LEARNING IN REAL-TIME ENVIRONMENT BASED ON CLASSIFIER SYSTEMS

Cédric Sanza, Christophe Destruel, Yves Duthen

Université Paul Sabatier IRIT
118 route de Narbonne
31062 Toulouse cedex
France
{sanza,destruel,duthen}@irit.fr

ABSTRACT

This paper presents a new architecture of a classifier system for learning in virtual environments. The model will be integrated in our multi-user platform to provide interaction between intelligent agents and user clones. An agent is an autonomous entity equipped with sensors and effectors. Its behavior is guided by rewards coming from the environment that produce rules called classifiers. The knowledge is shared between agents by using the "sending-message" protocol to increase the global efficiency of the group. The classifier system is specially adapted to a multi-task environment and incorporates a short-term memory to record the recent events of the simulation. These ideas have been implemented and used to develop a virtual soccer where the user plays with autonomous agents that combine communication and evolution.

Keywords: Agent, Evolution, Adaptation, Classifier Systems, Virtual Environments.

1 INTRODUCTION

Learning methods for adaptation and evolution of agents in virtual worlds is a promising research area. In our virtual reality platform [Torgu96], each user evolves through his clone and the contribution of this work allows to simulate artificial creature in order to increase the interaction [Blumb95] between the different virtual actors. For example, an agent could help the user by showing how to achieve to a peculiar task or cooperate with him. Several methods like Q-learning [Watki92] or neural networks [PinaA98] allow to build behaviors for agents to generate computer animation [McCal96], but we here focus on artificial life that uses biological mechanisms to perform learning.

Modeling such agents requires complex behavioral motors and immersion in dynamic environment increases computation time substantially. We therefore propose a system that builds behaviors for autonomous agents [TuXTe94] [MaesP95] for real-time applications. In our model, an agent is coupled to a reinforcement system (a classifier system [Holla91] [Bonar96]) which learns to solve complex tasks by interacting with the virtual world through sensors and effectors. By immersing one or more users in the scene [Pandz97] [Sanza98], we simulate a dynamic world including users' actions in order to add unpredictable changes into the scene.

With a classic classifier system, the evolution is guided by rewards computed with an evaluation function (called "fitness"). But this evaluation function is particularly difficult to achieve in a multi-tasks environment. Indeed, it has to be precise enough to ensure convergence, but it must not be too precise to allow evolution. Starting from this remark, we have decomposed the initial function into a sum of functions that represents a goal to achieve. However, this average is insufficient because it is not significant: one may have more influence than the other depending on the situation. We have solved this problem by using a single unit of neural network. Each function is weighted by a coefficient that is modified dynamically by back-propagation, according to the value of the function and the current context (sensors in the world).

An autonomous agent should also remember the recent events of its life or more precisely the result of the triggering of every rule to eventually re-use an old but efficient rule, and thus to face unexpected

events. Thanks to a short-term memory, rewards from the environment are recorded in the corresponding rule.
The following section presents artificial life notions (genetic algorithms) and our classifier system. After the presentation of the learning model, we will describe the first application: a virtual soccer game where agents play in two separate teams including human immersion and interaction.

2 ARTIFICIAL LIFE TOOLS

2.1 Genetic Algorithms

In 1975, Holland [Holla75] proposed a new technique of problem resolution in complex spaces: genetic algorithms. Founded on mechanisms of natural selection, they use principles of the best-adapted structure survival and information exchanges. Every generation, a new set of "artificial creatures" (solutions) is created from the best elements of the previous generation, thanks to genetic operations: reproduction, crossing and mutation. An evaluation function (fitness) rewards the best solutions that will be selected to transmit their genetic inheritance. The convergence is then expressed by a concentration of the population in the regions of space that present a global optimum of the evaluation function.

2.2 Classifier Systems

A classifier system [GoldbB91] [Donna94] is an adaptive system that learns syntactical rules called classifiers. A classifier is a production rule consisting of a bit string. It is composed of two parts: condition(s) and action and it is valued thanks to its strength. The evolution is guided by rewards that come from the environment. It is managed by a genetic algorithm and is almost continuous. It enables to record efficient rules and to reuse them at appropriate times, unlike genetic algorithms that look for the optimal solution to a unique problem in a finite time. Moreover, genetic mechanisms allow the combination of the best rules and evolution.

