Computer Simulations of Adaptive Behavior in Animats

Agnès Guillot and Jean-Arcady Meyer
Groupe de BioInformatique, CNRS UA 686
Ecole Normale Supérieure
75230 Paris cedex 05
France

Abstract

This paper reviews various architectures or working principles that enable animats - i.e. simulated animals or animal-like robots - to display adaptive behaviors and that might be useful within the context of computer animation.

1 Introduction

An *animat* [20] is an artificial organism - either a simulated animal or an animal-like robot - the structure and functionalities of which are based substantially on mechanisms observed in biological animals. The behavior of an animat is adaptive as long as it allows it to "survive" in more or less unpredictable and dangerous environments [12].

The study of adaptive behavior in animats is a novel scientific pursuit [13, 14, 1] which has a twofold objective. In the short term, it aims to identify the mechanisms which enable animals to adapt and survive in changing environments. In the longer term, it aims to contribute to our understanding of intelligent behavior in general and of human cognition in particular [10, 11, 21].

On the basis of current knowledge on animal physiology and behavior, there are various means whereby an animat can be endowed with adaptive abilities. Besides the fact that it seems mandatory to equip the animat with sensors and actuators and to interconnect these through an equivalent of a nervous system, further adaptive properties can evidently be gained from the use of a memory, of a motivational and emotional system, and of a planning system. Numerous simulations implement one or more such means.

Whether or not the corresponding simulations faithfully represent the actual mechanisms implemented in a real animal, they might nevertheless prove to be very useful in the context of computer animation provided they really exhibit adaptive behaviors. Indeed, one can easily imagine that it would be of utmost importance that the programmer be as detached as possible from the necessity of specifying every behavior that every "background character" has to exhibit in every circumstance, the task of directing the behavior and the appearance of central characters being already complex enough. In other words, every general architecture or working principle that allows an organism to live in a realistic environment and to switch from one behavior - like moving, feeding, drinking, escaping predators and the like - to another according to some autonomous motivations or to some survival necessity, is of great potential value for computer animation. Furthermore, the more varied the behaviors and adaptive abilities afforded by a simulation program of lesser complexity, the greater this value will be. The remainder of this article will describe some of the architectures or working principles which have been studied within the framework of animat research and which might be used for computer animation.

2 Emergent functionalities

One way to generate adaptive behaviors with rather simple programs consists in letting such behaviors emerge from the interactions between simple modules. An example of this approach is given by Maes [8], who devised a decentralized architecture based upon various behavior modules - which she simply calls *behaviors* - and upon various motivations. Every behavior is characterized by an *activation level* - which is a real number - by a set of *conditions* - which have to be observed in order for the behavior to be executable - by a *threshold* - which should be surpassed by the corresponding activation level for the behavior to be activated - and by an *add list* and a *delete list*. The add list consists of conditions that the behavior is likely to make true. The delete list consists of conditions that are likely to be made false by the execution of the behavior. The different behaviors of an animat, which Maes calls a *creature*, are linked in a network through *predecessor, successor* and *conflicter* links (Figure 1A). There is a predecessor link from behavior A to behavior B if behavior B makes certain conditions of behavior A come true. For example, *explore* behavior may make the conditions of *eat* behavior - that food has to be within the reach of the creature - become true. For every such predecessor link there is a matching successor link in the opposite direction, i.e. from behavior B to behavior A. There is a conflicter link from behavior A to behavior B if behavior B makes a certain condition of behavior A undone. For example, the behavior *flee from creature* may make the condition of *fight* behavior - that there is another creature nearby - untrue.

Every motivation in Maes' architecture has an associated real number representing the strength of that

motivation at any moment in time. A given motivation is associated with one or several behaviors which the creature wants to indulge in when the motivation is high and which reduces the strength of the motivation when they are executed. For example, the *eat* behavior is associated with the *hunger* motivation and decreases this motivation when executed.

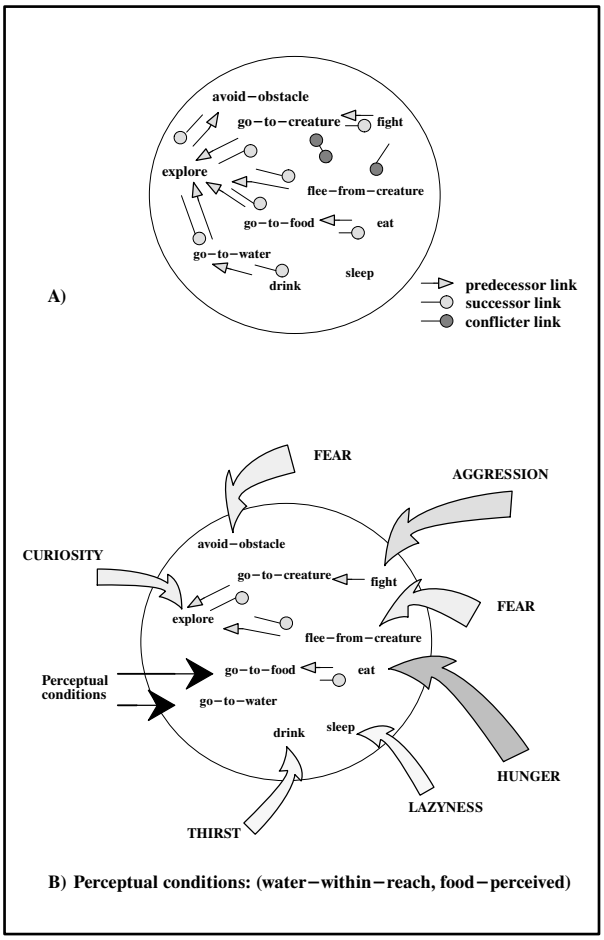

Figure 1: Maes' model. A) Links between behaviors B) A snapshot of the spreading activation process. Adapted from Maes (1991).

