MHICS, A MODULAR AND HIERARCHICAL CLASSIFIER SYSTEMS ARCHITECTURE FOR BOTS

Gabriel ROBERT and Agnès GUILLOT
AnimatLab, Laboratoire d'Informatique de Paris 6
8 rue du Capitaine Scott
75015 Paris
France
E-mail : (gabriel.robert; agnes.guillot)@lip6.fr

KEYWORDS

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 the adaptive action selection mechanisms of a Bot (a simulated player). After introducing MHiCS, our control architecture adapted to the specific constraints of multiplayer games, we will present the first results on a *Team Fortress Classic* scenario.

INTRODUCTION

A new Artificial Intelligence approach focuses on the synthesis of adaptive simulated animals or real robots (called *animats*), whose mechanisms are inspired from biology and ethology as much as possible (Guillot and Meyer 2000). An animat has both sensors – which provide information about its environment or internal state - and effectors – which allow it to change its environment. To be able to survive, it is endowed with a control architecture that connects its sensors to its effectors, such architecture being able to adapt to changing circumstances through unsupervised learning.

A Bot is used for simulating a human player in a multiplayer video game. With nearly the same information as human players, they must be able to select the appropriate actions to fulfil their goals. Bots must have a correct behaviour but not too perfect to let human players win.

Bots behaving in these games are similar to animats as these artificial players have to adapt on line to dynamically changing environments, to different goals and to unpredictable actions from the players.

The control architectures developed by the animat community are useful to give adaptive behaviours to a Bot. In particular, one kind of model - the so-called *Classifier System* (CS) which is a population of 'condition-action' rules called classifiers (Holland 1986) - is especially convenient to design architectures able to efficiently select which actions the Bot should perform. A CS can learn which classifier is better than another to achieve a given task. New rules can also be discovered through the creation of new classifiers thanks to an evolution process like a genetic algorithm.

Figure 1: Screenshot of a blue team scout in TFC

In this paper, we will describe the main characteristics of MHiCS, the architecture we designed on the basis of CS in order to cope with video game constraints (Robert et al. 2002). We will then present our first results on a *Capture The Flag* (CTF) scenario of *Team Fortress Classic* (TFC), a well known modification (MOD) of the *First Person Shooter* (FPS) game *Half-Life* (Valve, ©1999) (Figure 1).

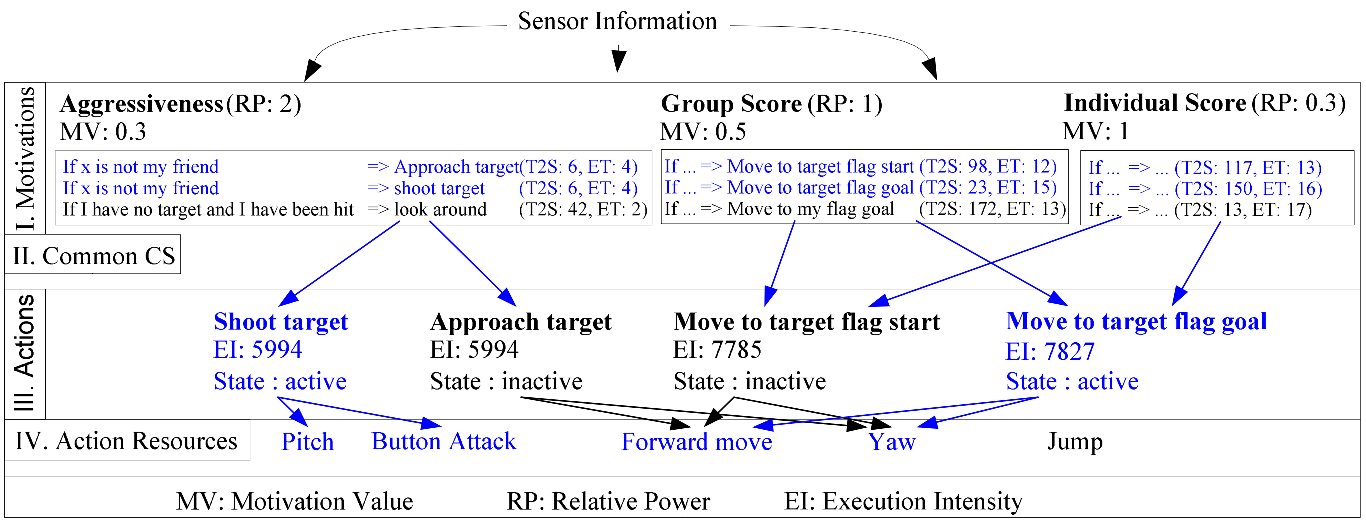
Sensor Information
I. Motivations
Aggressiveness (RP: 2)
MV: 0.3
If x is not my friend => Approach target (T2S: 6, ET: 4)
If x is not my friend => shoot target (T2S: 6, ET: 4)
If I have no target and I have been hit => look around (T2S: 42, ET: 2)
Group Score (RP: 1)
MV: 0.5
If ... => Move to target flag start (T2S: 98, ET: 12)
If ... => Move to target flag goal (T2S: 23, ET: 15)
If ... => Move to my flag goal (T2S: 172, ET: 13)
Individual Score (RP: 0.3)
MV: 1
If ... => ... (T2S: 117, ET: 13)
If ... => ... (T2S: 150, ET: 16)
If ... => ... (T2S: 13, ET: 17)
II. Common CS
III. Actions
Shoot target
EI: 5994
State : active
Approach target
EI: 5994
State : inactive
Move to target flag start
EI: 7785
State : inactive
Move to target flag goal
EI: 7827
State : active
IV. Action Resources
Pitch
Button Attack
Forward move
Yaw
Jump
MV: Motivation Value
RP: Relative Power
EI: Execution Intensity