A classifier system consists of three main components:
- the “rules-basis” where all the information acquired (sensors) produces one or several actions (effectors) after a selection of the best classifier,
- the reward system (generally the “Bucket Brigade” algorithm) where information is bought and sold by classifiers in order to link sensors and effectors,
- the genetic algorithm that produces new rules, using reproduction, crossing-over and mutation from selected rules and adds them in the “rule-basis” for evaluation.

3 STRUCTURE OF THE CLASSIFIER SYSTEM

3.1 Presentation

Initially based on Holland structure, our classifier system (figure 1) has been modified on three main points: the representation of a rule, the internal reward system and the recording of the rules. We have also adapted it to a multi-task environment and to knowledge sharing. We will describe these mechanisms and their interests in the following section.

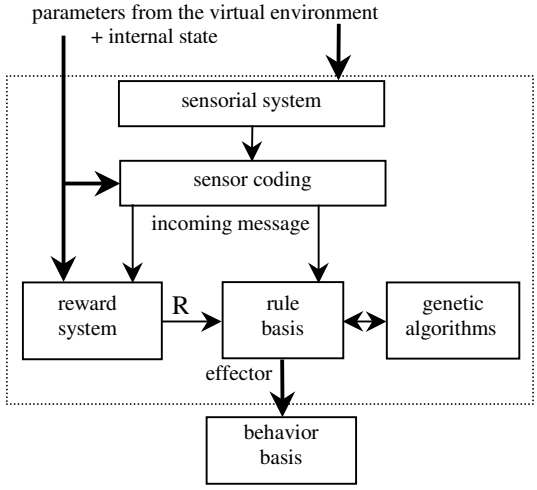

Figure1. Architecture of the classifier system

In this model, all the messages from the environment are boolean values. They are all used simultaneously to compose the incoming condition. Every sensor, producing a “1” (if “true”) or a “0” (if “false”) is concatenated in the sensor coding module to build the binary string of the incoming message that will trigger a classifier in the “rule-basis”.

3.2 Representation of a Rule

The condition part of a rule is then a binary string which length is the number of sensors. From the incoming message, the system will look for the best correspondence with the condition part of every classifier to select a rule.
The action part of a rule corresponds to a behavior. It is a bit string composed of B binary sub-strings (b is the number of different behaviors). Each sub-string “i” gives the binary strength of the behavior “i”. The selected behavior is the one that is the more powerful according to the value of the sub-string. With this example “00|01|10” the winner is the behavior 3. The corresponding rule will activate the third behavior. In case of equality, the priority is given to the first behavior: “01|10|10” gives the behavior 2. These mechanisms imply different

implicit probabilities for every behavior. In the precedent example, with 3 sub-strings of 2 bits, the probabilities of the behavior 1, 2 and 3 are respectively 30/64, 20/64 and 14/64. But a simple calculation shows that increasing the length of the sub-strings reduces the difference of probabilities between the behaviors. Adjusting the probability of a behavior allows to control the hierarchy and then the appearance of every behavior in the basis.
The "don't care" symbol # is generally used in classic classifier system to represent any single bit. With this representation of a rule, it would be interesting to use it in the condition part of a classifier. Thus, we will integrate this character in the future version of the classifier system in order to group the rules that produce the same behavior and to reduce the length of the base.

Compared with classic classifier systems, there are two main advantages of this representation of a rule. First, the system works with a single message containing all the information about the world (global vision). Second, genetic operators (especially crossover) are applied on a longer and significant genotype. The drawbacks may be the length of the classifier when there are a lot of sensors and effectors.

3.3. The Reward System

The reward system (figure 2) is the main part of the convergence of the system. Generally, a "fitness" function evaluates a classifier according to effects in the virtual world.

