c)	constraint addition
ask(c) => A	synchronization
A , A	concurrent execution
X:A + Y:A	max-indeterminism
p(X,Y,...)	agent creation
∃ X. A	local variables
next (A)	temporal construct
goal(c)	goal constraint

With respect to classical TCC, there are two new operations. *Goal constraints* are relations that the agent should try to achieve in the following time-steps if they are not satisfied at the current time-point. One thus need for each such constraint some *iterative-repair* method that should eventually converge to a satisfiable state. *Max-indeterminism* consists in choosing among two couples (variable,agent) to run the agent whose variable has maximal value (if any, otherwise an agent is randomly chosen). Obviously such a construct extend usual (blind) indeterminism and it is trivially extended to a Σ construct ranging over a finite set of agents.

2.3 Constraint Solving

The use of constraints as goals in order to describe behaviors is a simple and declarative way to specify animation of virtual creatures. Several goal constraints can be stated concurrently (e.g. non-collision with obstacles together with minimal distance w.r.t. a specific

goal area) and solved together by the constraint solver. However this somewhat differs from classical constraint satisfaction techniques, where the constraint solver is in charge of finding an assignment of the variables of the problem in order to achieve a global solution. Indeed the outcome of a behavior could be considered as a sequence of actions that will eventually lead to the satisfaction of the goal, but not within a single time-step. Thus the reactive nature of behaviors naturally leads to consider reactive constraint solving techniques instead of classical (transformational) ones. We propose to use "iterative repair" methods in order iteratively select actions that will eventually lead to the satisfaction of the goal constraints. We have therefore developed a new solving method called “adaptive search”, derived from local search techniques [1] [17].

One has to observe that, with respect to a general planning module, we restrict goals to be within the constraint domain proposed by the host language. It will thus be in general either numeric and symbolic constraints over finite domains, or arithmetic interval constraint over continuous domains, which are the two main domains for which efficient constraint solving techniques exist today [14].

2.4 VRCC

Let us consider how to integrate the TCC framework within the VRML model.

VRML is a very interesting model because of its generality and versatility, which grew out of SGI's format for storing 3D scenes in Open Inventor, and was thus primarily design to describe static 3D objects and scenes. VRML is based on the classical scene graph architecture for representing and grouping the 3D objects. Scene can be constructed from built-in nodes (boxes, spheres, cone, or any polygon-shaped form) and new user-defined nodes can be also constructed using PROTO nodes. VRML provides basic light sources and performs basic scene illumination calculations. VRML version 2.0 [16] introduced primitives for animation and user interaction, based on a simple *event model*: each object in the virtual world can receive input events and send output events, these events are communicated between objects through predefined *routes*, completely independent of the structure of the scene graph. A VRML world basically consists of nodes, describing various geometries, in the scene graph and of a series of routes between objects. An important feature is the possibility to have *script nodes*, with associated Java or JavaScript programs, for treating and modifying events.

VRML is based on a discrete time model, with special nodes called *time sensors* driving the animation.

The main idea for integrating TCC within the VRML environment is to consider that VRML should clock the TCC system, i.e. that some *TimeSensor* node in VRML should send its clock ticks (*events*) for generating time points in the TCC part. This means that TCC computations should run between two events generated by a VRML time sensor node, that is, two frames drawn by the browser. The basic assumption here is that time is external and driven by VRML; there is no control over time in Timed CC. Concurrent computation of running agents is started at each time point (generated by VRML) and until quiescence of the computation. Control is then given back to VRML when quiescence is achieved (that is, no agent can further be reduced), and so on so forth.

The first interesting property of VRCC computations is given by the underlying concurrency of TCC, both between agents and inside a single agent. Indeed, one can consider an agent composed of different sub-parts, with their own methods to perform animation, but which are linked together by some structural constraints.

The second property of the execution model is reactivity: changes in VRML world are taken into account at each time-point and therefore behaviors (TCC computations) can react in real-time, or more precisely at the next time-point. One should therefore take care of avoiding to slowdown the system by heavy computations in the TCC part.

The third property is compositionality : because one has logical variables and constraints for programming behaviors in TCC, constraints can be accumulated by several agents (or sub-agents) to compositionally construct a single behavior. Several constraints for the same agent will be treated together in order to propose a global solution.