Finally, some behaviors are also linked to sensors which are binary-valued. These sensors measure the truth or falsity of logical statements about the environment. For example, the behavior *drink* is linked to a *water within reach* sensor which is set to true if the creature is in the presence of water at a given moment.

The central idea underlying Maes' model is that some sort of activation energy is generated by the current situation and by the creature's motivations. This energy is continuously fed into the links of the behavior network via the behaviors whose condition list partially matches the current situation and via the behaviors which are associated with the motivations (Figure 1B). An executable behavior spreads activation forward and increases the activation level of its successors. A behavior that is not executable spreads activation backwards and increases the activation level of its predecessors. Every behavior decreases the activation level of its conflicters. As soon as the activation level of a behavior becomes higher than the associated threshold, this behavior is triggered. If two behaviors fulfill this condition, it is the one with the highest activation level which is activated. Although the activation level of an activated behavior is reset to 0, such resetting doesn't prevent the behavior from being eventually quickly reactivated, because the spreading activation process goes on continuously.

Figure 1 illustrates the overall architecture of a given creature with ten behaviors and seven motivations. Figure 2 illustrates the behavioral sequence that such architecture generates. The monitoring of the activation levels of the ten behaviors indicates that the nine last boots of activity performed were: *flee from creature, avoid obstacle, explore, avoid obstacle, explore, go to water, explore, go to food*. At this stage, the highest motivation is hunger, and it can be expected that, as soon as the creature reaches food, it will indulge in eating.

Whatever the case, it appears that the control structure regulating when a particular behavior is activated is emergent. There are no global rules or programs that specify how the functionality of behavior selection should be obtained; instead, a dynamics takes place which organizes the behavioral sequences in response to environmental conditions and to the creature's motivational state.

Another example of an emergent functionality [18] is provided by Reynolds [15] who demonstrates that a group of animats, each endowed with a very simple behavioral program, can exhibit an impressive emergent collective behavior. Basically, each animat is simulated as a sort of bird - Reynolds calls it a *birdoid* or *boid* for short - characterized by a direction of motion, a velocity and minimal visual abilities which allow it to detect nearby obstacles or conspecifics. Each boid obeys the following three rules:

- maintain a minimum distance from surrounding boids

- match its velocity with that of the boids in the neighborhood

- move towards the perceived center of mass of the boids in the neighborhood.

Although such rules rely on purely local knowledge - no boid has a global vue of the positions of each obstacle in the environment, nor of the velocities and position of all its conspecifics - the resulting behav-