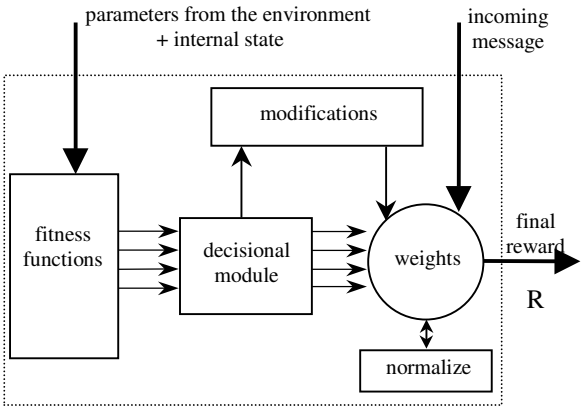

Figure 2. The reward system

However, defining a "fitness" function is very difficult: if it is too precise, there is no possible evolution. On the other hand, if it is not precise enough, the convergence will be very long. Moreover, it is impossible to define it in a multi-task environment.
To solve this problem, a new reward system has been designed. Instead of a single reward function, we work with several functions. Each "fitness" represents a simple goal [MaesP90] that the system has to achieve. These fitness are computed with two types of parameters: some are from the agent (internal state) and others come from the environment (other agents, context). At a given state (incoming message "im"), at a given time "u", a fitness "i" is associated to a variable weight "i" for the sum of the final reward R (equation 1).

$$R(u) = \sum_{i=1}^{n} weight_{i,im}(u) \times f_i(u)$$

where n is the number of functions to maximize

This formula simulates a single unit of neural network: every "fitness" is weighted by a value in [0;1]. The use of a weight for each fitness is essential. Indeed, considering these results {130,-10,-10,-10}, the average is 25. However, what we want to achieve is to give more power to 130 to obtain a better reward (-10 is a bad but low reward). Increasing and decreasing weights by back-propagation simulates this mechanism. These modifications are triggered by the decisional module where a weight is modified as shown in the formula 2:

$$weight_{i,im}(u) = weight_{i,im}(u-1) + W(reward(u))$$

"W" is the polynomial function represented on the following figure:

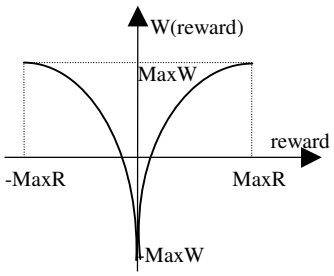

Figure 3. Addition applied to a weight

In the simulation, MaxW (maximum modification) is set to 0.1 and MaxR (maximum reward) is set to 200, but the add is applied so that a weight stays in the interval [0;1].
A really good or bad reward will produce the maximum for "W". It will give more influence to the final reward, to increase it as well as to decrease it. An insignificant reward (close to 0) will decrease the power of the function by decreasing the corresponding weight. While another function is satisfied, the global reward will remain high.
Every iteration, weights are normalized to avoid incoherent results (blocking of the system). For example, if every weight is equal to zero, the final reward is 0, it is necessary to re-initialize the weights.

3.4 The Classifier Selection

This module selects the best appropriate rule in the system, according to its structure (message) and its actual strength. At the start, every classifier is a potential candidate for the selection to be triggered. The determination of its pay-off is computed with the function P (equation 3).

$$P(u) = S(u) \times \left(1 + \frac{nb_cond}{\lg_cond}\right) \times C \times F \qquad (3)$$

$$S(u) = \frac{\sum_{j=0}^{t} coef(j) \times strength(u-j)}{\sum_{j=0}^{t} coef(j)} \qquad (4)$$

where
u : time when the rule was triggered
strength(u): strength at the time u
t: number of steps considered (time)
coef(j) = t-j+1 if *strength* (j)>0 otherwise 1
nb_cond : number of bits that "fits" to the incoming message
lg_cond : length of a message
C : parameter 1 or 0.
F: parameter 0.3

The notion of "fit" is determined by a comparison of the binary messages. The *nb_cond* term represents the number of concordances between the incoming message from the environment and the message part of the classifier. A concordance is defined as a notion of inferiority. Each bit of the classifier must be inferior to the corresponding bit of the incoming message. For example, the condition "0011" activates different messages like "0010" or "0001" but not "1011" and the latter is not considered for the selection by setting the parameter C to 0. In general, all the rules that satisfy a false condition are eliminated and the others increase or decrease their strengths according to their capacity of modeling the context.
The last term F is a parameter (we have used a value of 0.3 for the first experiments) producing the fraction of pay-off given by the classifier.
On the first tests, we only took into account recent strength but it is useful to memorize the past of a classifier. So, we have introduced a recording of every strength (function S, equation 4) corresponding to the time when the rule was triggered:. This historic memory of a classifier refines the global strength for the final pay-off and reduces the effects of a parasitic strength.
Each strength is weighted (by "*coef*") to give power to recent and efficient strengths thus avoiding negative effects of a low reward due to a punctual event. The global reward will not be influenced. Decreasing with time, weights favor recent strengths that are more significant, thanks to the convergence of the system.
After the selection, the winner will trigger the corresponding effector. At the time u, its strength will evolve (equation 5) by adding the reward from the environment R, and by subtracting the pay-off P.