Last but not least, the integration of constraint in the behavior language makes it possible to express problem solving capabilities: for instance a planner module, or scheduling of tasks, are easy to implement in a constraint-based language.

3. A Classical Example: N Queens as Intelligent Agents

Let us take a well-known example in order to understand the behavior of intelligent agents in VRCC.

We will consider the perennial *N Queens* problem which consists in placing 8 queens on a 8x8 chessboard in such a manner that no two queens attack each other. Only one kind of constraint will be used to prohibit two queens to be on the same row or diagonal : the disequation. Considering that there should be only one queen on each row, we will note Qi the queen on the ith row, and consider that the value of this variables is the number of the column on which it stands. We shall thus have for each couple of variables (queens) Qi and Qj :

$$Qj \neq Qi. \, , \, Qj \neq Qi + j - i \, , \, Qj \neq Qi - j + i$$

Although very classical in the constraint programming

community, this problem is quite artificial and there exists many methods to efficiently solve it. We use it only to illustrate the idea of hidden relations (the above mentioned disequations) constraining agents to move in order achieve a coherent state and we will focus on one particular point : how to express the problem solving aspects as a multi-agent system where each queen is an autonomous agent moving on the chessboard and cooperating with others to find a solution. This is close in spirit to the work of [18], but will indeed give rise to a quite different model. Another aspect is important as we want with to depict visually, as a 3D world, the solving process : when moving from one partial solution to another we are limited to physically feasible moves, that is, we cannot have queens randomly jumping from place to place, but only continuous moves from one position to another. The following figures depicts the initial and final states of the problem.

Fig 1. Initial state

Fig 2. Final state

4. Biologically-inspired creatures

We have experimented with less trivial examples of autonomous agents, adapting from biologically-inspired models of navigation. There is currently a growing interest for such models both in the Artificial Life and the robotics community, as exemplified for instance by [11] or [15] which provide excellent surveys of recent researches.

4.1 Obstacle Avoidance

Our first example will be motion planning for a virtual robot in a room filled up with various obstacle, see figure 3 for a snapshot of the initial configuration of this problem. The idea is to go to a point identified as a goal (the psychedelic sphere in the bottom right of the screen) and to avoid the brick-textured obstacles.

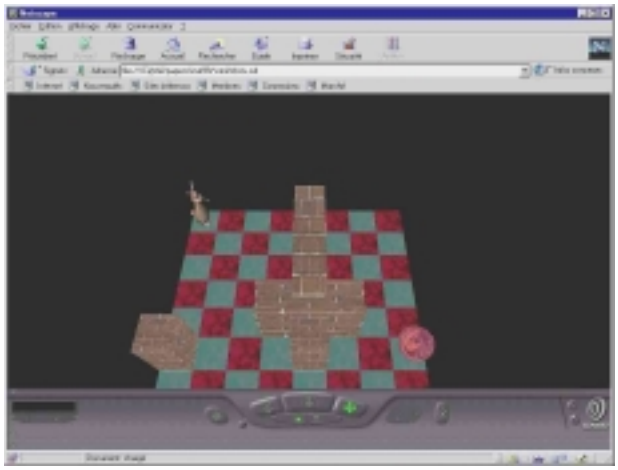


Fig. 3. Obstacle avoidance configuration

Let us remark however, that this example is nevertheless a complex robot navigation problem with moving target goal and with moving obstacles, because both could be interactively dragged by the user in real-time. The agent is thus reactive and has to take any modification of the environment or target goal position into account.

In this example avoidance constraints with respect to obstacles are used, in conjunction with a minimal distance constraint between the current position of the creature and that of the goal.