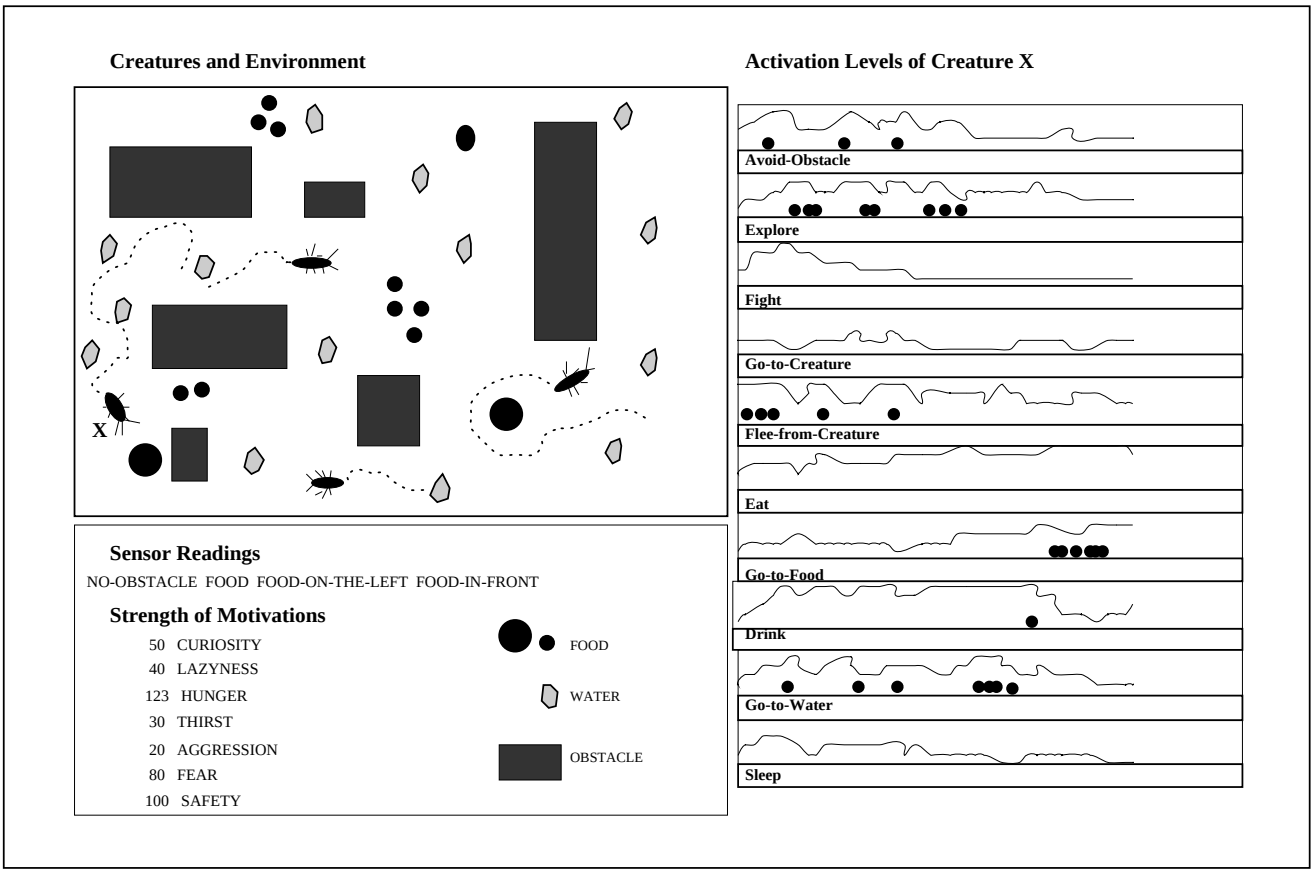

Figure 2: Maes' model. The creatures in their environment. The right part of the Figure shows the activation levels of creature X. Adapted from Maes (1991).

ior of the whole community is that of a flock. Such behavior appears extremely coordinated and realistic, because boids do not collide with each others and because, when the trajectory of an individual eventually diverges from that of the rest of the flock, this individual soon speeds up in the direction of its nearest neighboors and rejoins the group. Moreover, in the presence of an obstacle, the flock eventually gracefully splits into two subgroups, each skirting around one side of the obstacle (Figure 3). Such behaviors are emergent because nothing in the corresponding simulation program expressedly codes for them.

3 Genetic programming

If capitalizing on the emergent properties of a given program appears to be an efficient way to save on programmer's time, another fruitful way to explore is that of letting a program evolve until it generates a desired behavior. An efficient technique for the purpose is that of *genetic programming*, which was pioneered by Koza [7]. Basically, this technique is derived from that of *genetic algorithms* [6, 4] and automatically creates computer programs that satisfy a specified fitness criterion.

The programs sought by genetic programming are symbolic expressions representing compositions of functions. They are expressed as trees with nodes that can be functions or terminal symbols which belong to sets predefined by the programmer according to the problem addressed. For instance, these functions can be simple arithmetic functions, boolean operators, functions specific to the domain under consideration, control structures such as IF...THEN... ELSE or iterative structures like REPEAT... UNTIL. The terminal symbols, similarly, can be variables - such as the state variables of a given dynamic system - or input data, or constants. The genetic programming algorithm simultaneously manages a population of such programs, the quality of each being assessed by the fitness criterion. Such a population evolves from generation to generation because each program in the population has a probability of reproducing proportional to its fitness and because low fitness programs are replaced by better programs. Novelty is introduced in the population when programs reproduce, by means of so-called *genetic operators* like *mutation* or *crossover*.

This type of evolutionary process generally causes programs of ever-increasing fitness to be generated un-

til the optimum value is reached or sufficiently nearly so for all practical purposes.

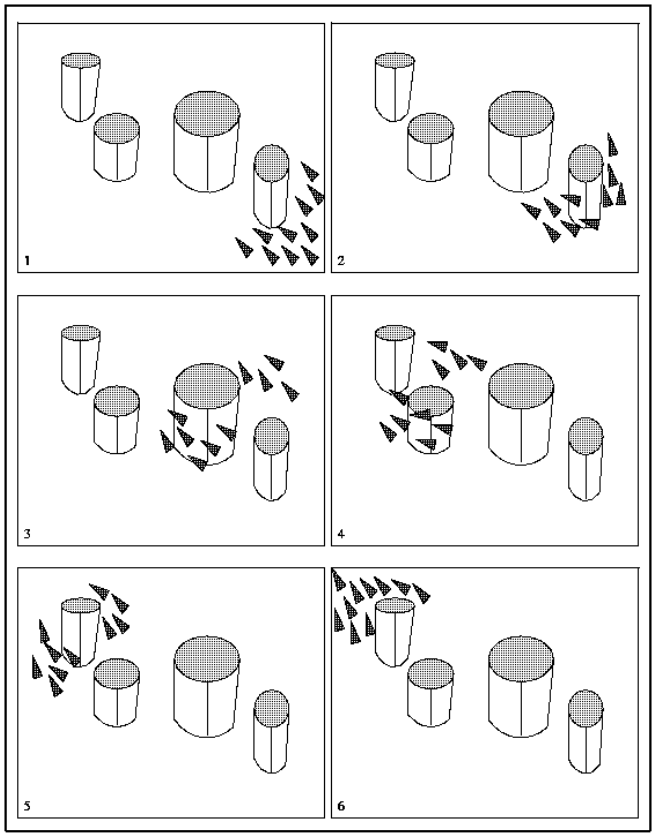

Figure 3: Reynold's boids skirting around obstacles. Adapted from Boids Demo (Video from Symbolics Graphic Division).

