Online Adaptation of Computer Game Opponent AI

Pieter Spronck Ida Sprinkhuizen-Kuyper Eric Postma

Universiteit Maastricht, P.O. Box 616, 6200 MD Maastricht

Abstract

Online learning in commercial computer games allows computer-controlled opponents to adapt to human player tactics. For online learning to work in practice, it must be fast, effective, robust, and efficient. This paper proposes a technique called "dynamic scripting" that meets these requirements. In dynamic scripting an adaptive rule-base is used for the generation of intelligent opponents on the fly. The adaptive performance of dynamic scripting is evaluated in an experiment in which the adaptive players are pitted against a collective of manually designed tactics in a simulated computer roleplaying game. The results indicate that dynamic scripting succeeds in endowing characters with adaptive performance. We therefore conclude that dynamic scripting can be successfully applied to the online adaptation of computer game opponents.

1 Introduction

The quality of commercial computer games is directly related to their entertainment value [9]. The general dissatisfaction of game players with the current level of artificial intelligence for controlling opponents (so-called "opponent AI") leads to their preference for human-controlled opponents [6]. Improving the level of opponent AI (while preserving the characteristics associated with the entertainment value [7]) is desired in case human-controlled opponents are not available or not feasible.

For complex games most game AI developers resort to scripts [10]. Controlling opponents with complex abilities requires these scripts to be quite long [1]. Two major weaknesses of long scripts are that (1) they are prone to containing errors because they are complex, and (2) they cannot deal with unforeseen player tactics because they are static. As a result, a human player can easily defeat supposedly tough opponents by exploiting the "holes" in their scripts. Evidently, easily defeatable opponents hamper the entertainment value of commercial computer games.

In our view, there are two ways to improve the quality of scripted opponent AI. The first way is to employ offline learning prior to the release of a game to deal with the problem of holes in the scripts that control the opponents. Our earlier work showed offline learning techniques to be successful in the identification of holes and even in the discovery of novel tactics [8]. The second way of improving the quality of scripted opponent AI is to apply online learning after the game has been released. Online learning allows the opponents to adapt to changes in human player tactics. Even though it has been shown to be feasible for simple games [2], unsupervised online learning is widely disregarded by commercial game developers [11] and literature on unsupervised online learning in commercial computer games is scarce. However, we believe it to be

of great potential for improving the entertainment value of commercial computer games.

Our research question reads: How can unsupervised online learning be incorporated in commercial computer games? This paper proposes a novel technique called "dynamic scripting", that realises unsupervised online adaptation of scripted opponents, and reports on preliminary experiments performed to assess the adaptive performance obtained with dynamic scripting.

The outline of the remainder of the paper is as follows. Section 2 discusses AI in computer roleplaying games. Section 3 describes our dynamic scripting technique. The experimental setup for evaluating the adaptive performance of dynamic scripting is described in section 4. The results of the experiments are presented and discussed in sections 5 and 6, respectively. Finally, section 7 concludes that dynamic scripting has the potential to be successfully incorporated in commercial games.

2 Computer Roleplaying Game AI

In Computer RolePlaying Games (CRPGs) the human player is situated in a virtual world represented by a single character or group of characters called a "party". Each character is of a specific type (e.g., a fighter or a wizard) and has certain characteristics (e.g., weak but smart). In most CRPGs, the human player goes on a quest, which involves conversing with the world's inhabitants, solving puzzles, discovering secrets, and defeating opponents in combat. During the quest the human-controlled characters gather "experience", thereby gaining more and better abilities, such as advanced spell-casting powers. Some examples of modern CRPGs are BALDUR'S GATE, NEVERWINTER NIGHTS, MORROWIND and EVERQUEST.

Opponent AI in CRPGs is almost exclusively based on scripts, i.e., lists of rules that are executed sequentially. Scripts have four main advantages; they are (1) understandable, (2) easy to implement, (3) easily extendable, and (4) useable by nonprogrammers [10]. Usually scripts are implemented in a formal language that has functions to test environmental conditions, to check a character's status, and to express commands. During the game-development phase scripts are manually adapted to ensure they exhibit the desired behaviour. After the game is released the scripts and associated behaviours remain unchanged (unless game patches are issued to update the scripts).

