

MHICS, A MOTIVATIONAL ARCHITECTURE OF ACTION SELECTION FOR NON-PLAYER CHARACTERS IN DYNAMIC ENVIRONMENTS (DRAFT)

Gabriel ROBERT and Agnès GUILLOT
Université Pierre et Marie Curie - Paris 6, UMR7606,
AnimatLab - LIP6, CNRS, UMR7606,
8 rue du Capitaine Scott
75015 Paris, France
E-mail : gabriel.robert@pamdev.com
agnes.guillot@lip6.fr (corresponding author)

KEYWORDS

Action selection; Animat approach; Learning Classifiers Systems; Non-Player Characters.

ABSTRACT

In the wake of today's drive for increasingly realistic video game environments, the credibility of non-player characters (NPC) behavior has become an important issue. This paper describes MHiCS (*Motivational and Hierarchical with Classifier Systems*), a control architecture for action selection that can be embedded in such characters. It implements mechanisms stemming from the animat approach, which aims at conceiving autonomous and adaptable entities able to survive in unpredictable environments. The originality of MHiCS lies in its combination of a motivation mechanism with intelligible condition-action rules (Classifiers Systems) and online unsupervised learning processes. It has been successfully tested in the Half Life game, against efficient hand-tuned NPC. This work contributes to the design of Classifiers Systems operating in non-deterministic environments and to the improvement of decisional autonomy in non-player characters.

INTRODUCTION

MHiCS – acronym for *Motivational and Hierarchical with Classifier Systems* - is a control architecture for non-player characters (NPC) in video games that enables the NPC to autonomously select their actions, even in situations not entirely expected by the game developers. Such situations are becoming frequent with the growing complexity of modern video games and the dynamics of multiple human players interaction. Following the classical '80/20 Pareto law' (Pareto 1896), 80% of the game developers' time is dedicated to writing behavioral rules to avoid such unpredictable situations, even though they only occur in 20% of the game.

Having to cope with unexpected events, NPC can be assimilated to creatures conceived in the 'animat approach' (or behavior-based approach), a branch of AI that focuses on the synthesis of adaptive *animat* (artificial animals) and draws as much of its inspiration as possible from biology and ethology (Guillot and Meyer 2001). Both animats and NPC must select what actions to perform to best satisfy their goals at any time, and they have to adapt online to highly non-deterministic environments, different goals and unpredictable actions from other players (whether human or artificial).

Adaptive control architectures built in the animat approach are mostly subtended by various models and techniques like neural networks, fuzzy models, potential fields, evolutionary methods (see Humphrys 1996; Pirjanian 1997; Prescott et al. 2001, for reviews). Though suitable for endowing adaptive capacities in animats, they are not necessarily adapted to the symbolic computations required by NPC. However, the *learning Classifier Systems' model* (CS, Holland 1986), gathering both transparent rules and efficient adaptive processes, may bridge that gap.

Classifier Systems

A CS contains a *classifier list*, a pool of 'condition-action' rules called *classifiers*. This list is generally initialized randomly. Each classifier is made of three parts: the *condition* part corresponds to the environmental information received by the animat sensors; the second part is the *action* to be performed in such a case; the third part is a '*strength*' value, a quantitative measure of the classifier's past successes and failures. Both the condition and

action parts are expressed as strings defined by low- or high-level symbols. As illustrated in Figure 1, when an animat detects some environmental features (1), it encodes this information into a string of symbols (here zeros, ones, and # as special “don’t care” symbols) that it deposits on the message board, together with other possibly internal messages (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 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 behavior corresponding to the activated effectors is performed in the environment (6).

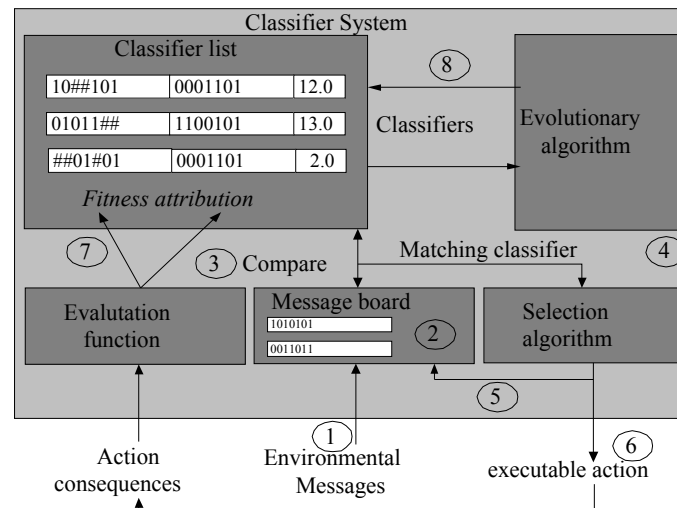


Figure 1. A Classifier System (see text for explanation).

Classifier systems use two different adaptive mechanisms: reinforcement learning and evolutionary process. The learning process tunes existing classifiers. Each time an action command is executed, the fitness value of the corresponding classifier is incremented or decremented depending on its outcome (7). If the perception matches several classifiers condition part at step (3), the classifier with the highest fitness value has a greater probability of being activated. This reinforcement learning process - well-known in ethology - allows for efficient association of classifiers to given tasks. The evolutionary process (or other similar techniques) creates new classifiers specifically relevant to the environmental context.

The literature on using CS to solve action selection problems in game situations is sparse. Example of their application include: anticipating the behavior of human opponents in Rock-Scissor-Paper games (Meyer and Ganascia 1996), generating efficient behavioral sequences of spaceships’ fights in the persistent universe of Mankind II (Girard et al., 2002) and simulating soccer (Sanza et al. 2000) or basket-ball players (Heguy et al. 2001).

Yet the online adaptive processes of CS running in game environments may generate a huge space of research, too difficult for being managed by a single CS. To break down this complexity, we choose, for the architecture proposed in this paper, to allocate parallel independent CS to several NPC’s ‘motivations’ serving to satisfy the various goals of the game. The research space each CS will have to explore is thus reduced to the associated motivation and the final behavioral path performed by the NPC will result from a compromise between the best actions proposed by each CS.

In the first following section, we will describe the main characteristics of MHiCS. In a second section, we will compare teams of NPC implemented with MHiCS with teams of hand-tuned NPC in a classical video game. We will then discuss the results on the basis of their generalization for other game situations.