Reynolds [16] used genetic programming to evolve a vison-based behavioral model of coordinated group motion in a simulated environment made of a group of preys, some static obstacles and a predator. In order to survive, preys must avoid predation and collisions with obstacles as well as with each other. They must steer a safe path through the dynamic environment using only information received through their visual sensors. The arrangement of the visual sensors, and the connections between sensors and actuators, is determined by the evolved controler program. There are two kinds of motor actions in the prey's repertoire: move forward at a constant velocity and turn at a bounded rate. The predator is controlled by a handcrafted program which doesn't evolve. In general, the predator chases the nearest prey and prefers isolated "stragglers". Because it's velocity is only 95% as fast as that of the preys, the latter can escape by running directly away from the predator. However, some of them, are nevertheless captured because the predator, rather than heading towards its prey's current location, heads to the predicted location of the prey at capture time.

Reynolds assumed that the prey's visual system allows it to distinguish between the three kinds of objects in its environment and accordingly provided the genetic programming algorithm with three functions: *look for obstacle*, *look for friend* and *look for predator*. For example, when *look for obstacle* is called, a ray-tracing operation is performed and a geometric ray is constructed from the prey's position in the direction specified by the sum of the prey's heading and the argument of *look for obstacle*. The value returned is a number between 0 and 1. A value of 1 would indicate that the obstacle is coincident with the prey and that a collison had occured. A value of 0 indicates that the distance between the prey and the closest obstacle in the given direction exceeds a given threshold. The fitness of each control program was evaluated by simulating the strategy it implemented in two sets of initial conditions, by measuring how long each prey could avoid predation and collisions and, then, by averaging the corresponding figures. Occasionally, such raw fitness scores were modified according to some subjective style criteria which would, for instance, favor preys following smooth paths or penalize preys using loop paths.

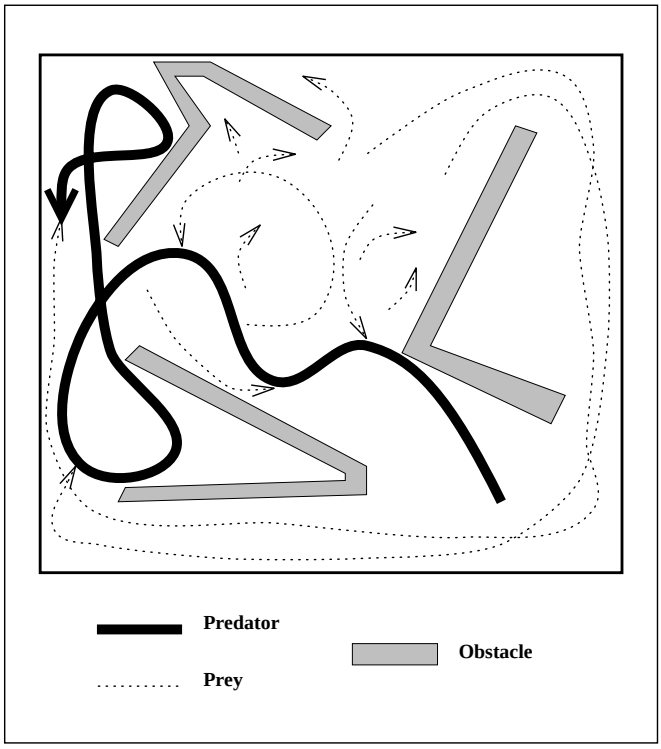

Figure 4: Reynold's predator-prey interaction. Simulation results obtained with an evolved program. Adapted from Reynolds (1993).

Figure 4 shows the simulation results of the best program obtained with a genetic programming algo-

rithm managing a population of 200 programs, after 6600 reproduction steps. Such simulation involved 20 preys and the corresponding control program was:

```
(- (look-for-obstacle 0.01)
   (look-for-predator ( turn (look-for-obstacle 0.01)))
   (iflte (turn (look-for-friend 0.1))
          (look-for-predator 0)
          (- (look-for-friend ( turn (look-for-obstacle 0.1)))
             (look-for-obstacle 0.1)
             (look-for-friend 0.5))
          0))
```

However simple this program may look, analyzing its operation turns out to be quite challenging. Whatever the case, although such control program causes most preys to die early due to predation or collisions, a few survivors manage to escape into the upper right hand corner and swoop around out of sight of the predator. Sharp turns in the predator's path generally correspond to a prey capture and to the selection of another prey. Wide, smooth turns generally indicate that the predator is prowling for a prey.

4 Hierarchical decision structures

Up to now, genetic programming approaches deal with control problems which involve mostly one type of behavior. For example, the problem that Reynold's preys have to solve is only to decide whether to turn or not in order to avoid predation or collisions. It would be extremely interesting to study what kind of control program would evolve, and under which selection pressure, in a case where the animat has to choose from among several actions which one it will exhibit at every moment. For instance, although this problem of *action selection* was solved by Maes by means of a *bottom-up* approach with emergent functionalities, several ethologists suggested that animals solve the same problem by means of a *top-down* approach involving hierachical controlers.

In such perspective, Halperin [5] describes a hierarchical model, inspired from biological knowledge about fighting behavior in Siamese fish, which could be used as a design principle for building sensory-motor interfaces to control a range of motivated behaviors in animats. The architecture of Figure 5 could be used to control the behavior of a hypothetical "scrap labelling" flying animat involved in a litter-collecting system for a park. Such an animat is normally intended to perform the following behavioral sequence: land on a plastic scrap, confirm or reject the hypothesis that this object is plastic, if it is plastic then label it and leave. There is also an over-ride command to pick up the label and leave if the animat has not successfully reached the leave stage within some pre-set time.