Figure 2: An illustration of MHiCS showing how action selection is made across the levels

BOTS AND MULTIPLAYER GAMES

A FPS is a real-time 3D multiplayer game in which each player (human or Bot) has to move and fight to reach the goals of the game.

For a few years, different FPS engines have been used in research lab as they bring a complex and rich test environment: (e.g. Quake (Id software, ©1996) by (Laird and Duchi 2000), Unreal (Epic, ©1998) by (Calderon and Cavazza 2001), Half-Life (Valve, ©1999) by (Khoo and Zubek 2002) and many MOD of those engines (Quake II, Unreal Tournament, Counter Strike, etc.)). Different fields of AI are already involved in the design of a FPS Bot, e.g. for navigation, body animation, fighting tactic and goal as well as action selection (Tozour 2002).

In the CTF we specially apply in this paper, there are two teams of players. The goal is to take the opponent team's flag in its base and to bring it to the team base. In this scenario, the main difficulty for action selection is to reach at the same time the team goal (bring back the enemy flag), the personal goal (killing a maximum of opponents) and proper motivations set up by the Bot designer (e.g. aggressiveness). As each human player may have an unpredictable behaviour, it is a challenge for Bots to learn because their behaviours will not always have the same efficiency, and they must be able to dynamically re-evaluate their knowledge. MHiCS, the control architecture we will introduce now, aims to solve this particular issue.

MHICS, AN ACTION SELECTION ARCHITECTURE FOR BOTS

We have already described the details of a Classifier System in a previous paper, together with our architecture MHiCS - a Modular and Hierarchical CS architecture dedicated to virtual player for multiplayer games (Robert et al. 2002). We will here only sum up its main characteristics, illustrated on Figure 2.

The modularity of the architecture allows the design of various kinds of Bots, in which modules could be assembled in different ways. These modules correspond to different CSs, dispatched on two hierarchical levels. At level I, several CSs manage the Bot's motivations. 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. In our test, level II has been removed to simplify the first learning experiences.

The *Motivation* level (level I)

Each Bot has its own motivations – e.g. *Team Score, Individual score, Aggressiveness*. A motivation is associated with two values: *Relative Power* (RP) and *Motivation Value* (MV). Through the RP value, the designer can attribute a 'personality' to the Bot, for example by giving it high or low aggressiveness. The MV is a value between 0 and 1, which increases when the motivation is not satisfied, and decreases otherwise.

Each motivation is associated with a specific CS that is not shared by other motivations - but different specific CSs can have similar action commands. The goal of a CS is to satisfy the motivation that has triggered it, then to minimise its MV. Each rule (classifier) in a CS has a priority part used to choose between different classifiers simultaneously eligible. To accelerate the learning process, this priority is based on two values: *Time to Success* (T2S) and *Execution Time* (ET). T2S is the average time between the activation of a classifier (when it is selected and its action part active) and the next MV decrement. ET is the average time a classifier takes to be executed.