MHiCS, AN ACTION SELECTION ARCHITECTURE FOR BOTS

Experimental platform

MHiCS will be applied to the experimental platform *Capture The Flag*, scenario of *Team Fortress Classic*, a well-known modification of the *first-person shooter* game *Half-Life* (Valve ©1999). In the scenario, two teams of players are confronted. The goal of each team is to fetch the flag from the enemy base and bring it back to

their own. When a player is killed, it re-appears instantly in its base, if it was carrying a flag, it is dropped. The main difficulty for action selection is the conflicts between the collective goal and the two personal goals. The collective goal is to increase the *team score* (+10 points for each enemy flags returned by the team); the personal goals are the *individual score* (+10 pts for each enemy flags returned by the bot, +1 pt for each opponent killed, -1 pt for each teammate killed), and the *survival score* (1 pt each time the bot is killed). Using this scenario is particularly convenient for the possibility to confront our bots implemented with MHiCS (or MHiCSbots) to different bots already developed, in particular HPBbots (v3.0) - which are equivalent to novice players - and Foxbots (v0.698) - equivalent to expert players.

MHiCS general structure

MHiCS has a hierarchical structure dispatched on four levels (Figure 2; Robert and Guillot 2003). At level I, the different motivations values vary according to the satisfaction or non-satisfaction of their respective goals. At level II, different Classifier Systems include sets of condition-action rules specific to each motivation. Levels III and IV respectively contain the possible actions and their associated resources allowed by the bot's effectors. At the initial step, the perceptions of the bot's sensors are matched against the condition part of the classifiers; this selects potential actions that could satisfy the motivations. Reciprocally, actions executed at the final step modify the motivation value and the external environment - and, consequently, the bot's perceptions.

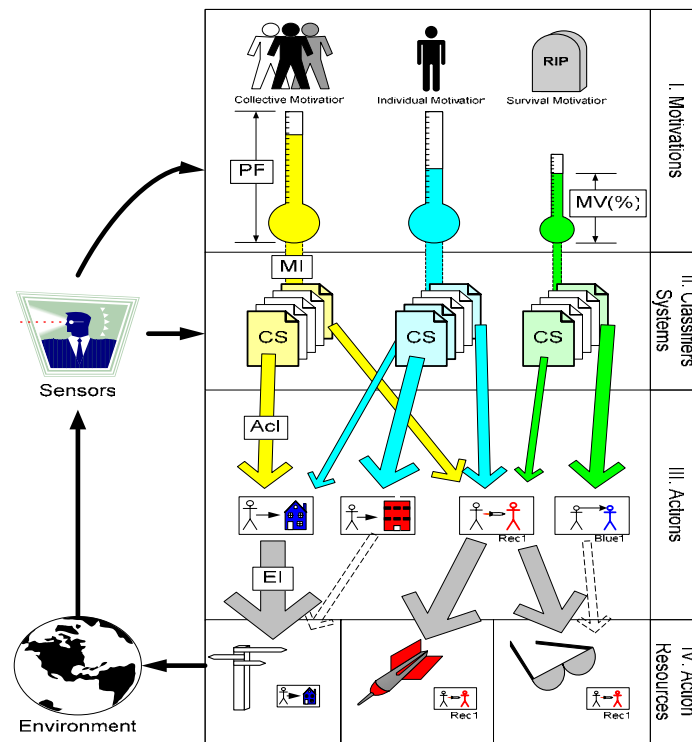


Figure 2. MHiCS general architecture (see text for explanation).

We will describe first the action selection mechanisms of MHiCS, then its adaptive processes – i.e., rule learning and rule discovery.

MHiCS action selection process

Inspired from Blumberg (1994), the decisional process of MHiCS results from a hierarchical diffusion of the motivation values of level I through the three other levels of the architecture. The selection mechanism is based on a “free-flow hierarchy” principle –as opposed to a “winner-takes-all”– allowing the parallel eligibility and execution of various elements at each level of the hierarchy. In animats, this principle has been demonstrated as the most efficient for action selection in complex environments (Tyrrell 1993).

Level I: The Motivations

This level defines the goals of the bot and their respective priorities. It takes the current scores as inputs, and outputs one *Motivational Intensity (MI)* for each motivation.

Ascribing motivations to a bot is easy in this context: they can be directly associated to the maximization of the various scores. Here, the first motivation (*Collective Motivation*) will be linked to the team score, the second (*Individual Motivation*) to the individual score, and the third (*Survival Motivation*) to the survival score. Note that these motivations are highly conflicting: *Collective* and *Individual Motivations* both incite offensive behaviors, while *Survival Motivation* incites defensive ones. *Collective Motivation* asks for cooperation among the bots, *Individual Motivation* for competition.

Motivational Intensities are products of two factors: a *Personality Factor (PF)* parameter and a *Motivation Value (MV)* variable which decreases or increases whether a motivation is – respectively - satisfied or not:

$$MI_{\text{Motivation}} = PF_{\text{Motivation}} \times MV_{\text{Motivation}}$$

Through the *Personality Factor* parameter, the developer can attribute a ‘personality’ to the bot. For example, if the *Collective Motivation* PF is greater than the *Individual Motivation* PF, the bot will end up being altruist (whereas the reverse would yield an egoist). The *Motivation Value* is an inverse-tangent normalized variable, which decreases when the motivation is satisfied. In persistent games such as TFC, the satisfaction of the motivation cannot be associated to absolute scores (e.g. returning 100 flags or killing 50 opponents). This is why we choose as a criterion the *maximization of the score difference between the competing teams*. Consequently, the *MV* for the *Collective Motivation* takes into account the number of flags returned by each team; for the *Individual* and *Survival Motivation*, the individual score of the bot is compared to the best score in the opposing team. For example, the *Motivation Value* for the *Collective Motivation (CM)* is computed as follows:

$$MV_{CM} = \left(\left(\frac{\text{atan}\left(\frac{\text{OpScore} - \text{Tscore}}{\text{Tscore}}\right) + 1}{\pi/2} \right) / 2 \right), \text{ where OpScore is the opponent score and Tscore the team score.}$$

Each *MI* value is transmitted to the corresponding specific CS on level II.

Level II: The Specific Classifier Systems

This level selects the eligible classifiers that can finally compete for action selection, depending on the current bot perceptions and the *Motivational Intensity* inputs from Level I. This level outputs the *Activation Intensities (AcI)* associated to the actions of each eligible classifier.

Each motivation is associated with a specific Classifier System. As seen above, this was done to reduce the research space for each CS while maintaining modularity: the addition or removal of a motivation doesn’t affect the set of rules already established for the other motivations. As a side effect, it is possible that duplicate rules be present in different CS, since each specific CS has its own set of rules.

The general format of the CS is inspired from the ZCS (Wilson 1994) and each classifier is made of a condition part, an action part and a priority part. The format of the condition and action parts has a heterogeneous nature (coded as Boolean or non Boolean symbols). The perceptions for the condition part describe – in a high-level way - either an environment or an internal state. Some of them also accept an argument (e.g., a ‘target’) when the perception concerns another bot or player, and then can code for this target’s state or for bot-target relationships. The actions of the classifiers are also coded as high-level symbols and some accept an argument. They control the destination of movement –be it static or dynamic, the direction of gaze and the gun trigger.