This architecture calls upon three kinds of neuron pools. S neuron pools are connected to unspecified sensory devices and respond to stimuli in the environment. For instance, it is supposed that many neurons in S1 are active when the animat's visual image contains features which tend to identify plastic within a given distance range. Likewise, the neuron pool S2 is supposed to be biased to recognize features of plastic which can be detected close up. S neuron pools excite R neuron pools which release behavior by exciting B neuron pools. Fixed inhibitory connections among B pools insure that only one behavior is activated at a time and that, when a new behavior is triggered, it inhibits the previous one. At the same time, there is a positive feedback loop between B and R pools which stabilizes behavior: once activated, two B and R pools fire persistently - even if stimuli in S disappear - until the firing of another B pool inhibits them.

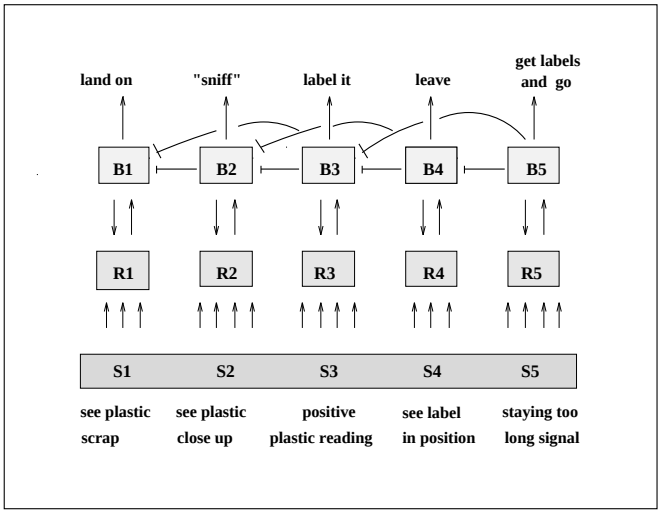

Figure 5: Halperin's sensory-motor controler for a scrap labelling animat. Adapted from Halperin (1991).

One original feature of Halperin's model is that synapses between S and R pools have variable strengths which can be changed under motivated learning. Such learning calls upon a complex *neuro-connector* rule which is related to Hebb's rule and to Sinclair's rest principle. For example, if during the animat's flight, S1 is activated by some object in the sensor's field of vision and if there are currently strong enough connections for R1 to fire, the animat has hypothesized that the object below it is plastic scrap. Therefore, R1 activates B1, and the animat lands on the object. Although the visual pattern that activated S1 disappears, the positive feedback between R1 and B1 maintains their activation. After landing, a new stable image is available again, which will trigger the activation of R2 and B2 if, for instance, the object appears not to be a shiny leaf. The animat will then start sniffing the object and try to confirm that it is plastic. At the same time, inhibitory connections between B2 and B1 will shut off B1 and, since R1 no longer has much S1 input, stopping B1 will stop R1 too. So the whole loop R1-B1 becomes silent and the animat ceases any behavior involved in landing. Because the hypothesis which activated S1 has been confirmed, in

the sense that the object was not a mere shiny leaf, the synapses involved in such an hypothesis are strengthened by the neuro-connector rule. On the contrary, if the animat does land on a shiny leaf, the close-up stimulus is too small to activate R2. Thus B2 remains silent and doesn't inhibit B1. According to such an hypothesis, the R1-B1 loop will be inhibited only after a long delay, when B5 is activated. Because the hypothesis which activated S1 has not been confirmed, the corresponding synapses are weakened by the neuro-connector rule. In these two cases, what makes the distinction between the strengthening and weakening of connections is the length of the time delay which separates S and R shutoffs. Whatever the case, it is interesting to note that, in such model, the success of a given behavior within a behavioral sequence depends upon the release or lack of release of a subsequent behavior. More precisely, a behavior worth reinforcement is one which alters the environment is such a way that a new response becomes possible. In other words, Halperin's control system is sensitive to the consequences of behavior.

In order to compare the relative merits of various controlers for action selection, Tyrrell [19] devised a simulated environment which posed 13 different survival problems to an animat, i.e. obtaining food, obtaining water, keeping clean, regulating body temperature, avoiding predators, being vigilant for predators, staying close to cover, avoiding the boundaries of the territory, avoiding dangerous places, avoiding irrelevant creatures, sleeping at night in a den, not getting lost and reproducing. To solve these problems the animat has a choice of 35 actions that it can undertake, such as drink, clean self, sleep, move north, move south-west and look around. By choosing appropriate actions in appropriate situations, the animat can exert some control over the values of its internal variables - like its levels of food and water - and over its environment - like how well it perceives its local environment or can be perceived by predators. The overall quality of a given action selection controler is measured by an equivalent of genetic fitness, i.e. by the number of times the animat manages to reproduce before it dies. Comparative results obtained by Tyrrell suggest that the most efficient architecture is that of a set of overlapping *free flow hierarchies* [17], each hierarchy being devoted to each survival problem that the animat is confronted with. Such hierarchies are made up of nodes interconnected via weighted connections, which are similar to standard artificial neurons except that their rule for combination of weighted inputs is not necessarily a strict summation. Nodes express multiple preferences for each of a set of lower-level alternatives and the spread of activation between nodes propagates combining evidence from the upper node - which codes for the survival problem to be solved - down to the lowest nodes - which code for the actions which can be executed. Because actions are mutually exclusive, it is the action-level node receiving the most activation at every moment which is activated.