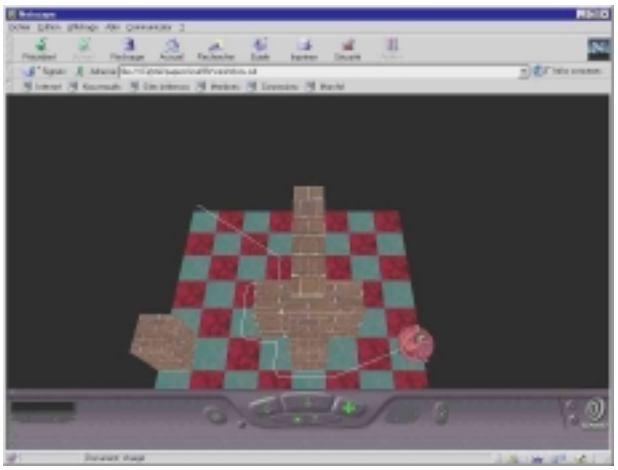


Fig. 4. Trajectory of the agent

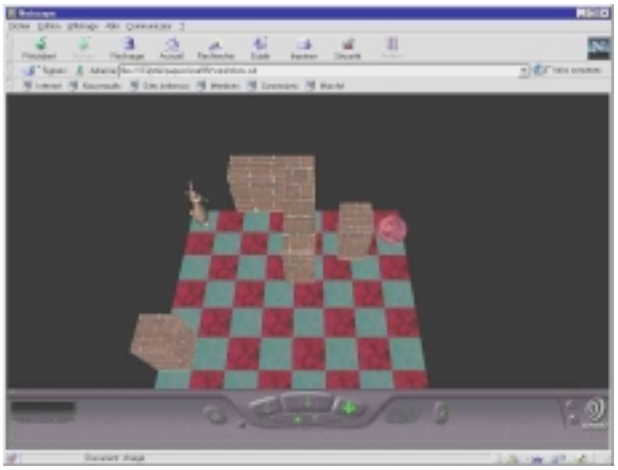


Fig. 5. Another configuration

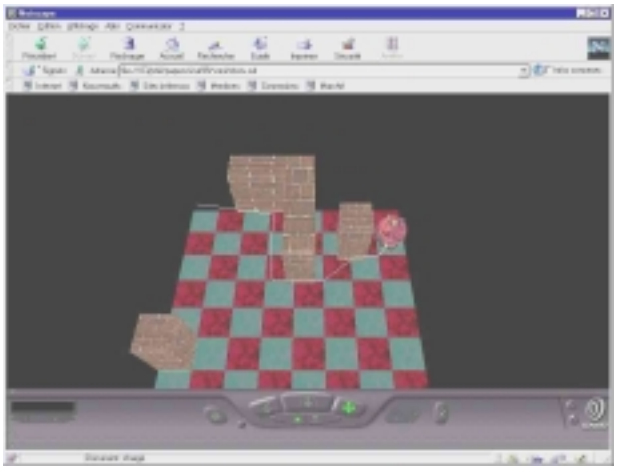


Fig. 6. Another Trajectory

4.2 Stimulus-driven Search

Biologists usually consider that the navigation system of small animals (rats and mice, etc) can be decomposed in two subsystems : one based on maps (called the *locale* system, as it is based on the recognition of locations linked in a graph-like manner), and the other based on routes (called the *taxon* system, for behavioral orientation). There seems to be evidences that the hippocampus is used to store the cognitive map for the *locale* system, cf. [10]. We will here only consider an agent with a very limited intelligence building no cognitive map but using only the *taxon* system for route navigation. The second example that we will consider is indeed an extension of the first one, where we will not give to the agent the location of the goal but rather use a exploration guided by a stimulus (e.g. smell) towards the goal (e.g. food), location of which is unknown. This exploration will be performed by using two different methods : 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 for instance by the chemotaxis (reaction to a chemical stimulus) of the Caenorabditis Elegans, a small soil nemapode [9]. Therefore the only constraint here is non-collision, that is, enforcing some minimal distance with respect to obstacles. However the creature will also have an additional operational (programmed) behavior that will be to check the input stimulus on its sensor at each time-step and to change direction if necessary. Again, as in the previous obstacle avoidance example, the source of the stimulus and the obstacle can be moved interactively in real-time and the agent has to react accordingly.

Figure 7 and 8 depicts the top and side views of the initial setting. Observe that the corridor between the two obstacles is not large enough for the creature to pass through and thus a detour will be needed. The source of the stimulus (“smell”) is the cake on the right-hand side.