Only the classifiers whose condition part matches with the current perceptions are eligible. In the priority part of each classifier is associated a *strength* used for the selection between classifiers simultaneously eligible. Its value is initialized at 1.00E-05 and will be updated in the range [-1; 1] during learning, depending on its ability to reduce the *MI* of the corresponding motivation when it is activated.

The *Activation Intensities (AcI)*, associated to each action, are computed as follows:

$$AcI_{M,Act} = MI_M \times \max_{C \in \text{eligible_cls}} [Strength_{M,Act,C}], \text{ where M is the current motivation, Act the action and eligible_cls}$$

the eligible classifiers for the motivation.

These variables are proportional to the specific *MI* and the maximum strength obtained by the eligible classifiers having the same action part in the same CS. The *AcI* will determine the relevance of eligible actions for the satisfaction of the motivations. The action selection proper occurs at levels III and IV.

Level III: The Actions

This level selects the best actions to perform, depending on the *Activation Intensity* inputs from Level II, which represent how much each action is expected to help satisfy the associated motivation. The outputs of level III are the *Executable Intensity* of the actions.

Just as several classifiers – appertaining to the same or to different CS - can be eligible at the same time at level II, several action commands can be eligible at level III. The best actions to execute are the ones that correspond to the best *compromise* of the *AcI* associated to all motivations. In the space of points associated to all *AcI*

(Figure 3, top), the so-called “Pareto front” corresponds to the boundary of non-dominated points. A point is non-dominated by the other points when it is better on one criterion, and not worse on the other criteria. In this Figure, where only two criteria ($AcI1$ and $AcI2$) are shown for the sake of clarity, the “zenith point” is the theoretical point that would correspond to the ideal combination of all AcI . It is situated at the top of the smallest square including all the points of the Pareto front. Ideal AcI^* for each motivation is the maximum value of all AcI for that motivation:

$$AcI^*_M = MI_M \times \max_{\substack{M \in Motivations \\ C \in eligible_cls}} [Strength_{M,C}],$$

where max allows the determination of the highest strength among all eligible classifiers of all motivations.

The actual point of best compromise is selected by a multicriteria optimization algorithm (Tchebycheff distance method; Deb 2001). It is the one with the *smallest* Tchebycheff distance, the one situated on the Pareto front that creates the smallest square with the zenith point.

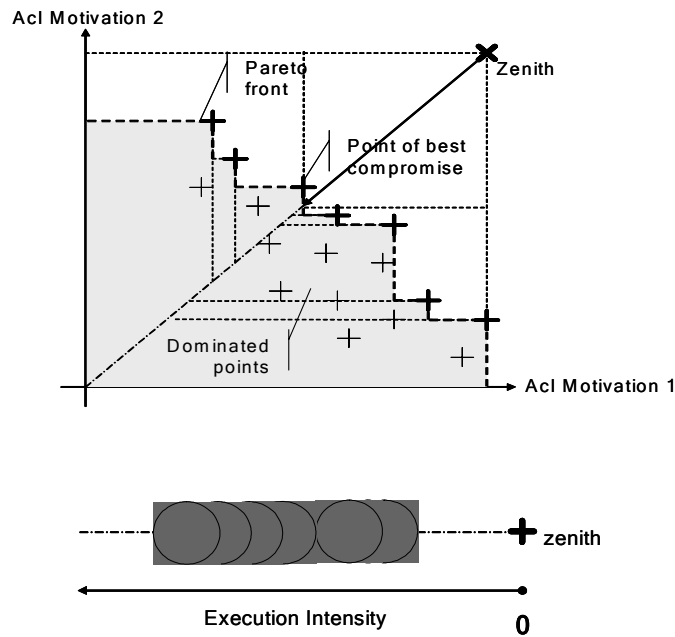


Figure 3. Top: The “Pareto front” corresponds to the boundary of the space of non-dominated points. The “zenith” point is the one situated at the top of the smallest square including all the points. **Bottom:** Executable Intensities (EI) are computed on the basis of the Tchebycheff distance of each point with respect to the zenith. Noise is added to these values, actions with the smallest EI become executable.

An *Executable Intensity* (EI) is computed on the basis of the Tchebycheff distance to the zenith point AcI^* (Figure 3, bottom). To avoid the problem of local minima, a Gaussian noise is added around the points, to allow the execution of different actions with close Tchebycheff values. The noise is calculated from the distribution and the standard deviation of the EI .

The noisy EI are computed as follows:

$$EI_{Act} = \max_{M \in Motivations} [AcI^*_M - AcI_{M,Act}] + Noise_{Act}$$

Eligible actions having the smaller EI become executable and can recruit resources at level IV.

Level IV: The Action Resources

This level allocates the action resources based on the *Executable Intensity* inputs from level III, which represent how close each action is to the ideal compromise. It outputs the actual actions to be performed in parallel by the bot.

The action with the lowest *Executable Intensity* has the primacy to use the required resources. The other executable actions can then no longer require these already-used resources. The final behavior displayed by the bot in the environment will result from the parallel execution of several actions recruiting different resources.

MHiCS adaptive processes

In classical AI, the next best action to be performed would be evaluated by ply search using heuristics. In video games, good heuristics are not easy to find because the consequences of any single action are many and hard to predict. There is no heavy heuristics or ply search in MhiCS: it just selects the best classifiers, according to their *EI* that are updated online at each time step. Two adaptive processes help improve this selection over time.

Rule learning

This first adaptive process updates the strength of each classifier eligible at level II, according to its implication in the decrease of the *Motivational Value MV*. In classical CS, in which actions have fixed duration and are sequentially displayed, this update occurs at the end of each classifier activation. In this game, the end of activation may be due either to the actual deactivation of the classifier (e.g., reach the team base), or to its temporary interruption by some short other action (e.g., stop for shooting while going to the team base). Moreover, actions with very long and unknown duration (e.g., going to the flag room) have to be evaluated from time to time before their deactivation. The solution we choose is to estimate the time after which each classifier is to be evaluated (*Eval*). This is done online by using the Widrow-Hoff learning rule (1960), which is a learning process that dynamically updates *Eval* for each classifier:

$Eval_X = Eval_{X-1} + \alpha(actual_duration - Eval_{X-1})$, where *actual_duration* is the actual duration of activation of the classifier, and α the learning rate (here fixed at 0.25).

At the end of this *Eval* time, a new strength of the classifier is computed on the basis of the *MV slope between two evaluations*. Indeed this particular criterion is the only one suitable for a persistent game where there is no final goal: a given classifier will be rewarded *according to its participation with others* to the satisfaction of the motivation and will be penalized otherwise, whatever the *MV* slope during its own activation – as illustrated in Figure 4.