$$strength(u) = strength(u\text{-}1) + R(u) - P(u) \qquad (5)$$

3.5 Hierarchy Between "Fitness" Functions

Another advantage of the use of the weights is the possibility to define a priority between "fitness" functions. Let us consider this new example of a constant reward of two functions: {200,-200}. According to the formula of a weight, the evolution will produce {1,1} for the two weights (the maximum) The final reward will be 1*200+1*(-200)=0. Here, there is no distinction between the two functions. If the first function is more important than the second, by keeping a maximum reward to 200, we just have to modify the maximum limit for a weight: the common constant "1" becomes the variable value Lim_i . In the precedent example, we modify Lim_i in order to have $Lim_1 > Lim_2$. The final reward is then strictly positive.
The use of Lim_i allows to minimize the effects of selected functions that are not very significant. Without this parameter, it would be impossible to take them into account as they sometimes disturb the final reward. This mere mechanism represents the final reward as better as possible by using these hierarchical constraints.

3.6 Genetics Algorithms

We use genetics operations (mutation and crossover) to create new rules from the elements of the current population. These elements are extracted from the base by "the tournament selection". This algorithm is a tournament between n classifiers randomly chosen in the base: the winner is the rule with the highest strength. Using this method, every rule has a chance to transmit its gene to a future element. Choosing the winner in the entire basis is very dangerous here: if there are only one or two very good classifiers, they will always be chosen and these rules will monopolize the basis, causing a loss in diversity and reducing the chances to find new classifiers.
In the simulation, the crossover rate was set to 5% and the mutation probability was set to 15%. As the number of rules is fixed, creating new rules implies destroying old ones. To keep the best classifiers, the eliminated rules are randomly chosen among the worse ones. So, on every classifier, a flag indicates the degree of usefulness (bad, good and not yet tested) to preserve the efficiency of the basis.

3.7 Optimization of Computation Time

Reducing computation time is essential for real-time applications. We have adapted our system on two main points.
The first one concerns the callback to the classifier selection. An efficient rule is used as long as the incoming message is unchanged (context of the world) and the average reward is positive. The classifier selection operates then at appropriate times. The computation of P (equation 3) for each classifier is very complex, it would be useful to compute it each time the classifier is activated. The reward system and the sensorial system are the only modules that work during the use of a "good" classifier.
The second point concerns the genetic algorithm (GA). The GA combines and creates new rules according to an adaptive frequency. This frequency depends on the final reward of the system. The GA is very active at the beginning, especially when the system tries to find a rule (rewards are generally very low). It allows to have a lot of diversity in the basis to find the best rule. Callbacks then decrease: a rule that will never be triggered does not need to be created.

4 STRUCTURE OF AN AGENT

4.1 Component of an Agent

An agent is an autonomous entity without initial knowledge about its environment. Its behavior is defined by a classifier system that is connected to the agent through the sensorial system and the behavior basis. All along the simulation, the agent builds its behavior according to the rewards of its "fitness" functions. By simply implementing what an agent can do and what he can feel, the system can create coherent links between conditions and actions and put these rules into practice for continuous evaluation.

4.2 Communication Between Agents

The sending-messages protocol (figure 4) is used to allow communication. This way to dialogue seems to be adapted to a classifier system. Indeed, rule-basis works on binary rules that are transmitted as messages.
Every time an agent has a difficulty to act correctly, the supervision module (that gives statistics about the way rewards evolves) triggers a time-out that send a message to all other agents. The agent waits for a solution to test it immediately.
The communication system is connected to the rule-basis. Indeed, a question corresponds to a condition and an answer to an action (effector). A response is determined by searching in the basis a "good" classifier with the same condition part. If it exists, the action part is send as a message.
This rapid system allows to share knowledge between compatible agents. Indeed, the main problem in communication is to manage with different knowledge. Initially, if all the agents do not have the same sensors and effectors, two identical binary rules may be interpreted differently. This parallel evolution implies a communication with entities that are equipped with the same sensors and effectors.

