An Adaptive Game AI Architecture

Emanuel M. Carneiro Adilson M. Cunha

Instituto Tecnológico da Aeronáutica, Computer Science Department, Brazil

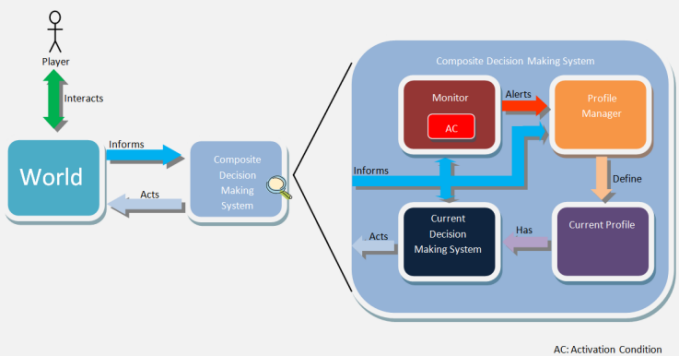

Figure 1: System architecture.

Abstract

Computer controlled characters in games are expected to present realistic behaviors and adapt them depending on current situation. Modern games use basic AI techniques that fail to meet this expectation.

This paper proposes an approach for a rapid and reliable adaptive game AI architecture based on reinforcement learning and Decision Making Systems composition.

Keywords: Game AI, adaptive behavior, reinforcement learning.

Authors' contact:
mineda@gmail.com
cunha@ita.br

1. Introduction

Even though there have been enormous advances in computer graphics, animation, and audio for games, most of them contain very basic Artificial Intelligence (AI) techniques, if any [Ram et al. 2007]. It's not an uncommon practice to increase challenge artificially by creating an imbalance between Non-Player Characters (NPC) and Player Characters (PC) rather than using a better game AI. The problem with this approach, based on the use of non-adaptive game AI, is that, once a weakness is discovered, nothing stops the human player from exploiting the discovery [Bakkes et al. 2009]. NPCs in modern games are predictable.

Adaptive game AI, capable of changing an NPC behavior based on the player behavior, can solve this problem. Unfortunately, this kind of AI, which usually makes use of machine learning techniques, often needs a huge amount of historical data and/or several training sessions to present a satisfying performance. In addition, some adaptive game AI can generate undesirable behaviors during the learning process. These behaviors can be considered efficient, when evaluated for some set of metrics but will look unnatural for a human spectator.

These issues are aggravated in linear single player games where there's little to none historical data available and the probability of playing against a defeated important NPC again is, in most cases zero, making it impossible to effectively apply classical machine learning training techniques.

This paper's objective is to present an approach to create a game AI architecture capable of adapting its behavior in-line, in a fast and reliable way, according to the player's inputs, improving the challenge level and reducing the predictability of NPCs.

2. Related Work

Computer controlled characters in games are expected to present realistic behaviors and adapt when faced with a disadvantageous situation. A lot of research has been made in recent years to fulfill this expectation.

Spronck et al. [2006] proposed a technique, named Dynamic Scripting, to dynamically adapt a rule base, in a rule-based system, using Reinforcement Learning. Adaptation runs between game sessions and is gradual. Another approach involving rule-based systems was presented by Crocomo et al. [2008] and makes use of evolutionary algorithms to generate new rule bases. As dynamic scripting, the adaptation runs between game sessions and is gradual.

A rapid and reliable case-based approach to adapt game AI was proposed by Bakkes et al. [2009]. It relies in samples of gameplay experience and is expected to present good results in games that have access to the Internet to store and recover these samples.

Tan and Cheng [2010] proposed an automated model-based approach built upon their Integrated MDP and POMDP Learning AgeNT (IMPLANT) architecture. This approach makes use of a player model based upon a weighted set of actions and mixes online and offline learning.

There are many other approaches for the problem, each one suited for some specific range of scenarios. The architecture proposed in this paper attempts to attack the problem of creating a rapid and reliable adaptive game AI from another angle, by combining simple specific Decision Making Systems (DMSs), implemented by using existing techniques.