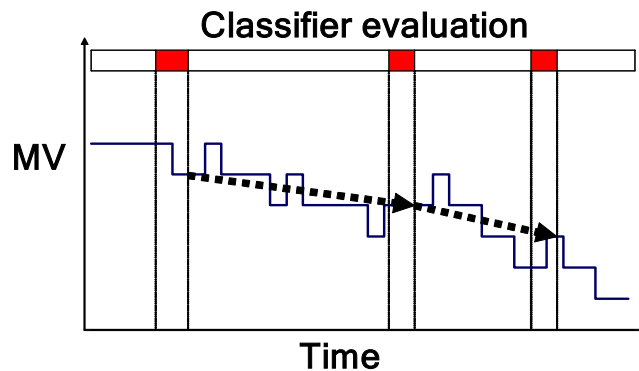


Figure 4. Illustration of the capacity of a classifier to participate to the *MV* decrease: its strength will be calculated according to the *MV* slope between two evaluations, not to the *MV* slope during its own activation.

This new strength is computed as follows:

$$New_Strgth_X = \frac{MV_{X-1} - MV_X}{T_X - T_{X-1}},$$

where *Strgth* is Strength, T_X and T_{X-1} are, respectively, the actual and the

previous times at which the classifier is evaluated. A positive strength corresponds to a *MV* diminution.

The previous strength is updated with the Widrow-Hoff rule :

$$Strgth_X = Strgth_{X-1} + \alpha(New_Strgth_X - Strgth_{X-1}),$$

where *Strgth* is Strength, and α fixed at 0.25.

Rule discovery

This second adaptive mechanism complements the previous one: rule discovery create new classifiers for unexpected situations, whereas rule learning tunes already existing classifiers. In CS, this step is generally dedicated to a genetic algorithm, the fitness function of which is based either on the classifier strength (Wilson 1994), or on their capacity to anticipate (Stolzmann et al. 2000). Even if this method has been successfully used offline (Ponsen 2004), it is far too slow for the online computation required to cope with the dynamics and

complexity of real-time video games (Laird and van Lent 2000). This is why we choose to implement mechanisms inspired from the “YACS” system of Gérard (2002), using heuristics explicitly directed by experience.

The first rule discovery mechanism is *specialization*. Whenever a classifier is evaluated - for example with a single current sensor value as condition associated to a given action -, all its potential specializations - with the addition of the other current sensor values as conditions - are recorded with the same strength. The next time this classifier is evaluated, all current sensors are again recorded, and a comparison is made between the strength of the previous possible specializations and the current ones. If one specialization obtains a “significantly higher” strength, a classifier is created with this specialization and strength. A strength is deemed “significantly higher” than another when it exceeds it by the interquartile distance computed from the strength of all possible specializations of the classifier.

The second rule discovery mechanism is *covering* (Wilson 1985). To ensure that there is at least one classifier for all the possible actions, new classifiers are created with an empty condition part when one CS does not associate an *Act* value to an action. The strength of these general classifiers is initiated with the minimum value (-1), but it will quickly increase through the specialization mechanism described above.

Deletion mechanism

New classifiers are created continuously by specialization, as the bot experiences specific settings. To avoid overpopulation and overspecialization, whenever the number of classifiers reaches an arbitrary threshold (200), half of them are deleted. However, we cannot rely on strength alone to determine which classifiers to suppress: because the final behavior is a compromise between all motivations, a classifier with a small strength in one CS could well be essential for the satisfaction of another motivation in another CS. Furthermore, general classifiers may be more relevant than specialized ones when the multiple conditions of the latter do not exactly match with the context.

A quota system was implemented that takes all these considerations into account. The classifiers kept are distributed as follows: 25% for the classifiers with the highest strengths, the remaining 75% divided evenly between all the actions. Then, for each action: 50% for classifiers with one condition, 25% with two conditions, 12,5% with three conditions, etc.

The action selection, rule learning and discovery processes of MHiCS previously described will now be confronted to hand-tuned bots in the CTF platform.

EXPERIMENTS AND RESULTS

The experiments test the ability of MHiCS to efficiently and adaptively select the actions of bots (called MHiCSbots) in the non-deterministic *Team Fortress Classic* environment. First, only one motivation will be attributed to MHiCSbots to demonstrate the simplicity of MHiCS’ design. Then, with three motivations, the capacity of MHiCS to cope with multiple conflicting goals will be illustrated.

Our experimental MHiCSbot team is composed of 4 players - not too few to preserve the dynamics of the game, not too many to facilitate future analysis. The opponents of the MHiCSbot team are composed of 4 HPBbots or 4 Foxbots. All bots are restricted to the role of soldiers, in order to make their behaviors homogeneous. All have similar perceptions, actions and resources (resp. of number 13, 9 and 3), but those of MHiCSbots lie in small separated tables, whereas those of the other bots are buried among respectively 20000 and 45000 lines of code. All bots have similar motivations: they have to maximize the 3 scores of the game, i.e., the team score, the individual score, and the survival score. Note that, since the team’s score is the most important for winning matches, the *Collective Motivation* is the main motivation to satisfy.

In all experiments (under the conditions of one motivation or three motivations; with or without adaptive processes; matches against HPBbots or Foxbots) the MHiCSbots are initialized with the same basic hand-designed Classifier System (see Appendix). All data are recorded during five 24 hours games –this duration allows the full course of learning and rule discovery to take place. The non-parametric Wilcoxon test for paired samples is used for scores comparison.

Experiment 1: One Motivation (*Collective Motivation*)

Because the *Collective Motivation* is the main motivation to satisfy in this game, we first experiment the influence of this single motivation on the action selection. As seen in above, this motivation decreases (i.e., is satisfied) when the difference between the team’s score and the opponent score increases – i.e., when MHiCSbots return more flags than the other team.

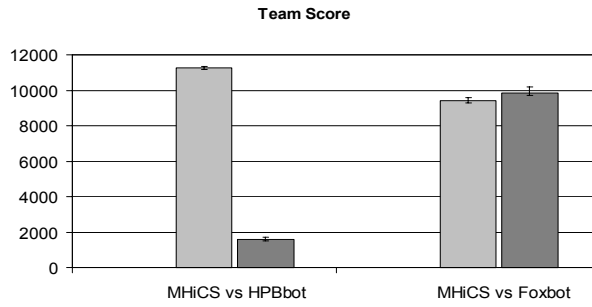


Figure 5. Median, min and max of teams' scores of MHiCSbots (without adaptive processes) versus HPBbots (W=15; p<0.05) and Foxbot team (W=15;p<0.05).

The Classifier System used by MHiCSbots in all these experiment is a very simple hand-designed one (Appendix A).

It contains two classifiers for situations involving the flag, two classifiers for situations involving other bots, and six classifiers with different actions to perform when another MHiCSbot is bringing the opponent flag. The nine classifiers are initialized with the same strength; the latter six – that share the same condition part - are therefore initially selected at random. Figure 5 shows that, without any adaptive processes, the MHiCSbot team significantly dominates the HPBbot team, but is also significantly dominated by the Foxbot team.