Figure 6A, for instance, describes the free flow hierachy which causes the animat move toward its den as nightfall approaches and then sleep in the den for the rest of the night. The stimulus Night Prox is 0.0 at daybreak, then increases as nightfall approaches and stays at its maximum value through the night. The stimulus Den in Square is 1.0 if the animat is in its den and is 0.0 otherwise.

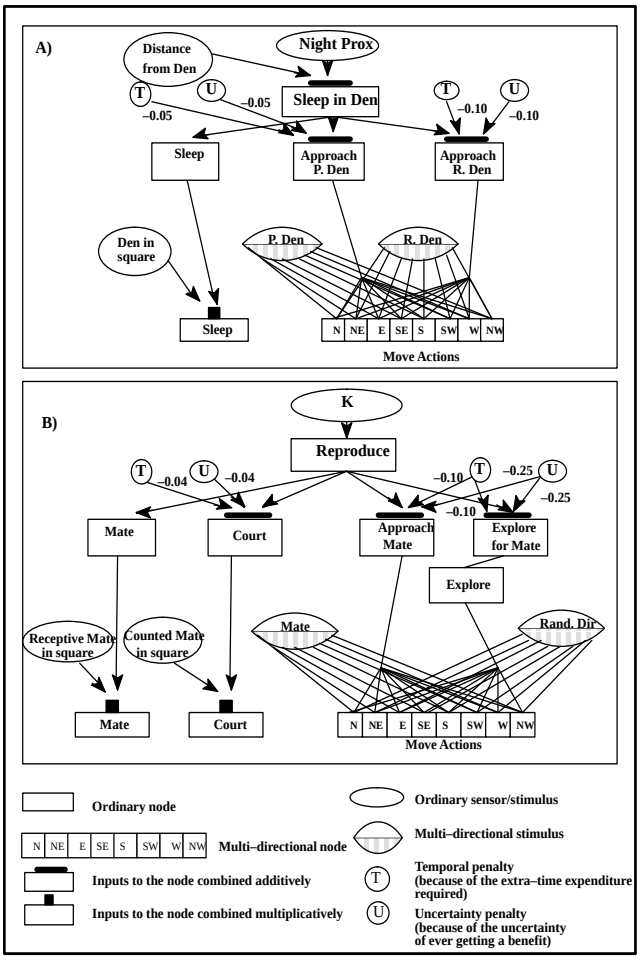

Figure 6: Examples of Tyrrell's free flow hierarchies. A) Control of Sleep in Den behavior. B) Control of Reproduce behavior. Adapted from Tyrrell (1993).

Stimuli P.Den and R.Den respectively code for the perceived and remembered directions of the den. Finally, T and U are temporal and uncertainty penalties which tend slightly to inhibit the Approach Den nodes. Likewise, Figure 6B describes the free flow hierarchy which makes the animat move toward a mate, court

a mate or reproduce. K is a constant stimulus that provides an unvarying motivation to reproduce.

5 Motivationally autonomous animats

As McFarland and Bösser [9] argue, the motivational state of an animal at any particular time depends on its physiological or internal state, on the cue state arising from its perception of the external world, on the consequences of its current behavior and on the expected consequences of its future behavior. The latter point helps characterizing an important functional difference: if the decision to perform a given behavior doesn't take into account the expected consequences of the alternatives, the animal behaves like a *motivated automaton*, otherwise it behaves like a *motivationally autonomous agent*. To do so, the animal requires knowledge of the probable consequences of its acts. In other words, it must have some memory of the past consequences of similar activities, and it must be capable of planning. Furthermore, as Dennett (1983) pointed out, it must want something, it must have goals.

Although various motivational control systems have been described above, they all belong to the category of motivated automata. Even Halperin's controler, which was said to be sensistive to the consequences of behavior, doesn't imply any planning and does not assess the consequences of the various alternatives of a given behavior. In fact, there are no real alternatives to a given behavior - but to give up when the *staying-too-long* signal fires - and such a controler can only exhibit what ethologists describe as *fixed action patterns*, i.e. stereotyped instinctive behavioral sequences.

Donnart and Meyer [3] devised the control architecture of a motivationally autonomous animat. To simplify somewhat the description of this architecture, let us say that it calls on two series of production rules: *action rules*, which take the form:

If <sensory information> and <current goal> then <action>

and *planification rules*, which take the form:

If <sensory information> and <current goal> then <new current goal>

The planification rules generate the various goals the animat seeks to attain, and these goals, in turn, are combined with sensory information to decide what action to perform. Each rule is associated with a *strength*, i.e. a real number which quantifies how successful the rule has been in the past in allowing the animat to get closer to the corresponding goal. These strengths are incrementally updated in the course of the various experiences the animat has in his environment and thus implement a learning procedure.

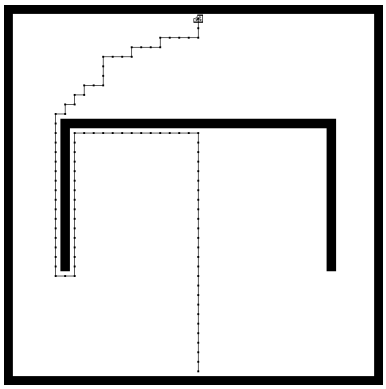

Figure 7: Simulation of Donnart and Meyer's animat. Navigation in an environment with a dead-end.