3. Proposed Architecture

This section discusses the proposed architecture, presenting details on all of its components.

3.1 System architecture

System architecture, as presented in Figure 1, is composed by the following elements:

- The digital World that contains both PCs and NPCs and is represented by a collection of data that represents the state of each of its elements;
- The Player which interacts with the world through a PC; and
- The Composite Decision Making System (CDMS) that is responsible for controlling a NPC based upon observations of the world's state.

3.2 Composite Decision Making System

The CDMS, also shown in Figure 1, represents the core of this paper and is based on a simple idea. Instead of having a unique DMS capable of handling all kinds of player strategies, it has a collection of specific DMSs to choose from based on player profile.

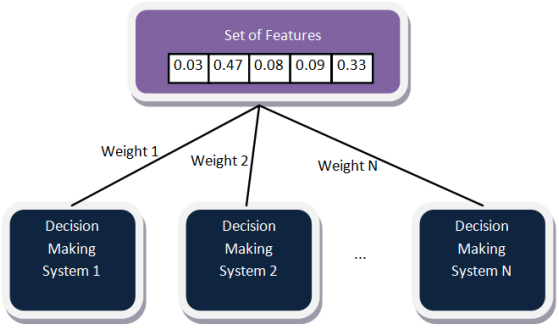

Figure 2: A profile

Before continuing, it is necessary to explain the profile concept in the context of this paper. Figure 2 exemplifies a typical profile. It is composed by a set of features that identify a specific player style and a set of weighted connections to all available DMSs in the CDMS. A greater weight means a better performance against a specific player style. "A percentage of player actions that were physical attacks of any kind" is a good example of a possible profile feature.

Monitor: is responsible for watching the world and producing alerts for the Profile Manager. An alert is generated every time the Activation Condition (AC) is met or a game session ends. The AC is a logical expression that identifies a bad NPC performance.

Current Profile: is the profile that best suits the player style and is defined by the Profile Manager.

Current Decision Making System: is chosen based upon the highest weighted connection between the Current Profile and all the available DMSs. It will choose the best action to perform based on the world's state and send it to the associated NPC.

Profile Manager: is responsible for recovering or creating profiles that match the human player behavior. It also uses Reinforcement Learning to manage the relative weights of DMSs connected to a profile.

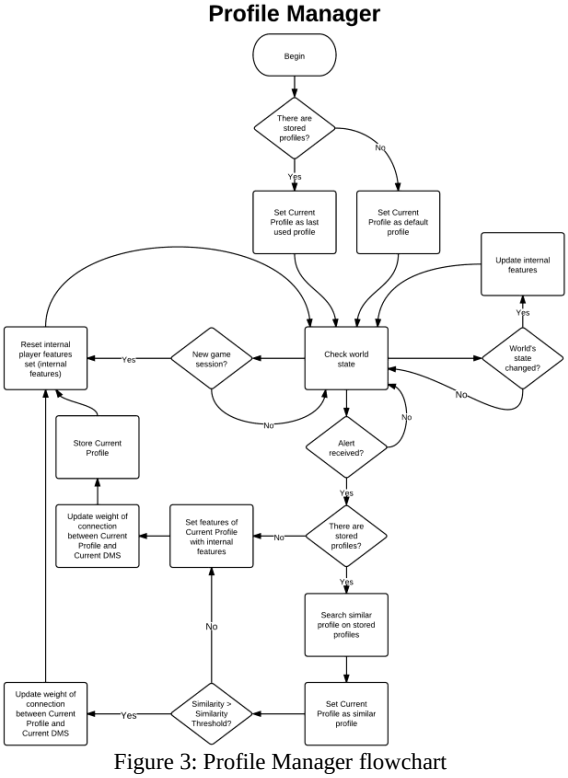

Figure 3: Profile Manager flowchart

In summary, Figure 3 shows that Profile Manager searches for a stored profile that matches the player style and updates the weights of its connections to a DMS, according to the results of the current game session. If there are no profiles that match the player

style a new one is created, based on the profile with the highest similarity, and then updated.