To deal with all possible choices and all possible consequences of actions the scripts controlling the opponents are of relatively high complexity. In contrast to classic CRPGs, such as the ULTIMA series, in modern CRPGs the human and opponent parties are often of similar composition (see figure 1 for an example), which entails that the opponent AI should be able to deal with the same kind of complexities the human player faces. The challenge such an encounter offers can be highly enjoyable for human players.

Figure 1: An encounter between two parties in BALDUR'S GATE.

However, the scripts controlling the opponents cannot anticipate on all tactics exhibited by human players. As a consequence, human players usually have little trouble in identifying and exploiting the weaknesses in the opponents. Since these weaknesses usually permeate throughout the entire game, this eventually leads to a decrease in entertainment value of the game.

3 Dynamic Scripting

We introduce dynamic scripting as a technique to overcome the limitations of static scripts in CRPGs. In dynamic scripting, the scripts controlling the opponents are modified during the game to adapt to the tactics of the human player. For online learning to work in practice, it must be fast (computationally cheap), effective (maintain at least the quality of the unadapted scripts), robust (able to deal with the inherent randomness of computer games), and efficient (require few experiments).

To achieve the goal of fast, effective, robust and efficient dynamic scripting, we need a learning algorithm of high performance. The two main factors of importance when attempting to achieve high performance for a learning mechanism are using deterministic experiments and adding knowledge [4]. The nature of our environment precludes determinism, so in our case it is imperative that the learning process is based on knowledge. To this end, our dynamic scripting technique relies on a rulebase that contains manually designed rules based on domain-specific knowledge.

The dynamic-scripting process is illustrated in figure 2. A single rulebase is associated with every opponent. Each rule in the rulebase has a weight indicating its importance. At the start of an encounter, a new script is generated for each opponent by randomly selecting a fixed number of rules from its rulebase. The probability of a rule being selected depends on its weight; i.e., rules with larger weights have a higher probability of being selected.

The learning mechanism in our dynamic scripting technique is inspired by reinforcement learning techniques [5]. It has been adapted for use in games because

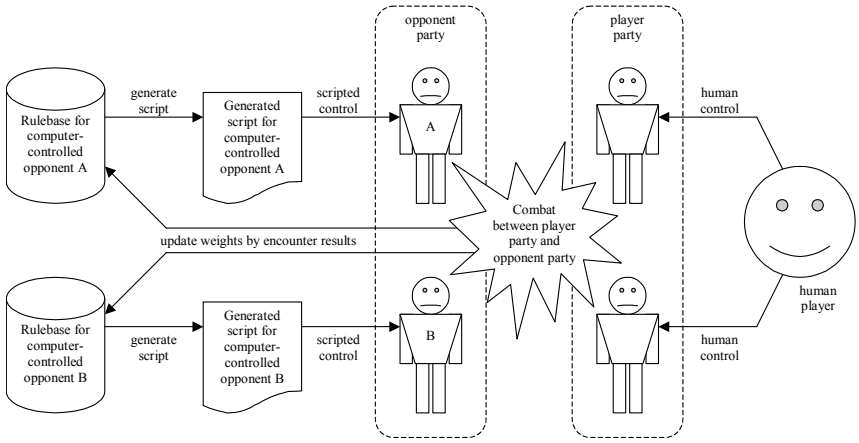

Figure 2: The dynamic-scripting process. For each computer-controlled opponent a rulebase generates a new script at the start of an encounter. After an encounter is over, the weights in the rulebase are adapted to reflect the results of the fight.

regular reinforcement learning techniques are unsuitable for this purpose, in particular since (1) it is difficult to decide what information should be placed in a state vector, and (2) reinforcement learning generally adapts too slowly for online use in games [3]. Our implementation is as follows. Upon completion of the encounter, the weights of the rules employed in the combat are adapted depending on their contribution to the outcome. Increasing the weight rewards rules that lead to success, whereas decreasing the weight punishes rules that lead to failure. The remaining rules get updated so that the total weight of the rules in the rulebase remains unchanged.