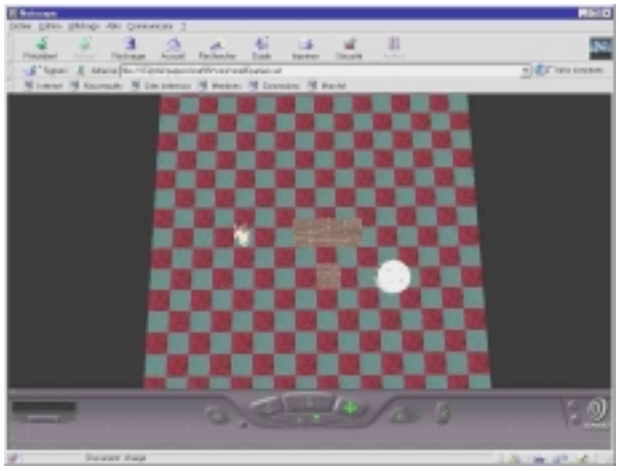


Fig. 7. Top view of initial configuration

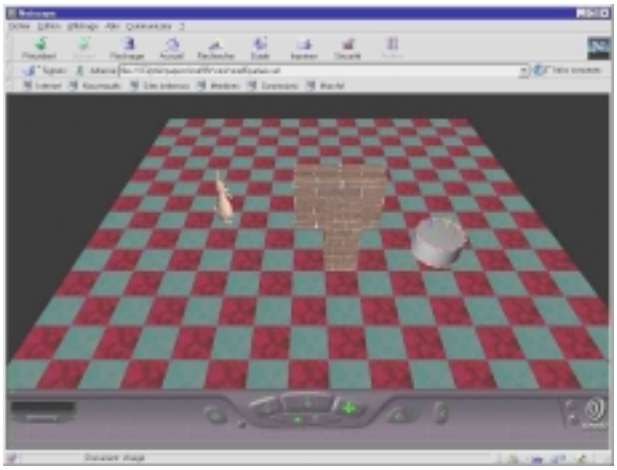


Fig. 8. side view of initial configuration

The resulting trajectories of the creature on two different runs of this experiment are depicted on figure 9 and 10, showing that the agent performs not so bad, i.e. it eventually reach the source of the stimulus, although it does not perform a direct trajectory and might wander in some irrelevant regions of the environment.

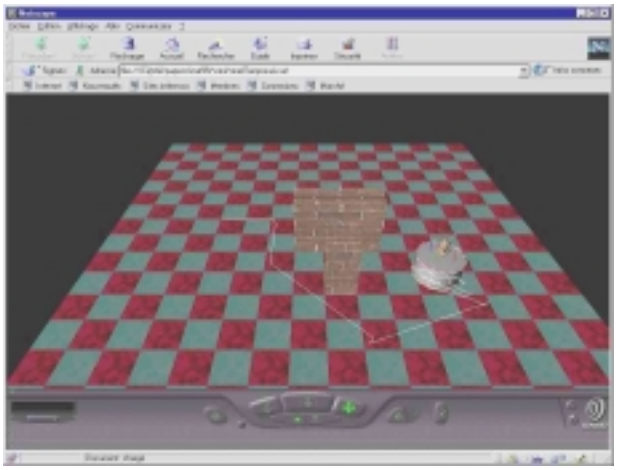


Fig 9. Temporal difference method (1)

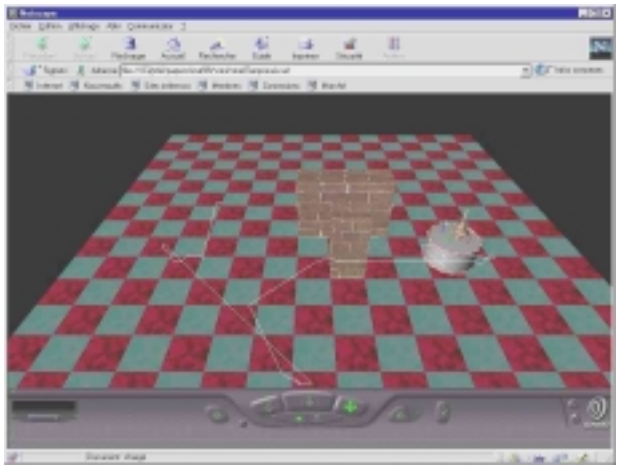
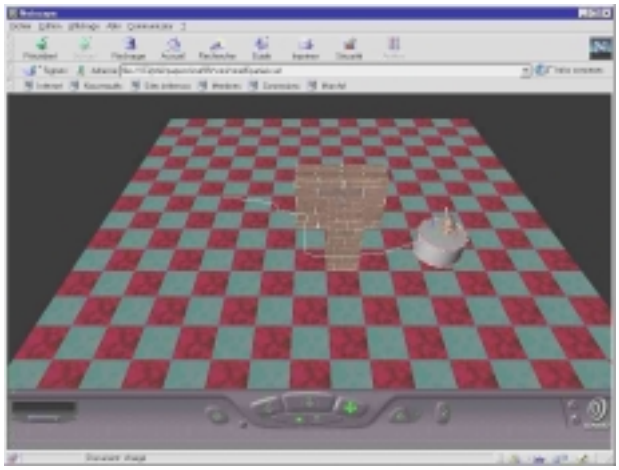


