CLASSIFIER SYSTEMS AS 'ANIMAT' ARCHITECTURES FOR ACTION SELECTION IN MMORPG

Gabriel Robert* ,**, Pierre Portier** and Agnès Guillot*
*AnimatLab, Laboratoire d'Informatique de Paris 6, 8 rue du Capitaine Scott, 75015 Paris, France
**Nevrax France, 104 Rue du Faubourg St. Antoine 75012 Paris, France
E-mail : {gabriel.robert ;agnes.guillot}@lip6.fr; portier@nevrax.com

KEYWORDS
Learning classifier systems, action selection, autonomous agents, video game.

ABSTRACT

Classifier systems (CS) are used as control architectures for simulated animals or robots in order to decide what to do at each time. We will explain why these systems are good candidates for action selection mechanisms of Non Player Characters. After having described different classifier systems, we will introduce a new CS architecture, acting in a multi-agent environment, which is adapted to the specific constraints of the 'Massively Multi-players Online Role Playing Games'.

INTRODUCTION

A new Artificial Intelligence approach focuses on the synthesis of adaptive simulated animals or real robots (called *animats*), the inner mechanisms of which being as much inspired from biology and ethology as possible (Guillot and Meyer 2000). An animat has both sensors – which provide information about its environment or internal states - and effectors – which make it possible to change its environment. In order to be able to survive, it is endowed with a control architecture that connects its sensors to its effectors, such architecture being occasionally adapted to changing circumstances through unsupervised learning.
Massively Multi-players Online Role Playing Games (MMORPG) are new games in which thousands of players interact with each other and with non-player characters (NPC) in the same continuous and persistent world (e.g., Everquest ©*Verant Interactive*, Asheron's Call ©*Turbine Games*, or Dark Age of Camelot ©*Mythic Entertainment)*. NPCs behaving in these games are comparable to animats, because these artificial creatures have to adapt on line to dynamically changing environments, to new goals assigned by game-designers, and to unpredictable actions from the players.

The control architectures developed by the animat community are useful to afford adaptive behaviours to a NPC. In particular, one kind of model - the so-called *Classifier Systems* (CS) - is especially convenient to design architectures able to efficiently select which actions the NPC should perform. A CS is a population of 'condition-action' rules called classifiers (Holland 1986). A CS can learn which classifier is better suited than another to achieve a given task. New rules can also be discovered through the creation of new classifiers.
In this paper, we will introduce different categories of CS used in the animat approach that could prove to be applicable to NPC action selection in video games. We also propose a new architecture, based on hierarchical CS and specifically tailored to Ryzom, a MMORPG developed by Nevrax.

CLASSIFIER SYSTEMS

A CS contains a *classifier list*, i.e. a pool of 'condition-action' rules, the classifiers (Figure 1). At initialization time, this list is generally hand-designed. Three parts characterize a classifier. The first one, the *condition* part, corresponds to the environmental information received by the animat sensors, and expressed as a string defined by a ternary alphabet {0,1,# : false, right, don't care}. The second part is an *action* command. The last part is a '*fitness*' value, a quantitative measure of the classifier's past successes or failures.

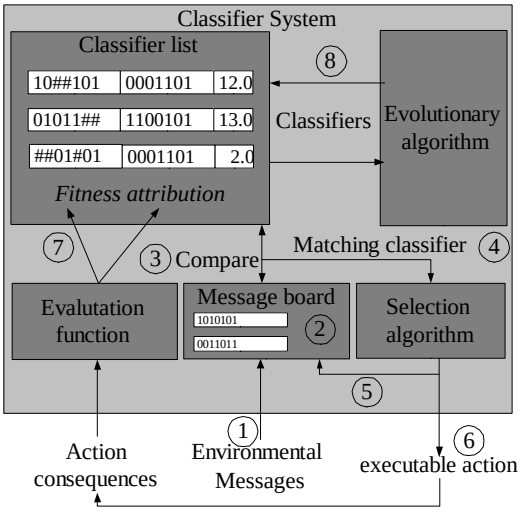

Figure 1. A Classifier System (see text for explanation)