Cosine similarity is used to measure similarity between sets of features. It was chosen because sets of features are usually long and sparse (have many 0 values), and is denoted by

$$\text{similarity} = \cos(\theta) = \frac{A.B}{\|A\|.\|B\|} = \frac{\sum_{i=1}^{n} A_i \times B_i}{\sqrt{\sum_{i=1}^{n}(A_i)^2} \times \sqrt{\sum_{i=1}^{n}(B_i)^2}}$$

The resulting value lies between -1 and 1, where 1 means identical and -1 means the exact opposite.

Similarity threshold is a constant that indicates the minimum value of similarity needed to consider two sets of features as the same.

Weighted connections between profiles and DMSs are updated using a performance update function.

$$\text{performance update function} \begin{cases} \geq 0 \text{ if NPC performance is satisfying} \\ < 0 \; if \; NPC \; performance \; is \; unsatistying \end{cases}$$

A default profile has an empty set of features and is connected to all available DMSs with default weights.

3.3 Applicability

The proposed architecture can only be applied in cases where it is possible to define a player profile based upon a set of features extracted from the available in-game data.

3.4 Computational Cost

The total computational cost of the CDMS is

$$O(f \times d) + \text{ Highest DMS cost}$$

Where f is the amount of features in the profile and d is the amount of DMS composing the CDMS. DMS computational costs tend to be far greater than O(f x d). So, in most situations, the total computational cost of the CDMS will be proportional do the cost of the DMS with the worst performance.

4. Implementation

This session presents some details on the game and the architecture implementation used on the experiment.

To validate the proposed architecture a simple turn based game was implemented. Two agents, a PC and an NPC, with only one main attribute, Health, and a minimal set of secondary attributes, compose the game's world. Health has an initial value of 1000 and to win it is necessary to reduce an agent's Health to 0.

In order to simplify the analysis process all DMS are Rule Based Systems. There are a total of five different DMSs. Two for the PC: both named PDMS1 and PDMS2. And three for the NPC: named CDMS1, CDMS2, and CDMS3.

DMSs were designed to present the performance detailed on Table 1.

Table 1: Performance of PC DMSs against NPC DMSs

	CDMS1	CDMS2	CDMS3
PDMS1	Strong	Strong	Very weak
PDMS2	Weak	Very weak	Strong

The function used to evaluate the performance of the CDMS is denoted by

$$\text{performance} = (\text{ph1} - \text{ph2}) - (\text{nh1} - \text{nh2})$$

Where ph1 is the PC's Health, when the Current DMS was chosen; ph2 is the PC's current Health; nh1 is the NPC's Health, when the Current DMS was chosen; and nh2 is the NPC's current Health.

AC is denoted by

$$\text{performance} < -100$$

Performance Update Function is denoted by

$$\text{performance update function} = \frac{\text{performance}}{\text{turns with Current DMS}}$$

The profile is composed by the following set of features: movement, physical attacks, ranged attacks, defensive actions, debuffing actions, cleaning action, concentrate action, and energize action. Default profile connections weights are presented on Table 2.

Table 2: Default profile weights

	CDMS1	CDM2	CDMS3
Weight	1000	999	998

5. Experiment

This session presents some details on the experiment used to test the implementation of the architecture.

The experiment held four consecutive game sessions. The PC used PDMS1, PDSM2, PDMS1, and PDMS2 in this specific order.

As results of the experiment it was expected that:

- Two profiles, P1 and P2, were created, representing PDMS1 and PDMS2; and
- The final weights of P1 and P2 resembled the programmed setup presented in Table 1.

6. Results Analysis

This session presents an analysis on the results of the experiment.

First Game Session: When the AC was met on the 13th turn, there was no profile stored, so P1 was created. As there was no historical data, CDMS2 were used, but presented a bad performance. The only DMS with a good performance against P1 is CDMS3.