Fig 10. Temporal difference method (2)

A more efficient strategy is possible by using the spatial differences method. It requires to have two identical sensing organs, placed at different slightly positions on the agent (cf. the two ears). The basic idea is to favor motion in the direction of the sensor that receive the most important stimulus, as in the well-known sensory-motor coupling in robotics. This behavior gives very good results, and the agents goes directly towards the source of the stimulus, see the corresponding trajectory depicted on figure 11.



More complex strategies are possible, e.g. by considering more sensors or by combining the temporal difference of several sensors, but they are not really more effective than the basic spatial difference method [8].

6. Conclusion

We have presented a high-level language for describing behaviors of autonomous agents in virtual environments. We use VRML for the 3D visualization part and a timed concurrent constraint programming (TCC) for agent

programming. This framework seems well-suited for animating simple 3D autonomous creatures. Interesting behaviors such as obstacle avoidance or stimulus-driven exploration towards a source (gradient-following) are easily expressible in this system and opens up for a variety of biologically-inspired behaviors.

References

1. E. Aarts and J. Lenstra (Eds). *Local Search in Combinatorial Optimization*, Wiley, 1997.
2. N. Badler. *Real-time virtual humans*, Pacific Graphics 1997.
3. G. Berry and G. Gonthier. The Esterel Programming Language : Design, Semantics and Implementation, *Science of Computer Programming*, vol. 19 no. 2, 1992
4. R. Carey and G. Bell. *The Annoted VRML 2.0 Reference Manual*, Addison–Wesley, 1997.
5. S. Donikian. Multilevel Modeling of Virtual Urban Environments for Behavioural Animation. *Proc. Computer Animation 97*, Genève, Suisse, IEEE Press 1997.
6. Y. Koga, C. Becker, M. Svihura, and D. Zhu. On intelligent Digital Actors. *Proc. Imagina 98*, Monaco, 1998.
7. R. Lea and K. Matsuda. *Java for 3D and VRML Worlds*, New Riders, 1996.
8. W. Leow. Computational Studies of Exploration by Smell, in [11].
9. T. Morse, T. Ferrée and S. Lockery. Robust Spatial Navigation in a Robot Inspired by Chemotaxis in C. Elegans, in [11].
10. J. O'Keefe and L. Nadel. *The hippocampus as a cognitive map*, Clarendon Press 1978.
11. R. Pfeifer and R. A. Brooks (Eds.). Special issue on Practice and Future of Autonomous Agents, *Robotics and autonomous systems*, vol. 20, no. 2-4, June 97.
12. V. Saraswat. *Concurrent Constraint Programming*, MIT Press, 1993.
13. V. Saraswat, R. Jagadeesan and V. Gupta. Timed Default Concurrent Constraint Programming, *Journal of Symbolic Computing* (1996) **22**, pp 475-520.
14. V. Saraswat, P. Van Hentenryck, P. Codognet *et al.* Constraint Programming, *ACM Computing Surveys* vol. 28 no. 4, December 1996.
15. N. Schmajuck (Ed.). Special issue on Biologically-inspired models of Navigation, *Adaptive Behavior*, vol. 6, no. 3/4, Winter/Spring 98.
16. The VRML Architecture group. *The virtual reality modeling language specification, version 2.0*, August 1996. Available at http://vag.vrml.org/VRML2.0/FINAL
17. J. P. Walser. *Integer Optimization by Local Search : A Domain-Independent Approach*, LNAI 1637, Springer Verlag 1999.
18. M. Yokoo and T. Ishida, Search Algorithms for Agents, In: *Multiagent Systems*, G. Weiss (ed.), MIT Press 1999.

.