When an animat detects some environmental features (1), it encodes this information into a 'string' of {0,1, #} that it deposits on the *message board*, together with other possibly internal messages (e.g. motivation) For example, if an animat is near a river and a dragon, its three sensors specific for water, food and predator will send the message {1,0,1} (2). This message is compared to the condition part of each classifier in the classifier list (3). A selection algorithm chooses one classifier among those whose condition part matches to the current message (4). The corresponding action command is either directly sent to the effectors, or deposited on the message board (5). In the latter case, the corresponding action message may be matched to the condition part of other classifiers, and the process returns to step (3). In the former case, the behaviour corresponding to the activated effectors is displayed in the environment (6).
A CS has two adaptive mechanisms. On the one hand, each time an action command is executed, the fitness value of the corresponding classifier is incremented or decremented relative to the resulting positive or negative outcome (7). If, at step (3), several classifiers are selected at the same time, the classifier with the highest fitness value has the greatest probability of being activated. This reinforcement learning process - well-known in ethology - allows an animat to efficiently associate given classifiers to given tasks. On the other hand, new classifiers may be created by an evolutionary algorithm (e.g. a genetic algorithm, see Holland 1975), according to so-called *mutations* and *crossovers* operators acting on classifiers with high fitness values. Other classifiers may be removed from the list if they are associated with low fitness values (8).

Mac Namee and Cunningham (2001) have asserted that a good action selection mechanism for a video game must be reactive (i.e., agents behave by means of event-action rules), proactive (i.e., agents exhibit goal-directed behaviour), and autonomous (i.e. agents do not call upon player or game master intervention), as well as configurable and extensible by a non-programmer, like a game-designer. It turns out that a classifier system affords these specific properties. Indeed, with such control architecture, an animat is reactive, as some classifiers are simple S-R rules. An animat is proactive, as the classifiers can code internal needs and desires. An animat is autonomous, as it can empirically build, through learning or evolutionary process, an efficient classifier list. Finally, a game designer can easily configure, change or extend the behavioural repertoire of the animats, because a classifier is written in a classical video game formalism, i.e., "if condition then action" rules.

A huge variety of CSs has been proposed in the animat literature (see Kovacs 2002, for a review). We will introduce here the main systems only.
The best known CS are called ZCS (Zeroth level Classifier System), that has been developed by Wilson (1994). This CS does not have a message board. Sensor and motor messages are directly linked to the condition and action parts of the classifier list. Wilson also designed XCS (Wilson 1995). Here, the fitness is split in two values, its *strength* - that evaluates the efficiency of the classifier - and its *quality* – that assesses the precision of the strength's evaluation. A classifier's overall fitness depends on the latter value. ZCS and XCS have been tested with success on animats, which had to survive in a dynamic environment, like woods with different kind of trees, foods, traps and predators.
The ACS (Anticipatory Classifier System) of (Stolzmann et al. 2000) adds an anticipation part to each classifier. This part is a string describing what the sensors should detect in the environment *after* the activation of this classifier. The fitness of a classifier is based on its capacity to well anticipate the consequence of its action in the environment. In a maze, for example, an animat is able to learn that it will reach a dead-end after turning right at a particular location.
In any given CS, the possibility of creating new classifiers – by hand, or with genetic algorithms - clearly increases the matching process time and entails a risk of combinatory explosion. Barry (1996) accordingly suggested the use of hierarchical CSs, in order to reduce the search space. This has been done by Donnart (1996) within the framework of animat navigation. Basically, his architecture relied on three interconnected CSs, a first one responsible for reactive behaviour, the second one responsible for planning behaviour, and the third one being in charge of building a cognitive map of the environment.
The different CSs just described were used in markovian environments only - i.e., in environments where a given sensory input corresponds to a given environmental state. However, a MMORPG is definitely a non-markovian environment, especially when it is implemented as a Multi-Agent System. The corresponding worlds are indeed continuously changing, according to the numerous actions of the NPCs and players. Such changes may well not be detected by the primitive sensors of NPCs, nor by humans themselves.

CS IN MULTI-AGENT ENVIRONMENTS

When a CS is embedded within a Multi-Agent System, every CS is seen as an agent that tries to satisfy its own goals and shares the same environment with other CS agents. The agents can

communicate, in order to improve their performance.
On the one hand, some animats acquire information about other animats indirectly, i.e., through the environment. This is the case, for example, of the so-called ”El Farol bar problem”, in which an agent has to decide whether or not it will enter into the bar, on the basis of the frequency of consumer visits over the last weeks (Hercog and Fogarty 2001).
On the other hand, other animats communicate explicitly, i.e., by exchanging classifiers or rewards. For example, in OCS (Organisational Classifier System), several CSs cooperate to solve a collective task, the design of an electronic circuit (Takadama et al. 2000). Each OCS represents an electronic component. By exchanging good rules with the others OCSs, the agents can collectively decide how they should be arranged in a spatially optimal circuit. In another work that simulates soccer, the players have to decide at each time what to do, on the basis of both an individual fitness value and a collective reward, the latter being evaluated relatively to the efficiency of the whole team (Sanza et al. 2000).