The dynamic-scripting technique meets at least three of the four requirements listed above. First, it is fast because it only requires the extraction of rules from a rulebase and the updating of weights once per encounter. Second, it is effective because all rules in the rulebase are based on domain knowledge (although they may be inappropriate for certain situations). Third, it is robust because rules are not removed immediately when punished. The dynamic-scripting technique is believed to meet the fourth requirement of efficiency because with appropriate weight-updating parameters it can adapt already after a few encounters. To determine whether this belief is warranted, we performed an experiment with dynamic scripting in a simulated CRPG situation.

4 Experimental Setup

This section describes the experimental setup used to test the efficiency of dynamic scripting in CRPGs. It describes the problem situation to which dynamic scripting is applied (4.1), the scripts and rulebases (4.2), the fitness functions (4.3), the learning parameters (4.4) and the actual experiments (4.5).

4.1 The CRPG Simulation

The gameplay mechanism in our CRPG simulation was designed to resemble the popular BALDUR'S GATE games (see figure 1). These games contain the most complex and extensive gameplay system found in modern CRPGs, closely resembling classic "pen 'n paper" roleplaying games. Our simulation entails an encounter between player and opponent parties of similar composition. Each party consists of two fighters and two wizards of equal experience level. The armament and weaponry of the party is static; each character is allowed to select two (out of three possible) magic potions; and the wizards are allowed to memorise seven (out of 21 possible) spells. The spells incorporated in the simulation are of varying types, amongst which damaging spells, blessings, curses, charms, area-effect spells and the summoning of allies.

The choices for potions and spells are made before the encounter starts and depend on the (generated) scripts. In the simulation, this is done as follows. Before the encounter starts the script is scanned to find rules containing actions that refer to drinking potions or casting spells. When such a rule is found, a potion or spell which can be used in that action is selected. If the character controlled by the script is allowed to possess the potion or spell, it is added to the character's inventory.

4.2 Scripts and Rulebases

The scripting language is designed as to define rules composed of an optional conditional statement and a single action. The conditional statement consists of one or

more conditions combined with logical ANDs and ORs. Conditions can refer to a variety of environmental conditions, such as the distances separating characters, the characters' health, and the spells that are suffered or benefited from. There are five basic actions: (1) attacking an enemy, (2) drinking a potion, (3) casting a spell, (4) moving, and (5) passing. In the scripting language, spells, potions, locations and characters can be referred to specifically (e.g., "cast spell 'magic missile' at closest enemy wizard"), generally (e.g., "cast any offensive spell at a random enemy") or somewhere in-between (e.g., "cast the strongest damaging spell available at the weakest enemy").

Scripts are executed in sequential order. For each rule the condition (if present) is checked. If the condition is fulfilled (or absent), the action is executed if it is both possible and useful in the situation at hand. If no action is selected when the final rule is checked, the default action 'pass' is used.

As explained in section 3, the rulebase consists of a list of weighted rules. The weight determines the probability that a rule from the rulebase is selected for the script that is generated at the start of an encounter. In addition, each rule is assigned a priority. The priority determines the position of the rule in the script. If two rules with the same priority are selected, the rule with the highest weight value gets precedence over the other. If both rules have the same weight value, their ranking is determined randomly.

4.3 The Fitness Functions

For the weight adaptation mechanism two fitness functions are used in the CRPG simulation: a fitness function for the party as a whole, and a fitness function for each individual character. The fitness of a party is a value on the unit interval [0,1]. The fitness is defined to be zero if the party has lost the fight, and 0.5 + half the average remaining health of all party members if the party has won the fight.

The fitness of a character is also a value on the unit interval [0,1], that is based on four factors, namely (1) the average remaining health of all party members (including the character), (2) the average damage done to the opposing party, (3) the remaining health of the character (or, if he died, the time of death) and (4) the party fitness. The fitness function for individual characters assigns a large reward to a victory of its party (even if the individual itself did not survive), a smaller reward to the individual's own survival, and an even smaller reward to the survival of its comrade party members and the damage they inflicted on the opposing party. This definition therefore attempts to illicit the emergence of successful party behaviour, and in a lesser sense the aim of a character to ensure its own survival.