The addition of adaptive processes enhances the results against the Foxbot team (Figure 6, top): learning alone improves the scores but with non significant differences, whereas learning and rule discovery lead to significant winnings - as illustrated by the constant decrease of *Collective Motivation Intensity* during one match (Figure 6, bottom).

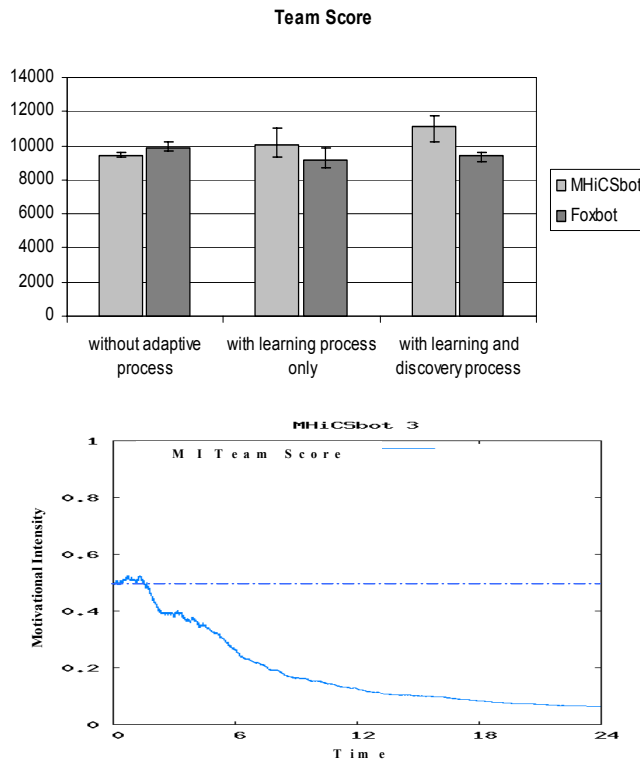


Figure 6. Top : Median, min and max Team scores of MHiCSbots versus Foxbots, without adaptive processes, with learning (W=13; p>0.05), with learning and discovery processes (W=15; p<0.05). Bottom: *Collective MI* during a 24-hr experiment. The dotted line corresponds to the level at which the scores of both teams are equal; below that line, the score of MHiCSbots is greater than that of Foxbots.

The dynamic adjustment of the time after which classifiers are evaluated (*Eval*) is efficient: classifiers 2 and 3 associated with looking at teammates and shooting opponents have a shorter *Eval* than the other involving

moving to different parts of the map (Figure 7, top). Figure 7 (bottom) illustrates that each classifier is no longer performed with similar frequency by each bot, as learning allows the emergence of *roles* among them. For example, MHiCSbot 1 and 4 concentrate on attack, focusing on catching and returning flags (classifiers 1 and 4), whereas MHiCSbot 2 and 3 concentrate on defense – looking at and shooting opponents, protecting the team base (classifiers 2, 3 and 7).

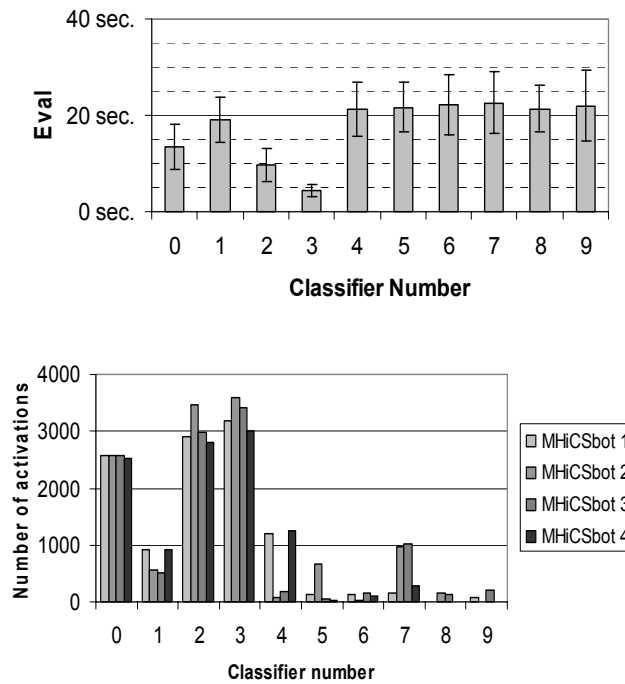


Figure 7. Top: Mean and standard deviation of Eval (time of evaluation) for each classifier; all Eval were initialized at 20 sec. Bottom: Use of the classifiers by each MHiCSbot (with learning process only).

About 5000 classifier specializations occur during one 24-hour match. The analysis of the classifiers kept at the final step (for the best MHiCSbot during the best experiment) shows that the specialization focuses around the choice of a target.

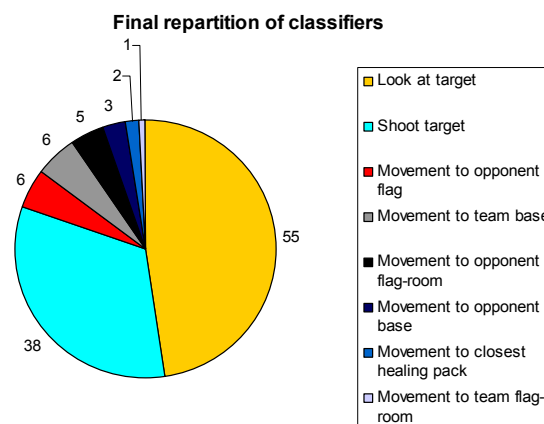


Figure 8. Final repartition of classifiers, sorted by their action parts. Actions are listed clockwise, beginning at noon.

This choice depends on contextual situations that can hardly all be anticipated by a developer (Figure 8) and though they are rarely used, they are essential for significantly beating the Foxbot team (Figure 6, top). Rule discovery also allowed the MHiCSbots to relax the too restrictive initial handcrafted CS. For example, classifier 3 “if the target is in the MHiCSbot team = False, then Shoot target” forces the bot to engage combat with all the enemies it encounters, though it may be dangerous in specific situations (e.g., when the MHiCSbot is wounded). A new classifier was created that allows the bot to “look at another MHiCSbot” in such conditions, thus avoiding dangerous fights.

We will now see how MHiCS deals with three motivations and show that is able to strike good compromise between conflicting goals. In all these experiments, MHiCSbots are only confronted to Foxbots.