Figure 2. A snapshot of Ryzom

MHiCS, A PROTOTYPE FOR AN ACTION SELECTION ARCHITECTURE OF NPC IN MMORPG

All the above-mentioned CSs were not especially dedicated to NPCs in MMORPG. This is why we are developing a specific architecture, inspired from previous works, in order to fit the different needs and constraints of these new games. It will be applied to Ryzom, a MMORPG developed since 2000 by ©Nevrax (Figure 2).
Ryzom is a MMORPG elaborated with NeL (Nevrax Library), a free software library developed under General Public License. Like others MMORPG, Ryzom is a game playable only through Internet, in which players incarnate a character in a huge virtual world. The game is persistent and will be shared by thousands of players simultaneously. The player’s goals may be concrete - like exploring the world, killing monsters, searching for food - or more abstract - like increasing his competencies, being member of a community, becoming famous, etc. The NPCs will be merchants and craftsmen, making and selling artefacts, they will be people animating towns, wild animals living in forests and deserts, tribes and monsters that provide challenge to players, etc. They will manage multiple goals that may be conflicting, like sleeping, eating, hunting, protecting territory, finding resources and the like. Finally, they must be endowed with an appropriate action selection system, able to manage different goals in a massively multi-agent environment.

MHiCS is a *M*odular and *Hi*erarchical *CS* architecture dedicated to these NPCs (Figure 3). The modularity of the architecture will allow the design of various kinds of NPCs, in which modules could be assembling in different ways. These modules will correspond to different CSs, distributed in two hierarchical levels. At level I, several CSs will manage the motivations of the NPC. At level II, other CSs will refine the action commands of level I. Various motivations in the system may have some of these CSs in common. Two lower levels (III and IV) do not include any CS, but concern the execution of the final action.

The *Motivation* level

Each NPC owns different motivations - like self-protection, hunger, flocking. Each motivation is associated with a specific CS that is not shared by other motivations - but different specific CSs can have similar action commands.
A motivation is associated with two values, its Relative Power (RP) and its Motivation Value (MV). Through the RP values, the programmer can attribute a ‘personality’ to the NPCs. For instance, if a given individual has a hunger RP of 4, while another has a hunger RP of 10, the latter will have, during its whole life, a stronger tendency to practice all the actions linked to hunger than the former one. MV is a value between 0 and 1, giving the current strength of the motivation. If a NPC is eating, its hunger MV decreases to 0, otherwise it increases to 1.
Several CS belonging to motivation of level I can be triggered at the same time. Their activation values depend on their RP and MV values. Several action commands belonging to different CS can then be selected. These action commands will trigger the CS of level II.

The *Common CS* level

Each CS of level II can be activated by more than one motivation of level I. If a CS is selected by a