4.4 The Learning Parameters

The learning parameters determine how the character fitness of a script is translated into adaptations of the weights associated with the rules in the rulebase. For our experiment we set the break-even point to 0.3, i.e., with a character fitness lower than 0.3 the weights of the rules executed during the encounter were penalised whereas those with a fitness higher than 0.3 were rewarded. All weights in the rulebase were initialised with a value of 100 and were constrained to values between 0 and 2000.

The maximum reward (awarded to scripts with the maximum fitness of 1) was set to 100. The maximum penalty (issued for a minimum fitness of 0) was set to 30. At the break-even point the weights are not adapted. For other fitness values the weight

adaptation is calculated as a linear mapping between the break-even point and the maximum, e.g. a fitness of 0.65 was rewarded with 50 points. To keep the sum of all weight values in a rulebase constant, weight changes correspond to a redistribution of weights in the rulebase. Note that the weight shifts in the rulebase will, if successful, generate opponent behaviour that favours winning a fight. Whether or not this is fun for the human player is currently not taken into account, although stronger computer-controlled opponents are usually considered to be more fun for experienced players.

The size of the script for a fighter was set to five rules, which were selected out of a rulebase containing 20 rules. For a wizard, the script size was set to ten rules, which were selected out of a rulebase containing 50 rules. One or two default rules were added to the end of each script to ensure the execution of an action in case none of the rules from the rulebase could be activated.

4.5 The Experiments

The experiments aim at assessing the adaptive performance of an opponent party controlled by the dynamic scripting technique, against a player party controlled by static scripts. To quantify the relative performance of the opponent party against the player party, after each encounter for both parties we calculate the average of the fitness over the last ten encounters. If for the opponent party this number is higher, the opponent party *outperforms* the player party. We define the *average turning point* as the encounter after which the opponent party first outperforms the player party for at least ten consecutive encounters. Furthermore, we define the *absolute turning point* as the first encounter after which a consecutive sequence of encounters in which the opponent party wins is never followed by a longer consecutive sequence in which the opponent party loses. Low values for the turning points indicate good efficiency of dynamic scripting, since they entail that the opponent party, which uses dynamic scripting, needs only a few encounters to achieve the goal of outperforming an unchanging tactic, as used by the player party.

Four different static tactics were employed by the player party, namely the following: (1) strongly offensive, (2) disabling (freezing the enemies before attacking them), (3) cursing (hindering and weakening the enemies), and (4) strongly defensive. To assess the ability of the dynamic-scripting technique to cope with sudden changes in tactics, we defined three additional composite tactics: (1) random-party (which randomly picks one of the four basic tactics for each encounter), (2) random-character (whereby each character picks his own tactic independent from his comrades), and (3) consecutive-party (whereby a party keeps using one of the four basic tactics until it loses a fight, then switches to the next tactic).

For each of the basic tactics we ran 21 tests, and for each of the composite tactics we ran 11 tests. The results of these experiments are presented in the next section.

5 Results

Table 1 presents the results of the experiments described in section 4. We make the following three observations. The first observation is that the disabling tactic is easily defeated. Apparently the disabling tactic is not a good tactic, because it does not require adaptation of the rulebase to be dealt with. The second striking observation is that for

	Average Turning Point				Absolute Turning Point			
Tactic	Low	High	Avg.	Med.	Low	High	Avg.	Med.
Offensive	27	164	57	54	27	159	53	45
Disabling	11	14	11	11	1	10	3	1
Cursing	13	1784	150	31	4	1778	144	31
Defensive	11	93	31	23	1	87	27	18
Random Party	13	256	56	29	5	251	50	26
Random Char.	11	263	53	30	1	249	47	33
Consecutive	11	160	61	50	3	152	55	48

Table 1: Results of the experiments described in section 4. For each tactic the lowest, highest, average and median average and absolute turning points are shown.

both turning points the average is (in most cases) significantly higher than the median. The explanation is found in the rare occurrence of extremely high turning points. During early encounters chance can cause potentially successful rules to get a low rating or unsuccessful rules to get a high rating. As a result, the rulebase diverges from a good weight distribution to which it has trouble to return. Our technique has no mechanism to reduce the effect of early divergence, but it is clear that such a mechanism is needed to make dynamic scripting a useful technique. Third, the consecutive tactic, which is closest to human player behaviour, is overall the most difficult to defeat with dynamic scripting. Nevertheless, our dynamic-scripting technique is capable of defeating this tactic rather quickly.