Several CS belonging to motivations of level I can be triggered at the same time. Their *Activation Values* (AV) depend on their RP and MV values (AV = RP*MV). As some actions (like "shoot to" or "approach target") need to be associated with a target (“opponent” or “flag”), each selected classifier is associated with a target.

Several action commands belonging to different CS can then be selected on different targets. These action commands can trigger the CS of level II (not in our actual implementation) or directly the actions of level III.

The *Action* level (level III)

As several classifiers can be selected at the same time, several action commands can be executable at level III. To select which action will be executed, they are sorted by priority before going through the level IV Action Resources. This priority is determined by the *Execution Intensity* (EI) value of each pair (action, target) selected by the motivations at level I (or II). EI is computed on the basis of the AV of the corresponding classifier(s) and on an execution time:

$$EI = (AV * 10000) - \max (T2S, ET)$$

In this formula, AV gives priority to classifiers which satisfy a maximum of motivations. T2S and ET select different classifiers with the same AV.

The *Action Resources* level (level IV)

Level IV provides resources for action execution, especially for behavioural animations like *Pitch Yaw, Button Attack, Forward move*, etc.

The action command with the highest EI value has the primacy to use the required resources. Other executable actions cannot require these already-used resources, and have only access to available ones. The behaviour that will be adopted by the Bot in the environment will be a combination of all the activated resources.

MHiCS Base and MHiCS Agent

MHiCS is built around two components: MHiCS Base and MHiCS Agent. All the rules of the different CS are stored in MHiCS Base. This base is unique and shared by all the active Bots. The purpose is to share knowledge and learning between the agents and to reduce memory used by classifiers storage. For each Bot there is a MHiCS Agent component. It is the part of MHiCS which takes track of the Bot motivation diffusions, active classifiers and actions as well as all dynamic information computed by the MHiCS algorithms for this Bot.

The Learning Process

In classical AI, the best next action to be done could be evaluated by ply research using heuristics. In video games, good heuristics are not easy to acquire by such means because of the multiplicity of possible consequences. For example, if a Bot’s death is disadvantageous for its frag score, it could be a useful sacrifice for its team’s goal – due to an efficient re-implementation of this Bot in the game. MHiCS has the convenience of not using heavy heuristics and ply research. It just selects the best classifier according to its T2S and ET values updated online at each time step. When a motivation decreases, T2S is updated for all classifiers that have been activated by this motivation since the last MV decrement. Each time an action stops its execution, the ET value is updated for all classifiers that have activated this action. After an update, the new value for the T2S and the ET of a classifier stored in the MHiCS Base is replaced by the weighted average between the previous stored values and the new one. Making such an average between old and new values smoothes the adaptation process.

EXPERIMENTS AND RESULTS

The experiments aim to test the MHiCS capacity to modify, through a learning process, the T2S and ET of each classifier in order to decrease the MV of their classifiers system.

In the test application (CTF scenario of TFC) there are 2 (1 vs. 1) or 8 (4 vs. 4) players. The game duration is set to 20 minutes. Our Bots have been tested by human players but, for quantitative results, we have compared them to HPB Bots developed by Botman, whose code is used by most of Half-Life MOD Bots - ours included. In HPB Bots, all the rules are hard-coded without learning capacities. They use a waypoint navigation system and a schedule to manage the different actions.

Here the red team is composed of HPB Bots and the blue team of MHiCS Bots. Each time a Bot from a team captures the opponent team’s flag (by bringing it back to the team base), the team increases its score with 10 points.

A MHiCS Bot has three motivations: *Aggressiveness* (RP: 2), *Team Score* (RP: 1) and *Individual score* (RP: 0.3). Aggressiveness MV increases when the Bot gets hurt and decreases when it kills an opponent. Team Score MV increases when opponent Team Score increases and decreases when the Team Score increases. Individual score MV increases with time and decreases when a Bot kills an opponent.

Its classifiers have six different condition parts: *Bot has flag, Bot has a target, Team has flag, Enemy team has flag, Damage recently taken* and *Target is my friend*; and eight different action parts: *approach target, shoot target, look around, move to 4 different waypoints (opponent team flag start, opponent team flag goal, my team flag start, my team flag goal)* and *move to opponent team flag*. As in a human team, a Bot can always know where the opponent team flag is.