Experiment 2: Three Motivations (*Collective, Individual and Survival Motivations*)

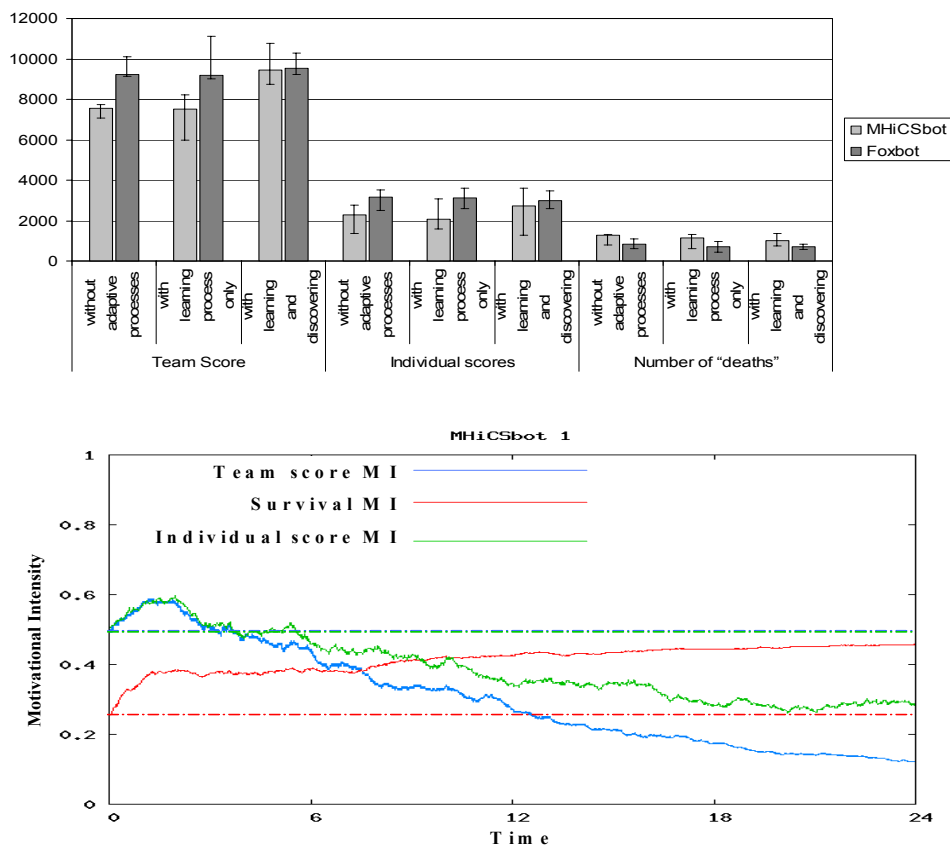


Figure 9. Top: Median, min and max scores (Team, Individual and Survival) of MHiCSbots vs Foxbots, without adaptive processes ($W=-15$; $p<0.05$), with learning process only ($W=-15$; $p<0.05$), with learning and discovery processes ($W=-1$; $p>0.05$). **Bottom:** *Collective, Individual and Survival MI* plotted during a 24-hr experiment for the best MHiCSbot in the best experiment (with learning and discovery process), compared to the best Foxbot. The dotted lines correspond, for the highest, to the level at which the Team and Individual scores of both bots are equal, for the lowest, to the level at which the Survival scores of both bots are equal. Scores of the MHiCSbot are greater below, lower above. Note that greater Survival scores (numbers of deaths) lead to lower satisfaction of the *Survival Motivation*.

The *Individual Motivation* decreases (i.e., is satisfied) when the individual score difference between the MHiCSbot and the best opponent bot increases. The associated CS is initialized with the same classifiers used for the *Collective Motivation* in the previous setting plus one that allows chasing opponents (Appendix B).

The *Survival Motivation* decreases when the difference in “deaths” between the MHiCSbot and the best opponent bot increases. The associated CS is initialized with three classifiers: one to look at other MHiCSbots, one to shoot enemy bots and the last to move towards the closest healing pack (Appendix C).

The *Personality Factors (PF)* attributed to *Collective, Individual* and *Survival Motivations* are respectively 1, 1 and 0.5, since the objective is to capture as many flag as possible and that goal is mainly fulfilled by the first two motivations.

The results shows that having to cope with three motivations, rather than just the collective one, reduces the overall performance of the MHiCSbots, although they still perform as well as the Foxbots for team and individual scores when they are equipped with both rule learning and discovery processes (Figure 9 top). This last result is also illustrated in Figure 9 (bottom) by the continual decreases of *Collective* and *Individual MV*. The *Survival Motivation* is less satisfied than the others, as it is initially associated to a lower *PF*.

With three motivations, the learning process does not lead to the emergence of roles among MHiCSbots, because no single role can satisfy these conflicting goals at the same time. Yet the learning process allows the online adjustment of all classifiers strength, values that not only reflect the relevance of the classifiers to the satisfaction of their specific motivation, but also their relevance - or irrelevance - to the satisfaction of the other motivations. This maximizes the performance of compromise behaviors.

As illustrated in Figure 10, the rule discovery process operates in parallel in each CS and automatically creates different distributions of classifiers adapted to each task. For example, the number of classifiers having “shoot target” in their action part is greater for the *Survival Motivation* CS than for the others. That is because, to survive, a bot must know precisely who and when to shoot. Of course, the percentage of classifiers associated to “movement to closest healing pack” is also much higher.

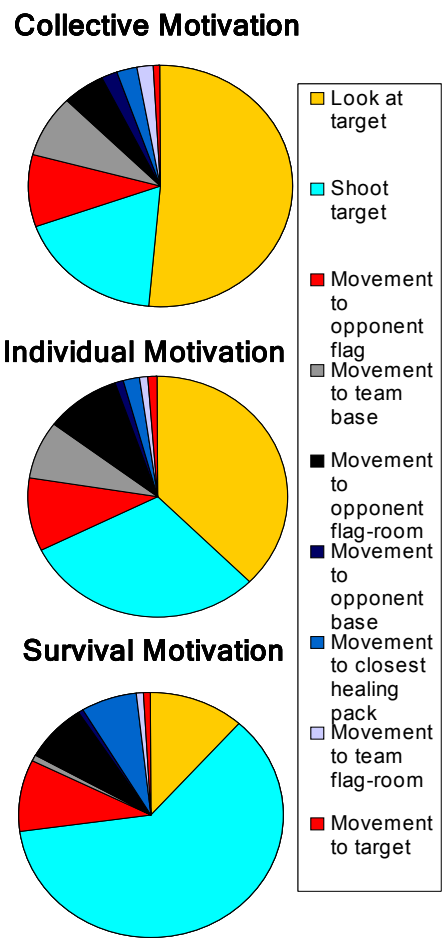


Figure 10. Final repartition of classifiers, sorted by their action parts, for the CS of *Collective Motivation, Individual Motivation* and *Survival Motivation*. Actions are listed clockwise, beginning at noon.

DISCUSSION

Contributions

To Non Player Characters design

The MHiCS architecture of action selection provides substantial contributions to the decisional autonomy of Non Player Characters in a highly dynamic video game environment.