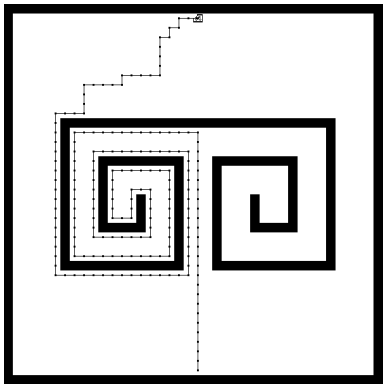

Figure 8: Simulation of Donnart and Meyer's animat. Navigation in an environment with a double spiral.

At any time, there may be many rules whose condition part matches the current situation and which can, therefore, be triggered. The choice of which rule is actually actuated is effected probabilistically, the greater its strength - and thus the most useful a rule has been in the past - the greater its probability of being activated. It is this kind of mechanism which ensures that both the goals the animat wants to achieve and the actions it chooses to perform depend upon the memory of the past consequences of similar choices or activities.

Within the context of a navigation task, the architecture allows the animat to generate plans, i.e. specific paths that a special module abstracts from the various places the animat passes through when trying to reach a given place from a particular starting position. This module detects *salient positions* in the environment which tend to shorten the distance travelled by the animat when it passes through them, and the animat's various goals are either to reach a succession of such places or to skirt around any encountered obstacle. The animat is equipped with three proximity sensors that keep it informed of the presence or

absence of material elements in front of it, 90° to its right, or 90° to its left. It is also able to estimate the spatial coordinates of the position it is located in and the direction of a goal to be reached in each of the eight sectors of the space surrounding it. Lastly, it is capable of moving straight ahead, 90° to its right, or 90° to its left. Figures 7 and 8 illustrate the capacity of the animat to learn to navigate in an environment where obstacles of various shapes can be encountered and its capacity to adapt to changing circumstances. The path of Figure 7 is the one obtained after 50 iterations of the same navigation task in an environment containing a dead-end. The path of Figure 8 is the one obtained when the dead-end is replaced by a double spiral, after 5 additional iterations in the new environment.

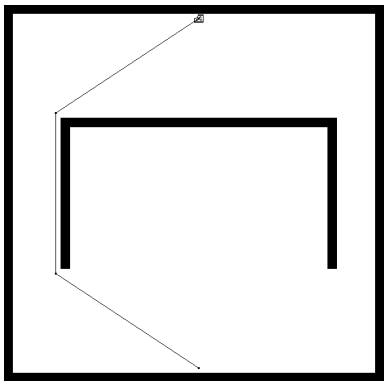

Figure 9: Plan abstracted in the dead-end environment.

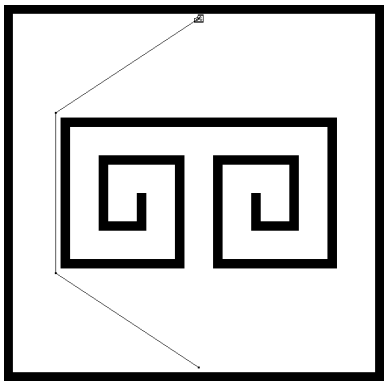

Figure 10: Plan abstracted in the double spiral environment.

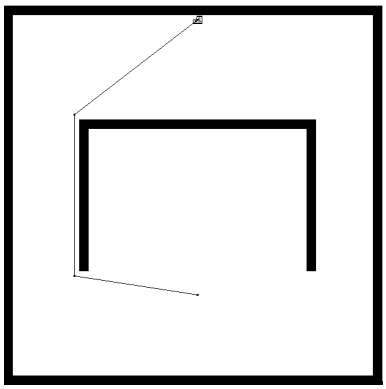

Figure 11: The best plan at iteration 19.

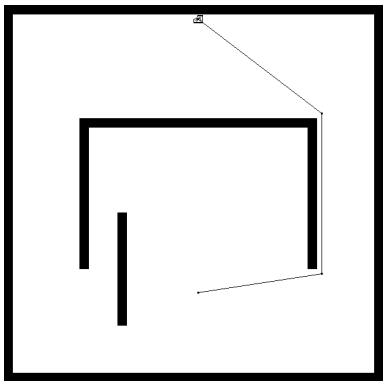

Figure 12: The best plan after a modification of the previous environment (iteration 24).

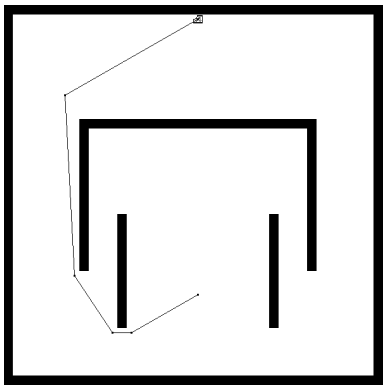

Figure 13: The best plan after a modification of the previous environment (iteration 51).

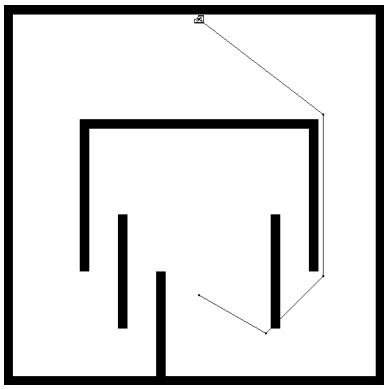

Figure 14: The best plan after a modification of the previous environment (iteration 65).