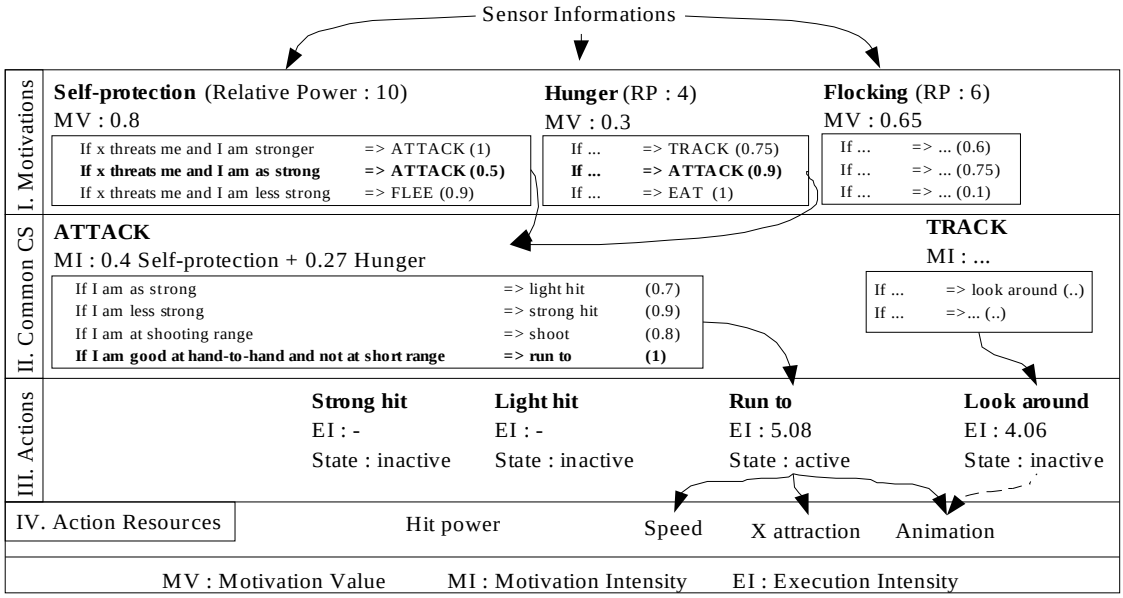

Figure 3. An illustration of MHiCS (see text for explanation). For the sake of clarity, the CS are only depicted by their classifier list, and the condition and action parts of the classifiers are not translated in strings of {0,1,#}.

single motivation only, it inherits a Motivational Intensity (MI) value, which is function of both the MV of the motivation and the fitness value of the level I classifier which has triggered the CS of level II. If a CS (e.g. on Fig.3, Attack) is activated by several motivations, its MI value depends on the MV of all the motivations (e.g. on Fig. 3, Self-protection 0.8 and Hunger 0.3), and on the fitness values of all the level I classifiers that have triggered this CS (on Fig. 3, bold classifiers at level I: 0.5 and 0.9). A classifier that is activated by several motivations will have more chances to be triggered than a classifier activated by a single motivation.

The *Action* level

All motivations diffuse their MI in the CS involved at level II. As a consequence, several classifiers can be selected at the same time, and several action commands can be executable at level III. Each action command is associated with an Execution Intensity (EI), depending on the fitness value of the corresponding classifier, the MI of the corresponding CS(s), and the RP of the corresponding motivation(s). For example (see Fig.3), the action command 'Run to', ordered by a classifier with a fitness value of 1 (bold classifier at level II), belonging to a CS with a MI of 0.4 (through Self-protection, RP=10) + 0.27 (through Hunger, RP=4), will have an EI of 1[(0.4*10)+(0.27*4)] = 5.08.

The *Action Resources* level

Level IV provides resources for action execution. In particular, it supplies resources for behavioural animations (for eating, running, etc.), and for the management of motion speed, attraction forces from X, repulsion forces from Y, etc.

The action command with the highest EI value has the primacy to recruit the needed resources. Other executable actions cannot require these already-used resources, but have only access to free ones. The behaviour that will be displayed by the NPC in the environment will be a combination of all the activated resources.

Evaluation and creation of CSs

The CS fitness values are computed on line and depend on the executed actions. If these actions satisfy the motivations that have triggered them at level I, all the classifiers that were implied in the action execution, at whatever level, will have their fitness value increased. In the opposite case, their fitness value will be decreased. If there are no classifiers matching to a particular environmental context, new ones will be discovered off line by a genetic algorithm.

The MAS environment

Each NPC equipped with MHiCS will be considered as an agent in a MAS environment. It will be able to communicate with other NPCs, for example to indicate the value of its internal variables (MV, MI, EI), in order to influence the motivations or the EI of other agents. It will also be able to exchange efficient classifiers or modules with other NPCs, in order to increase its learning process or its intrinsic skills.

Communication will be also possible between NPCs and players. On the one hand, players could train NPCs to achieve a given task, through the reinforcement of some classifiers. On the other

hand, through the players' actions, NPC could learn to detect the players' motivations or personalities, and decide to cooperate or to compete with them.

A preliminary test of MHiCS

Such a complex architecture must be tested step by step, in order to check the operational efficiency of each mechanism.

The first step – the only one already done –checked the diffusion of the motivations through a small number of CSs, in a simplified environment having the same characteristics as Ryzom. The corresponding experiments involved the simulation of prey, predators and 'preydators' – which behave either as predators or prey – in a closed environment. Each MHiCS included 2 motivations at level I, 4 CSs at level II, 4 actions at level III. Level IV was not implemented, the actions being simulated directly with their resources (see Robert 2002, for the detailed results).