In MHiCS, initial sets of rules can be simply designed: with their transparent production rules (“if condition then action”), CS can be initialized by a game developer with quite good start sets. MHiCS also easily allows the addition of new motivations, perceptions, actions or resources, as the decisional mechanisms are completely separated from the data basis. The diffusion of motivational needs within specific CS ensures that NPC select autonomously and in real time efficient behavioral sequences on the basis of their current perceptions. Moreover, using learning and discovery processes, MHiCS can adjust the initial set of rules in order to increase their efficiency (here, increasing scores in a *Team Fortress Classic* scenario). The comparison of MHiCSbots with hand-tuned bots (here, HPBbots or Foxbots) reveals the capability of these automatic mechanisms. These adaptive processes operate online – but can also work off-line if needed - and are particularly suitable for persistent games, as their fitness is based on the scores difference between the teams rather on particular finite values. Compromise between conflicting goals (here, *Collective, Individual and Survival Motivations*) can be achieved, on the one hand, by the online adjustment of the strength of all rules relatively to their relevance towards these goals and, on the other hand, by the creation of new rules suited to the many contextual situations faced by the bot, a part of their programming that would be omission-prone and time-consuming for a game developer.

Such an enrichment of the repertoire of conditions-actions improves the efficiency but also the “credibility” of NPC, as confirmed by some informal matches between MHiCSbots and human players, in which MHiCSbots were preferred to Foxbots – particularly because they were considered as more fun and less “violent” (Robert 2005).

To CS models

Besides its main contribution, which is to link motivational diffusions with classifiers selection, MHiCS presents several other improvements to CS in general.

Classical CS use fixed perception spaces, which require the attribution of specific sensors for each possible target of the NPC. MHiCS classifiers accept one argument corresponding to a particular target in their condition part, avoiding the artificial addition of multiple sensors and, consequently, reducing the computation time. MHiCS classifiers, unlike classical CS but similarly to the TCS of Hurst et al. (2002), can also manage “pseudo-continuous” actions – i.e. action of varying and unknown duration. The mode of evaluation of the classifiers was modified accordingly: since the time at which their strengths can be evaluated cannot be fixed, this time has to be dynamically updated by learning. Furthermore, we cannot apply the reward retro-propagation algorithms used in CS at large to update classifier strength (e.g., bucket-brigade; Holland 1986), because several actions are performed in parallel. We choose to take into account the slope of the motivational values between two evaluations of a classifier. Lastly, following the work of Gérard (2002), the strength of all possible specializations is estimated as they are experienced by the NPC.

Limitations

The questions about MHiCS limitations concern its generic capability: is MHiCS able to be applied to other NPC in other platforms?

In TFC, MHiCSbot dealt with high-level perceptions and actions, and this can well hide some action selection issues. For example: friends are already distinguished from foes, movements are automatically determined by waypoints with an A* algorithm, etc. If a lower level of control were required, the number of classifier –and therefore the amount of memory and computing power needed- would quickly increase. Some solutions exist in the animat approach to address the navigational issues through the use of force fields (Flacher and Sigaud 2004) or ant algorithms (Dorigo et al. 1996); but the use of continuous values in CS is still mostly an open problem.

The introduction of multiple motivations was seen to decrease the overall performance of the bots. This is mainly due to the prevalence of *Collective Motivation* in TFC, but further experiments would be needed to investigate how to avoid this, in particular by adapting the *Personality Factor (PF)* attributed to each motivation. In this version of MHiCS, bots learn continuously. But there are conditions or situations in which learning is questionable (Blumberg et al. 2001). When NPC have too few interactions with human players, or when NPC are not persistent, learning is a waste of time. Then adaptive processes have to be differentially attributed to

NPC, either for all the game duration, or temporarily during the game – depending on particular game circumstances.

The capacity of MHiCS to build the initial set of classifiers entirely from scratch was also examined. Experiments – in the details of which we cannot enter in this short paper – have shown that MHiCSbots initialized with empty CS were able to discover the main classifiers needed to win the game (Robert 2005). A prior training in a map devoid of other NPC or human players was however mandatory, to allow the learning of the essential instructions (catch flags and return them to the base) without the disturbance brought by dynamic opponents. However, even after this initial training, the performance of MHiCSbots was only similar to novice players, leading to the conclusion that the methodology for optimizing rule discovery from scratch has to be improved.

CONCLUSION AND PERSPECTIVE

We have shown that the Motivational and Hierarchical architecture with Classifier Systems MHiCS, with its online action selection mechanisms, rule learning and discovery processes, provides an efficient decisional autonomy for NPC in the highly non-deterministic environments of real-time video games like *Team Fortress Classic* (MOD of the game *Half-Life* Valve, ©1999). Some of the improvements discussed above are being implemented, the corresponding code is open source.

Our current perspective is to compare the performance of the current MHiCS architecture to other models written in various script languages. Scripting is becoming the prevailing system used for video-games NPC and it has some similarities with CS models, in particular transparent production rules. Most scripts offer no adaptive processes (e.g., Richard et al. 2001), but recently Spronck et al. (2004) designed “Dynamic Scripts” (DS) that are capable of online reinforcement learning and offline evolutionary algorithms (Ponser and Spronck 2004; Spronck 2005). DS were successfully applied to non-deterministic environments, like computer Role Playing games and Real-Time Strategy games. The comparison with MHiCS in a same platform would estimate the respective contributions of both models, which differ by their selection and adaptive mechanisms.

Acknowledgments

This research has been granted by Nevrax, the French Ministry of Research and the LIP6. We thank for useful discussions Fabien Flacher, Thomas Degrès, Thierry Gourdin, and Vincent Cuzin.

APPENDIX A. Specific CS for *Collective Motivation* (F=false; T=True)

No	Force	Conditions	Action
0	1.00E-05	Another Bot carries a flag = F	Displacement to opponent flag
1	1.00E-05	Bot carries a flag = T	Displacement to team basis
2	1.00E-05	The target is in the Bot team = T	Look at target
3	1.00E-05	The target is in the Bot team = F	Shoot target
4	1.00E-05	Bot carries a flag = F Another Bot carries a flag = T	Displacement to opponent flag-room
5	1.00E-05	Bot carries a flag = F Another Bot carries a flag = T	Displacement to opponent flag
6	1.00E-05	Bot carries a flag = F Another Bot carries a flag = T	Displacement to team basis
7	1.00E-05	Bot carries a flag = F Another Bot carries a flag = T	Displacement to team flag-room
8	1.00E-05	Bot carries a flag = F Another Bot carries a flag = T	Displacement to opponent basis
9	1.00E-05	Bot carries a flag = F Another Bot carries a flag = T	Displacement to closer medical care

APPENDIX B. 10th classifier added to the 9 of Appendix A in the specific CS for *Individual Motivation*.