There are three specific classifiers for the Aggressiveness CS, 10 for the Individual Score CS and 14 for the Team Score CS. In the experiments we will focus on the learning of this last CS. Four are for moving to each different waypoint when the Bot has the Flag, five when someone else in the team has the flag and five when no one in the team has the flag. Each classifier starts with the same T2S (1s) and ET (0s) values.

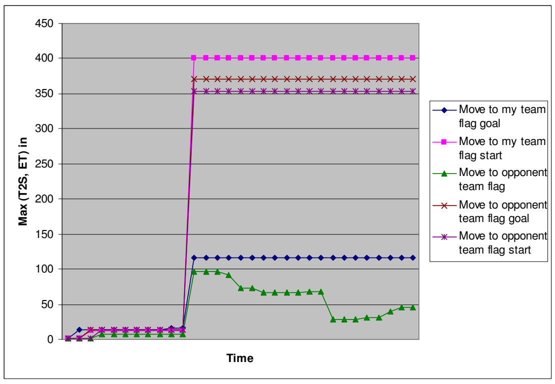

Figure 3: Max (T2S, ET) values of each classifier which condition is "Team has flag==False" in the game 2 of the first experiment

We have run each experiment on five games with the same initial conditions.

In the first experiment, one HPB Bot competes with one MHiCS Bot. The purpose is to demonstrate the MHiCS Bot's capacity to learn how to increase its Team Score in spite of the opponent's actions and its Aggressiveness motivation that might conflict with the *Team Score* motivation.

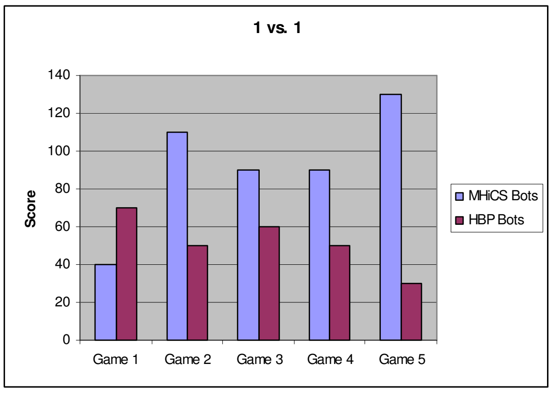

Figure 4: Score of both teams in the first experiment

Figure 3 shows two phases in the learning process. In the first phase the CS will select each classifier to find which one can satisfy the Team Score motivation. When for the first time the flag is successfully captured, Team Score motivation decreases and the T2S value is updated for each classifier. Here the classifier with the action part *Move to opponent team flag* has the minimum T2S value. As this classifier is a good one, it will continue to be selected and to adjust its T2S and ET values.

On the 5 games, the MHiCS Bots won 4 with an average Team Score of 92 over 52 (Figure 4) whereas they were starting with no initial knowledge on how to increase the team score and often captured their first flag after the opponent team had already captured many.

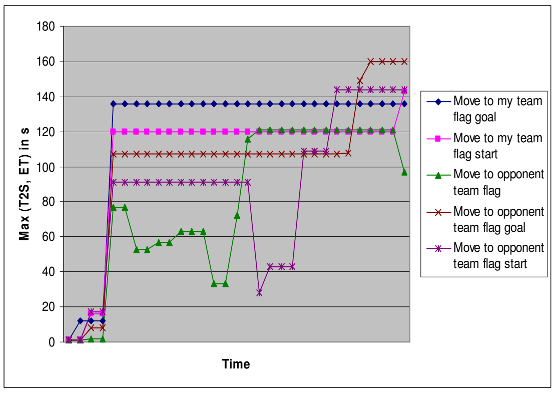

Figure 5: Max(T2S, ET) values of each classifier which condition is "Team has flag==False" in the game 4 of the second experiment

In the second experiment, 4 HPB Bots compete with 4 MHiCS Bots. The purpose is to demonstrate that, in a multi-agent environment, MHiCS Bots can still learn how to increase the score, with the same update signal (the motivation decrease) given to all team members whatever the actions already done (even though only one Bot can bring back the flag).