In such conditions, we observed how easy it was to attribute a personality to NPCs thanks to RP values. Actually, significant differences in the duration of the displacements were exhibited by our three kinds of NPCs, characterized by different Exploration RP values. More importantly, we observed that the diffusion of the motivations entailed a correct chaining of actions for all NPCs. It also turned out that bad parameter fitting could induce unwanted effects, like dithering, i.e., a rapid oscillation between two actions. This issue - a classical one in action selection – could easily be solved at level IV, by locking by hand undesirable motions. But, for the design of autonomous NPCs, an adaptive solution has to be designed.

Such issues are being tackled in the second series of check tests that are under current development. Additionally, learning and evolutionary processes are implemented in the same experimental conditions as above. Future extensions will concern several NPCs in a Multi-Agent system, with the implementation of interaction mechanisms between NPCs and real players.

CONCLUSION

In this paper, we argue that classifier systems are particularly appropriate to be used as action selection architectures for autonomous NPCs. They are written in a classical video-game formalism and they integer adaptive capacities that allow NPCs to behave without human intervention. CSs have provided many sophisticated cognitive abilities in animats, like generalisation, specialisation, latent learning or planning (Lanzi 1999; Gérard 2002). To our knowledge, only a single video game – a classical one – currently integrates such a model (Conflict Zone, ©Masa). The aspiration of MHiCS is to demonstrate its relevance for more promising kind of games, the MMORPG.

REFERENCES

Barry A. 1996. "Hierarchy Formation within Classifier Systems A Review". In *Proceedings of the First International Conference on Evolutionary Algorithms and their Application EVCA'96*, E. G. Goodman, V. L. Uskov, and W. F. Punch (Eds.), 195-211.

Donnart, J. Y. and J. A. Meyer. 1996. "Hierarchical-map building and self-positioning with MonaLysa". *Adaptive Behavior*, No.5(1), 29-74.

Gérard, P. 2002. "YACS : a new Learning Classifier System using Anticipation". *Soft Computing*, No.6(3-4), 216-228.

Guillot A. and J.A. Meyer. 2000. "From SAB94 to SAB2000 : What's new, animat ?". In *From Animals to Animats 6*, J. A. Meyer, A. Berthoz, D. Floreano, H. Roitblat, and S. W. Wilson (Eds.), 3-12.

Hercog, L. M. and T. C. Fogarty. 2001. "Social Simulation Using a Multi-agent Model Based on Classifier Systems: The Emergence of Vacillating Behaviour in the ``El Farol" Bar Problem". *Computer Science*, No.2321, 88-114.

Holland, J. H. 1975. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor.

Holland, J. H. 1986. "Escaping brittleness: the possibilities of general purpose algorithms applied to parallel rule-based systems". *Machine Learning Journal*, No.2, 593-623.

Kovacs, T. 2002. "Learning Classifier Systems Resources". *Journal of Soft Computing*, No.6(3-4), 240-243.

Lanzi, L. 1999. "An Analysis of Generalization in the XCS Classifier System". *Evolutionary Computation*, No.7(2), 125-149.

Mac Namee, B. and P. Cunningham.2001. "A Proposal for an Agent Architecture for Proactive Persistent Non Player Characters". Department. Technical Report, TCD-CS-2001-20, Trinity College, Dublin.

Robert, G.2002. "Contribution des méthodologies animat et multi-agent à l'élaboration des jeux en ligne, persistants et massivement multi-joueurs.". http://animatlab.lip6.fr/Robert/index_fr.html

Sanza C. ; C. Panatier ; and Y. Duthen. 2000. "Communication and Interaction with Learning Agents in Virtual Soccer". In *Proceedings of Virtual Worlds 2000*, J.-C. Heudin (Ed.), 147-158.

Stolzmann W. ; M. Butz, V ; J. Hoffmann ; and D.E. Goldberg. 2000. "First Cognitive Capabilities in the Anticipatory Classifier System". In *From Animals to Animats 6*, J. A. Meyer, A. Berthoz, D. Floreano, H. Roitblat, and S. W. Wilson (Eds.), 287-296.

Takadama K. ; T. Terano ; and K. Shimohara. 2000. "Learning Classifier Systems Meet Multiagent Environments". In *Third International Workshop on Learning Classifier Systems (IWLCS-2000)*, L. Lanzi, W. Stolzmann, and S. W. Wilson (Eds.), 192-210.

Wilson, S. W. 1994. "ZCS: A Zeroth Level Classifier System". *Evolutionary Computation*, No.2(1), 1-18.

Wilson, S. W. 1995. "Classifier Fitness Based on Accuracy". *Evolutionary Computation*, No.3(2), 149-175.