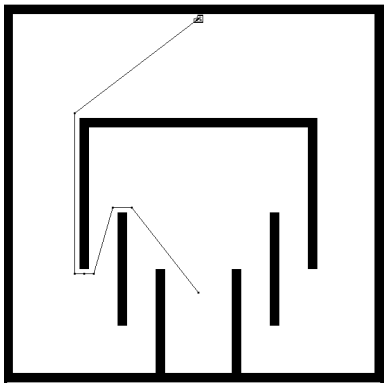

Figure 15: The best plan after a modification of the previous environment (iteration 98).

Figures 9 and 10 show the plans that are abstracted in the dead-end and in the double spiral environments just after completion of the paths shown in Figures 7 and 8. Such plans are simple because they depend on the convex envelope of the obstacles, rather than on their internal complexities. They urge the animat to pass through two specific salient positions and thus to avoid the obstacles. In other words, these plans allow the animat to shorten its navigation paths.

Figures 11 to 15 illustrate the fact that the animat retains several plans in its memory and that it is continuously updating the strengths of its rules. It can therefore rapidly switch from one plan to another, or create new plans, according to the new obstacles appearing in its environment.

6 Conclusions

The means to generate adaptive behaviors in animats are numerous. This diversity raises the question of which kind of controler is best suited to which kind of survival problem. The use of these different means within the framework of computer animation might considerably extend the range of problems traditionally studied in animat research and prove to be of considerable value in assessing the potentialities and shortcomings of these means.

References

[1] D. Cliff, P. Husbands, J.A. Meyer and S.W. Wilson. *From animals to animats 3: Proceedings of the Third International Conference on Simulation of Adaptive Behavior,* The MIT Press, 1994.

[2] D. Dennett, Intentional Systems in Cognitive Ethology: the "Panglossian paradigm" defended. *Behavioral and Brain Science.* 1983, 6, 343-390.

[3] J.Y. Donnart and J.A. Meyer, A Hierarchical Classifier System Implementing a Motivationally Autonomous Animat. *Submitted to SAB94: from Animals to Animats 3. Brighton, U.K.*

[4] D.E. Goldberg, *Genetic algorithms in search, optimization, and machine learning,* Addison Wesley, 1989.

[5] J.R. Halperin, Machine Motivation. In Meyer, J.A. and Wilson, S.W. (Eds.) *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior,* The MIT Press, 1991.

[6] J.H. Holland, *Adaptation in natural and artificial systems,* Univ. Michigan Press, 1975.

[7] J.R. Koza, *Genetic programming: On the programming of computers by means of natural selection,* The MIT Press, 1992.

[8] P. Maes, A Bottom-Up Mechanism for Behavior Selection in an Artificial Creature. In Meyer, J.A. and Wilson, S.W. (Eds.) *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior,* The MIT Press, 1991.

[9] D. McFarland and T. Bösser, *Intelligent Behavior in Animals and Robots,* The MIT Press, 1993.

[10] J.A. Meyer, The Animat Approach to Cognitive Science. In H.L. Roitblat and J.A. Meyer (Eds.) *Comparative Approaches to Cognitive Science,* The MIT Press, 1994a.

[11] J.A. Meyer, Artificial Life and the Animat Approach to Artificial Intelligence. In M. Boden (Ed.) *Artificial Intelligence,* Academic Press, 1994b.

[12] J.A. Meyer and A. Guillot, Simulation of Adaptive Behavior in Animats: Review and Prospect. In Meyer, J.A. and Wilson, S.W. (Eds.). *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior,* The MIT Press, 1991.

[13] J.A. Meyer and S.W. Wilson (Eds.) *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior,* The MIT Press, 1991.

[14] J.A. Meyer, H.L. Roitblat and S.W. Wilson (Eds.) *From animals to animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior,* The MIT Press, 1993.

[15] C.W. Reynolds, Flocks, Herds, and Schools: A Distributed Behavioral Model. *Computer Graphics,* 1987, 21, 25-34.

[16] C.W. Reynolds, An Evolved, Vision-Based Behavioral Model of Coordinated Group Motion. In J.A. Meyer, H.L. Roitblat and S.W. Wilson (Eds.) *From animals to animats 2: Proceedings of the Second International Conference on Simulation of Adaptive Behavior,* The MIT Press, 1993.

[17] K.J. Rosenblatt and D.W. Payton, A fine-grained alternative to the subsumption architecture for mobile robot control. In *Proceedings of the IEEE/INNS International Joint Conference on Neural Networks,* Washington, D.C. 1989.

[18] L. Steels, Towards a Theory of Emergent Functionality. In J.A. Meyer and S.W. Wilson (Eds.) *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior,* The MIT Press, 1991.

[19] T. Tyrrell, The Use of Hierarchies for Action Selection. *Adaptive Behavior.* 1993, 1, 387-420.

[20] S.W. Wilson, Knowledge growth in an artificial animal. In J.J. Grefenstette (Ed.) *Proceedings of the First International Conference on Genetic Algorithms and Their Applications,* Lawrence Erlbaum, 1985.

[21] S.W. Wilson, The Animat Path to AI. In J.A. Meyer and S.W. Wilson (Eds.) *From animals to animats: Proceedings of the First International Conference on Simulation of Adaptive Behavior,* The MIT Press, 1991.