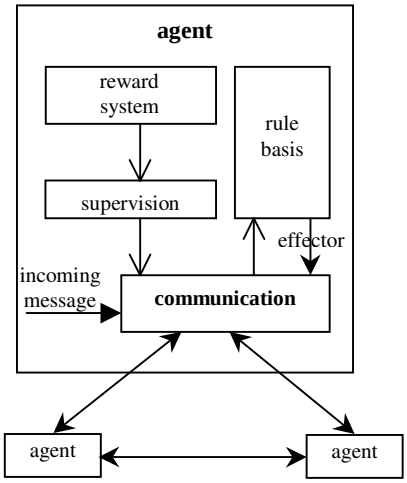

Figure 4. Data exchanges between agents

5 FIRST TESTS

5.1 Description of the World

By using a generic oriented-object architecture and the WorldToolKit platform, we have implemented a virtual soccer game where the user evolves through his clone in a 3D world (figure 5).

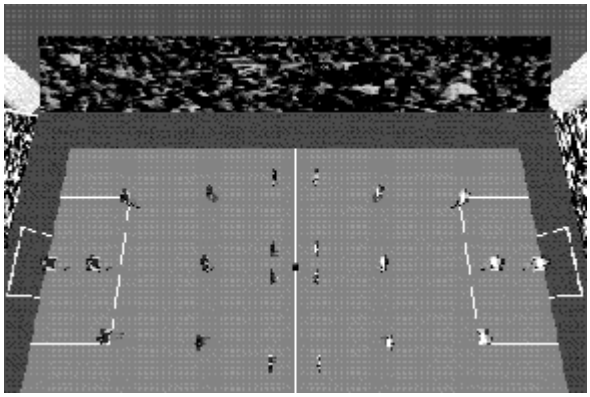

Figure 5. The virtual soccer

The other players are autonomous agents with constraints about moving. The most natural one is collision detection with virtual actors. The second

one avoids regrouping around the ball: so, an agent must stay in a circle defined by his first position and a constant radius depending on its role in the game.
The user clone is seen as an agent by the others and so can execute the same actions with the spacemouse that offers 3D moving possibilities.

5.2 Learning

The following list shows the sensors and effectors of an agent , according its role in the team: goalkeeper (number 1), attacker (number 2) and defender (number 3).

Sensors are: has-the-ball (1,2,3), doesn't-have-the-ball (1,2,3), near-from-other-goal (2,3), far-from-other-goal (2,3), near-from-my-goal (1), far-from-my-goal (1), team-has-the-ball (1,2,3), team-member-near (2,3), adversary-near (2,3).
Effectors are: go-to-ball (1,2,3), go-to-goal (3), go-to-my-goal (1), go-between-player-ball (2), go-between-player-goal (3), pass-strong-to-player (1,2,3), pass-weak-to-player (1,2,3), shoot-strong-to-goal (2), shoot-weak-to-goal (2).
Each agent has 2, 3 or 4 "fitness" functions defined below:
F1 (2,3) is Dist(t-1)-Dist(t) where Dist(t) is the distance of the ball to the goal.
F2 (1,2,3) is Dist(t-1)-Dist(t) where Dist(t) is the distance of the agent to the ball.
F3 (1) is Dist(t-1)-Dist(t) where Dist(t) is the distance of the agent to his own goal.
F4 (2,3) determinates if a team has a ball (reward 100) or not (reward -10).
F5 (1,2,3) is a linear function that maximizes the position of the ball in the game.
F6 (2,3) is Dist(t)-Dist(t-1) where Dist(t) is the distance to the closest adversary.
F1,F3 and F6 are local functions: an agent will use its internal state to compute it, the behavior of the other agents here does not matter. On the contrary, F2, F4 and F5 are global functions [Panat98]. They represent the state of the group. This kind of function makes the team progress: some agents get positive rewards without acting directly on the ball.

6 RESULTS

6.1 Global Performances of the System

The system has been tested with two teams of eleven players. Performance is measured in figure 6 as the percentage of correct actions of three agent during a simulation.
A positive reward of every fitness let us consider the action is correct.
Every arrow corresponds to a new situation that the agent perceives. The system finds a solution after testing a few rules. Indeed, genetic operators (crossover and mutation) preserve a high diversity in the base to offer different choices for the same context. Several rules may be triggered until the system finds an efficient classifier. Performances then increase as the rule is used.