10	1.00E-05	The target is in the Bot team = F	Displacement to target
----	----------	-----------------------------------	------------------------

APPENDIX C. Specific CS for *Survival Motivation* (the condition “!High” is “True” when all different values are verified)

N ^o	Force	Conditions	Action
0	1.00E-05	Health level of Bot = !High	Displacement to closer medical care
1	1.00E-05	The target is in the Bot team = T	Look at target
2	1.00E-05	The target is in the Bot team = F	Shoot target

REFERENCES

- Blumberg B. 1994. “Action-Selection in Hamsterdam: Lessons from Ethology”. In *From Animals to Animats 3: Proceedings of the 3rd international conference on the simulation of Adaptive behaviour*. The MIT Press, Cambridge MA, 108-117.
- Blumberg B., B. Tomlinson and M. Downie. 2001. “Multiple conceptions of character-based interactive installations”. In *Computer Graphics International 2001*, 5-11.
- Deb, K. 2001. *Multi-Objective optimization using evolutionary algorithms*. John Wiley & Sons, Inc. New York, NY, USA.
- Dorigo M.; V. Maniezzo; and A. Colomi. 1996. “The Ant System: Optimization by a colony of cooperating agents”. *IEEE Transactions on Systems, Man, and Cybernetics-Part B*, 29-41.
- Flacher F. and O. Sigaud. 2004. “BASC, a Bottom-up approach to automated design of spatial coordination”. In *From Animals to Animats 8: Proceedings of the Eighth International Conference on Simulation of Adaptive Behavior*, MIT Press, Cambridge MA, 435-444.
- Gérard P. 2002. “YACS : A new Learning Classifier System using anticipation”. *Soft Computing*, 6, 3-4: 216-228.
- Girard B., G. Robert and A. Guillot. 2002. “Jeu Vidéo et Intelligence Artificielle Située ». In *Cognito*, 22: 57-72.
- Guillot A. and J. A. Meyer. 2001. “The Animat contribution to cognitive systems research”. *Journal of Cognitive Systems Research*, 2: 157-165.
- Heguy, O.; A. Berro; and Y. Duthen. 2001. “Learning System for Cooperation in Virtual Environment”. *5th World Conference on Systemics, Cybernetics and Informatics SCI 2001, ISAS 2001*, Orlando ,USA,
- Holland J. H. 1986. “Escaping brittleness: the possibilities of general purpose algorithms applied to parallel rule-based systems”. *Machine Learning Journal*, 2: 593-623.
- Humphrys M. 1996. “Action Selection methods using Reinforcement Learning”. In *From Animals to Animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior*, The MIT Press, Cambridge MA., 135-144.
- Hurst J.; L. Bull; and C. Melhuish. 2002. “TCS Learning Classifier System controllers on real robots”. In *Parallel Problem Solving from Nature - PPSN VII*. Springer-Verlag Heidelberg, pp. 588-597, 2002.
- Laird, J. E. and M. van Lent. 2000. “Human-Level AI’s Killer Application: Computer Game AI”. In *Proceedings of AAAI 2000 Fall Symposium on Simulating Human Agents*, Technical Report FS-00-03. AAAI Press 2000, 80–87.
- Meyer C. and J. G. Ganascia. 1996. “Utilization of imitation and anticipation mechanisms to bootstrap an evolutive distributed artificial intelligence system”. In *Proceedings of ICMAS’96*.
- Pareto V. 1896. *Cours d’Economie Politique*. Rouge, Lausanne.
- Pirjanian P. 1997. “An overview of system architectures for action selection in mobile robotics”. Technical Report TR-97, Aalborg University, Denmark.
- Ponsen, M. 2004. “Improving Adaptive Game AI with Evolutionary Learning”. MSc Thesis, Delft University of Technology, The Netherlands.
- Ponsen, M. and P. Spronck. 2004. “Improving Adaptive Game AI with Evolutionary Learning ». *Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004)*, University of Wolverhampton, 389-396.
- Prescott, T. J.; K. Gurney; F. Montes Gonzalez; and P. Redgrave. 2001. “The evolution of action selection”. In *The whole iguana*. The MIT Press, Cambridge MA
- Richard, N.; P. Codognot and A. Grumbach (2001) "The InViWo toolkit: describing autonomous agents and avatars", In *Lecture Notes in Computer Science*, 2190: 195-209.
- Robert G. and A. Guillot. 2003. “MHiCS, A Modular and Hierarchical Classifier Systems architecture for bots”. In *GAME-ON 2003*, EUROSIS, Institute of Electrical Engineers, London, UK, 140-144.
- Robert, G. 2005. “MHiCS, une architecture de sélection de l’action Motivationnelle et Hiérarchique à Systèmes de Classeurs pour Personnages Non Joueurs adaptatifs ». Ph.D. thesis, Université Pierre et Marie Curie, Paris, France.
- Sanza C.; C. Panatier; and Y. Duthen. 2000. « Communication and interaction with learning agents in virtual soccer”. In *Proceedings of Virtual Worlds 2000*, Springer-Verlag, 147-158.
- Spronck, P. 2005. “Adaptive Game AI”. Ph.D. thesis. University of Maastricht, Maastricht University Press, Maastricht, The Netherlands.

- Spronck, P.; I. Sprinkhuizen-Kuyper; and E. Postma. 2004. "Online Adaptation of Game Opponent AI with Dynamic Scripting." In *International Journal of Intelligent Games and Simulation*, 3, 1: 45–53.
- Stolzmann W.; M. Butz; J. Hoffmann; and D.E. Goldberg. 2000. "First Cognitive Capabilities in the Anticipatory Classifier System". In *From Animals to Animats 6, Proceedings of the 6th International Conference on Simulation of Adaptive Behavior*. The MIT Press, Cambridge MA, 287-296.
- Tyrrell, T. 1993. "Computational Mechanisms for Action Selection". Ph.D. thesis, University of Edinburgh, Centre for Cognitive Science.
- Widrow B. and M.E. Hoff. 1960. "Adaptive switching circuits". In *IRE WESCON. Convention Record.*, New York, 96-104.
- Wilson, S.W. 1985. "Knowledge Growth in an Artificial Animal". *ICGA 1985*, 16-23.
- Wilson S. W. 1994. "ZCS: A Zeroth level Classifier System". *Evolutionary Computation*, 2, 1: 1-18.

BIOGRAPHY



Gabriel Robert is born with Starwars and the first video games. He prepared his PhD about MHiCS architecture with NevraX, a French MMORPG company, together with the AnimatLab of the Laboratoire d'Informatique de Paris 6 (LIP6). He currently works as an AI game developer in PAM Development, a French company making sport games on console.



Agnès Guillot is Associate Professor in Psychophysiology at the University of Paris-X. She graduated in Human and Animal Psychology, and holds PhDs in Psychophysiology and Biomathematics. Her main scientific interest is in action selection in natural and artificial systems.