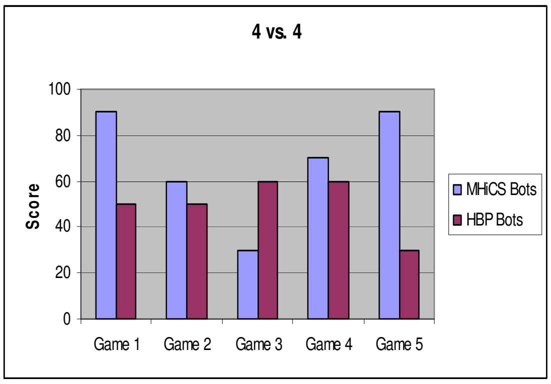

Figure 6: Score of both teams in the second experiment

With a greater number of opponents, Bots sometimes encounter difficulties in capturing the flag. Figure 5 shows how MHiCS handles this kind of situation. After learning that the classifier with the action part *Move to opponent team flag* is the most efficient, Bots encounter difficulties with this action. A second classifier with the action part *Move to opponent team flag start* becomes active and gives good results too. But the Bots encounter also difficulties with this new classifier. After testing without success the classifier with the action part *Move to opponent team start*, the classifier *Move to opponent team flag* is successfully selected just before the end of the game. This example shows the capacity of the MHiCS architecture to dynamically adapt the classifier priority.

On the 5 games, the MHiCS Bots won 4 (Figure 6) with an average Team Score of 68 over 50.

DISCUSSION AND CONCLUSION

As a classifier is basically an *"if condition then action"* rule, the CS of a Bot could be initialized by a game designer with quite a good starting classifiers set. In this paper, this first step has been automatically done by MHiCS, which has learned how to improve a given set of rules in order to increase the team score in a *Team Fortress Classic* scenario. The comparison of MHiCS Bots with hand-tuned HPB Bots reveals the efficiency of this automatic process.

In further experiments, a similar learning process will be used with the purpose of dynamically adapting Bots behaviours to specific players' tactics.

Classifier systems incorporate individual learning but they also integrate collective learning processes - i.e. evolution. Genetic algorithms allow the discovery of new useful classifiers and the elimination of bad ones. New classifiers are created by genetic operators like crossover and mutation, which exchange or transform the condition or action parts of old classifiers. Bad classifiers are rejected on the basis of appropriate fitness criterion.

Here this criterion will focus on the evaluation of max (T2S, ET) values on which the learning process is based. As shown in Figure 3 and Figure 5, these values are subject to great variations. To better assess the quality of a classifier, these variations will be taken into account, as some CSs already do (XCS, Wilson 1995; Lanzi 1999), by selecting classifiers associated with the most consistent values and removing classifiers associated with the most fluctuating ones.

Other improvements are also under consideration like the addition of other kinds of learning or planning abilities (YACS, Gérard 2002). With such abilities, Bots would be able to take advantage of generating plans to achieve given goals during a latent learning phase – i.e. without reward - and to learn much faster in complex environments.

LINKS

HPB Bot: http://www.planethalflife.com/botman/
TFC: http://www.planethalflife.com/tfc/

REFERENCES

Calderon C. and M. Cavazza. 2001. "Using games engines to implement intelligent virtual environments". In *Game-On 2001*, Q. Mehdi, N. Gough, and D. Al-Dabass (Eds.).SCS Europe Bvba, 71-75.

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.).The MIT Press/Bradford Books, 3-12.

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.

Khoo, A. and R. Zubek. 2002. "Applying Inexpensive AI Techniques to Computer Games". *IEEE Intelligent Systems*, No.17(4), 48-53.

Laird J.E. and J.C. Duchi. 2000. "Creating Human-like Synthetic Characters with Multiple Skill Levels: A Case Study using the Soar Quakebot". In *Papers from the 2000 AAAI Spring Symposium on Arti cial Intelligence and Computer Games*, 54-58.

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

Robert G., P. Portier and A. Guillot. 2002. "Classifier systems as 'Animat' architectures for action selection in MMORPG". In *Game-On 2002*, Q. Mehdi, N. Gough, and M. Cavazza (Eds.).SCS Europe Bvba, 121-125.

Tozour P. 2002." First-Person Shooter AI Architecture" In AI Game Programming Wisdom Jenifer Niles (Eds.). Hingham, Massachusetts 02043, 387-396.

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