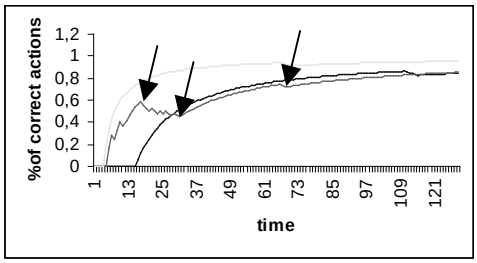

Figure 6. Percentage of correct actions

6.2 Analysis of a Sequence

Let us now analyze a little sequence from the simulation. The figure 7 represents the global reward of an agent that moves towards the ball (arrow 1) and kicks on it (arrow 2) successively. We consider two "fitness" functions F1 and F2 in the precedent list. Lim_1 is set to 1 (maximum value for the $weight_{1,im}$), Lim_2 is set to 0.5 so that F1 is more important than F2.
In the phase 1, the agent needs time to go to the ball's position, the phase 2 is more punctual. Using two rules, the system always gets positive rewards (in average) by pushing the ball to the goal.

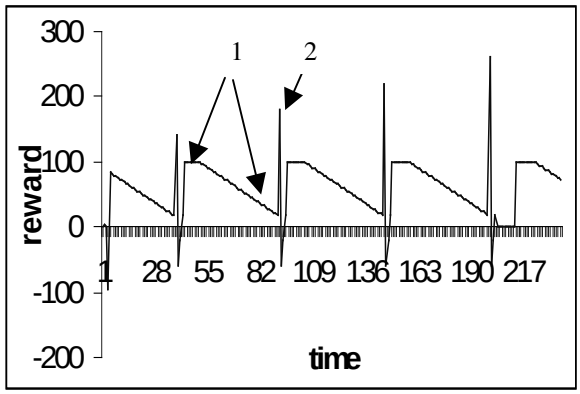

Figure 7. Reward of an agent

Generally, when an efficient classifier is found, weights converge to a constant value (Lim_i or 0). This mechanism allows to record which function is maximized in which context: the specificity of a rule to a function implies that weights converge to the set { Lim_i, 0}. Indeed, the weight corresponding to the maximized function "i" is set to Lim_i and the others are set to 0.
The figure 8 gives in detail the evolution of the reward R .The sequence is composed of four parts. The first and the second part correspond to the

behavior 1 (high priority): the agent moves toward the ball, using the same classifier. The function 2 gives the maximum reward in the first phase. The state 1 (incoming message number 1) gives respectively 0 and 0.5 for *weight*1,1 and *weight*2,1. In the third phase, the agent kicks the ball (behavior 3). In the state 2, weights allow to consider the function F1, the corresponding weight (*weight*1,2) reaches the maximum and the other the minimum. The fourth phase starts with two negative rewards, this parasitic effect is due to the movement of the ball that continues to move. This problem does not disturb the system, the rule will loose a little part of its strength and the average reward keeps a positive value for R. When the ball stops, the weights take their initial value and the cycle starts again.

state	W11	W12	F1	W21	W22	F2	R
1	0.0		0.0	0.5		200.0	100.0
1	0.0		0.0	0.5		100.0	100.0
1	0.0		0.0	0.5		200.0	100.0
1	0.0		0.0	0.5		200.0	100.0
1	0.0		0.0	0.5		195.28	97.64
1	0.0		0.0	0.5		179.27	89.63
1	0.0		0.0	0.5		163.27	81.63
1	0.0		0.0	0.5		147.27	73.63
1	0.0		0.0	0.5		131.27	65.63
1	0.0		0.0	0.5		115.27	57.63
1	0.0		0.0	0.5		99.27	49.63
1	0.0		0.0	0.5		83.27	41.63
1	0.0		0.0	0.5		67.27	33.63
1	0.0		0.0	0.5		51.27	25.63
2		1.0	200.0		0.0	0.0	200.0
1	0.2		200.0	0.5		-200.0	-60.0
1	0.4		200.0	0.5		-200.0	-20.0
1	0.2		0.0	0.5		200.0	100.0
1	0.0		0.0	0.5		200.0	100.0

Figure 8. Evolution of the rewards

The two classifiers used for the precedent sequence are:
- rule 1: if (has-the-ball) and (far-from-other-goal) and (team-has-the-ball) then (shoot-weak-to-goal)
- rule 2: if (doesn't-have-the-ball) and (far-from-other-goal) then (go-to-ball)

Advantages of the preliminary coding of the incoming messages are shown here: a simple rule is composed by a non-fixed number of conditions. A classifier then takes into account not only a message from the environment but all the information. Data exchanges with the environment are reduced to the maximum: a single binary message. This particularity is very interesting to produce real-time systems.