Second Game Session: When the AC was met on the 86th turn, the Profile Manager searched for a profile that matched the player style, but found nothing. P2 was then created, based on the values of P1. As the performance of CDMS3 against P1 was very strong, it took three alerts to devaluate the weight of its connection enough for CDMS1 to be used.

Third Game Session: When the AC was met on the 142th turn, the Profile Manager recognized P1 as a profile that matched the player style and CDMS3 was immediately chosen.

Fourth Game Session: When the AC was met on the 209th turn, the Profile Manager recognized P2 as a profile that matched the player style and CDMS1 was immediately chosen.

Figure 4 summarizes the obtained results.

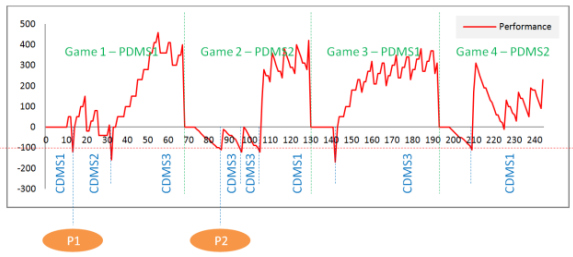

Figure 4: Experiment results

6.1 Generated Profiles

Table 3 presents the final weights of the profiles generated during the experiment.

Table 3: Profiles weights

	CDMS1	CDMS2	CDMS3
P1	978	991	1015
P2	1016	999	972

7. Discussion

From the experiment's results, it was possible to observe that the proposed architecture was able to meet all expectations.

Profiles P1 and P2 were correctly associated to PDMS1 and PDM2, and the automatically generated weights of both profiles, presented on Table 3, resemble the programmed setup presented on Table 1. The only difference being the connection between P2 and CDMS2, what is understandable as it was not necessary to try this configuration as CDMS1 presented a performance that was good enough.

The proposed architecture was able to adapt in a very short time, even with the bias introduced via the default profile, requiring one match to identify the most suited DMS. Besides that, it continued to prune the weights using reinforcement learning, making it possible for it to adapt even with suboptimal sets of features.

The DMSs were responsible for deciding the actions to take and they were programmed to generate valid actions, so no undesirable behavior was generated. It must be noticed that the architecture will not prevent undesirable behavior if unreliable DMS is used in its composition.

8. Conclusion

This paper presented an approach for a game AI adaptive architecture based on reinforcement learning. The experiment's results have shown that, in the given scenario, it could adapt in a rapid and reliable way, without the need of long training periods or huge amount of historical data.

This is still a work in progress. Though, it is still requiring further research, in order to cover other scenarios. An analysis of its performance against adaptive DMSs, which are better approximations to human players, is highly recommended.

Future works could focus on the automatic creation of new DMSs, for situations where none of the current DMSs can present an adequate challenge, when faced against a certain player style.

Acknowledgements

The authors would like to thank all the people that make this work possible, directly and indirectly contributing to it.

References

BAKKES, S., SPRONCK, P., AND HERIK, J. VAN DEN, 2009. Rapid and reliable adaptation of video game AI. *IEEE transactions on computational intelligence and AI in games*. IEEE Press, v. 1, p. 93-104.

CROCOMO, M. K., AND SIMÕES E. V., 2008. Um algoritmo evolutivo para aprendizado online em jogos eletrônicos. *In: Proceedings of SBGames, Belo Horizonte, MG*. SBC, p. 159-168.

RAM, A., ONTAÑÓN, S., AND MEHTA, M., 2007. Artificial intelligence for adaptive computer games. *In: Proceedings of the International Flair conference on Artificial Intelligence, 2007, Key West, FL*. AAAI Press.

SPRONCK, P., PONSEN, M., SPRINKHUIZEN-KUYPER, I, AND POSTMA, E., 2006. Adaptive game AI with dynamic scripting. *Machine Learning*. Kluwer Academic Publishers, v. 63, p. 217-248.

TAN, C. T. AND CHENG, H., 2010. An automated model-based adaptive architecture in modern games. *In: Proceedings of the Sixth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, 2007, Stanford, CA*. AAAI Press, p. 186-191.