6 Discussion

Our experimental results show that dynamic scripting is capable of adapting rapidly to static or changing tactics. Hence, dynamic scripting is efficient and fulfils the fourth requirement stated in section 3.

Although dynamic scripting turns out to be surprisingly efficient, the question remains whether it is sufficiently efficient for application in commercial games. For action CRPGs, such as DIABLO, the answer would be an unequivocal yes, because action CRPGs typically pit the player against hundreds of similar opponents. For more strategic CRPGs, such as BALDUR'S GATE, it depends on the definition of "similar opponents". While each opponent wizard in the game might be different, overall successful tactics for one wizard will also work for most other wizards. Furthermore, in our experiments we started our rulebase from scratch with identical weights for all rules. In a commercial release the rulebase would have been trained offline against pre-programmed scripts (cf. our experiments with the consecutive-party tactic). Confronted with standard tactics such a rulebase would adapt very quickly, while it still would have the ability to learn to generate good scripts to deal with novel tactics.

Finally, it should be noted that while our fitness criterion relied heavily on winning and losing encounters, in commercial games the fitness criterion should be different. In commercial games, the human player should (because of entertainment purposes) and will (because of saving and reloading functionalities) always win an encounter. In an actual commercial game fitness should rely more on the length of a fight and the amount of damage done. It might even be useful to punish a rulebase for winning a fight, so that the entertainment value of the game for weaker players is assured.

7 Conclusions

In this paper we proposed dynamic scripting as a technique to deal with online adaptation of opponent AI, suitable for complex commercial computer games such as CRPGs. Dynamic scripting is based on the automatic online generation of AI scripts for computer-game opponents by means of an adaptive rulebase. From our experimental results, we conclude that that dynamic scripting is fast, effective, robust, and efficient and therefore has the potential to be successfully incorporated in commercial games, although some more work must be done before the technique is ready to be implemented in actual commercial games.

References

[1] Mark Brockington and Mark Darrah. How *Not* to Implement a Basic Scripting Language. *AI Game Programming Wisdom* (ed. S. Rabin), pp. 548-554, Charles River Media, 2002.

[2] P. Demasi and A.J. de O. Cruz. Online Coevolution for Action Games. *GAME-ON 2002 3rd International Conference on Intelligent Games and Simulation* (eds. Q. Medhi, N. Gough and M. Cavazza), pp. 113-120, SCS Europe Bvba, 2002.

[3] John Manslow. Learning and Adaptation. *AI Game Programming Wisdom* (ed. S. Rabin), pp. 557-566. Charles River Media, 2002.

[4] Zbigniew Michalewicz and David B. Fogel. *How to Solve It: Modern Heuristics*. Springer Verlag, 2000.

[5] Stuart Russell and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Second Edition, Prentice Hall, Englewood Cliffs, New Jersey, 2002.

[6] Jonathan Schaeffer. A Gamut of Games. *AI Magazine*, Vol. 22, No. 3, pp. 29-46, 2001.

[7] Bob Scott. The Illusion of Intelligence. *AI Game Programming Wisdom* (ed. S. Rabin), pp. 16-20, Charles River Media, 2002.

[8] Pieter Spronck, Ida Sprinkhuizen-Kuyper and Eric Postma. Improving Opponent Intelligence through Machine Learning. *Proceedings of the Fourteenth Belgium-Netherlands Conference on Artificial Intelligence* (eds. Hendrik Blockeel and Marc Denecker), pp. 299-306, 2002.

[9] Paul Tozour. The Evolution of Game AI. *AI Game Programming Wisdom* (ed. S. Rabin), pp. 3-15, Charles River Media, 2002.

[10] Paul Tozour. The Perils of AI Scripting. *AI Game Programming Wisdom* (ed. S. Rabin), pp. 541-547, Charles River Media, 2002.

[11] Steven Woodcock. Game AI: The State of the Industry. *Game Developer Magazine*, August 2000.