7 CONCLUSION

In this paper, we have presented the first version of our learning system for agents immersed in a virtual environment. The most interesting features of the model are the possibility to define a hierarchy between goals ("fitness" functions), the representation of a rule that offers a global information about the context and the efficiency on real-time tasks.

The future step is now to evaluate the effects of the communication on a single agent and in the team behavior. We will integrate this study in our virtual reality platform: VIPER [Torgu96] in order to build a multi-user application. An interesting possibility for further development is to increase the interaction between user clones and autonomous agents in order to provide an adaptive learning system. The agents would evolve according to the user's actions to create a cooperative behavior [Salus98].

REFERENCES

[Blumb95] B. Blumberg, T. Galyean "Multi-Level Direction of Autonomous Creatures for Real-Time Virtual Environment". Proceedings of Siggraph 95. Computer Graphics Proceedings. August 1995.

[Bonar96] A. Bonarini "Evolutionary Learning of Fuzzy rules: competition and cooperation". In W. Pedrycz (Ed). Fuzzy Modelling: Paradigms and Practise. Kluwer Academic Press. Norwell. MA. 1996.

[Donna94] J. Donnart, J. Meyer "A Hierarchical Classifier System Implementing a Motivationally Autonomous Animat". From Animals to Animats 3: Proceedings of the 3rd International Conference on Simulation of Adaptive Behavior. Brighton. UK. pp 144-153 August 1994.

[GoldbB91] D. E. Goldberg "Genetic Algorithms". Addison-Wesley USA. 1991.

[Holla75] J. Holland "Adaptation in natural and artificial systems". University of Michigan Press. Ann Harbor 1975.

[Holla91] J. Holland "Escaping brittleness: the possibilities of general-purpose learning algorithms applied to parallel rule-based system". R.S. Michalski, J.G. Carbonell, T.M. Mitchell. Machine learning, an artificial intelligence approach, Vol.II. Morgan Kaufmann 1991.

[MaesP90] P. Maes "Situated Agents Can Have Goals". Journal for Robotics and Autonomous Systems 6(1&2). pp 49-70. 1990.

[MaesP95] P. Maes "Modeling Adaptive Autonomous Agents". Artificial Life : An overview. MIT Press 1995.

[McCal96] A. McCallum "Learning to use Selective Attention and Short-Term Memory in Sequential Tasks". From Animals to Animats 4: Proceedings of the 4th International Conference on Simulation of Adaptive Behavior. North Falmouth. USA. 1996.

[Panat98] C. Panatier, H. Luga, Y. Duthen "Collective learning for Spatial Collaboration". From Animals to Animats 5: Proceedings of the

5th International Conference on Simulation of Adaptive Behavior. Zurich. Switzerland. August 1998.

[Pandz97] I. Pandzic, T. Capin, N. Magnenat Thalmann, D. Thalmann "A Versatile Navigation Interface for Virtual Humans in Collaborative Virtual Environments". Proceedings of the Symposium on Virtual Reality Software and Technology 1997. Lausanne. Switzerland. Sept 1997.

[PinaA989] A. Pina, F. Seron "Obtaining Autonomous Computer Animated Actors using Fuzzy Experts Systems and Neural Networks". Proceedings of the 3rd International Conference on Computer Graphics and Artificial Intelligence. France. April 1998.

[Salus98] R. Salustowicz, M. Wiering, J. Schmidhuber "Learning Team Strategies: Soccer Case Studies". Machine Learning. Kluwer Academic Press. Boston. 1998.

[Sanza98] C. Sanza, C. Destruel, Y. Duthen, "A learning method for adaptation and evolution in virtual environments". Proceedings of the 3rd International Conference on Computer Graphics and Artificial Intelligence. France. April 1998.

[TuXTe94] X. Tu, D. Terzopoulos "Artificial Fishes: Physics, Locomotion, Perception, Behavior". Proceedings of Siggraph 94. Computer Graphics Proceedings. 1994. pp 327-351. Orlando. USA. July 1994. pp 43-50.

[Torgu96] P. Torguet, F. Rubio, V. Gaildrat, R. Caubet "Multi-User Interactions in the Context of Concurrent Virtual World Modeling". 3rd Eurographics Workshop on Virtual Environments. Monaco. Feb 1996.

[Watki92] C. Watkins, P Dayan. Q-learning. Machine Learning, 8(3) pp 279-292, 1992.