Enhancing the Performance of Dynamic Scripting in Computer Games

Pieter Spronck[1], Ida Sprinkhuizen-Kuyper[1], and Eric Postma[1]

[1] Universiteit Maastricht, Institute for Knowledge and Agent Technology (IKAT), P.O. Box 616, NL-6200 MD Maastricht, The Netherlands
{p.spronck, kuyper, postma}@cs.unimaas.nl

Abstract. Unsupervised online learning in commercial computer games allows computer-controlled opponents to adapt to the way the game is being played. As such it provides a mechanism to deal with weaknesses in the game AI and to respond to changes in human player tactics. In prior work we designed a novel technique called "dynamic scripting" that is able to create successful adaptive opponents. However, experimental evaluations indicated that, occasionally, the time needed for dynamic scripting to generate effective opponents becomes unacceptably long. We investigated two different countermeasures against these long adaptation times (which we call "outliers"), namely a better balance between rewards and penalties, and a history-fallback mechanism. Experimental results indicate that a combination of these two countermeasures is able to reduce the number of outliers significantly. We therefore conclude that the performance of dynamic scripting is enhanced by these countermeasures.

1 Introduction

The quality of commercial computer games is directly related to their entertainment value [1]. The general dissatisfaction of game players with the current level of artificial intelligence for controlling opponents (so-called "opponent AI") makes them prefer human-controlled opponents [2]. Improving the quality of opponent AI (while preserving the characteristics associated with high entertainment value [3]) is desired in case human-controlled opponents are not available.

In complex games, such as Computer RolePlaying Games (CRPGs), where the number of choices at each turn ranges from hundreds to even thousands, the incorporation of advanced AI is difficult. For these complex games most AI researchers resort to scripts, i.e., lists of rules that are executed sequentially [4]. These scripts are generally static and tend to be quite long and complex [5]. Because of their complexity, AI scripts are likely to contain weaknesses, which can be exploited by human players to easily defeat supposedly tough opponents. Furthermore, because they are static, scripts cannot deal with unforeseen tactics employed by the human player and cannot scale the difficulty level exhibited by the game AI to cater to both novice and experienced human players.

In our research we apply machine-learning techniques to improve the quality of

scripted opponent AI. When machine learning is used to allow opponents to adapt while the game is played, this is referred to as "online learning". Online learning allows the opponents to automatically repair weaknesses in their scripts that are exploited by the human player, and to adapt to changes in human player tactics. While supervised online learning has been sporadically used in commercial games [6], unsupervised online learning is widely disregarded by commercial game developers [7], even though it has been shown to be feasible for games [8,9,10].

We designed a novel technique called "dynamic scripting" that realises online adaptation of scripted opponent AI, specifically for complex games [10]. While our evaluations showed that dynamic scripting meets all necessary requirements to be generally successful in games, we noted that, occasionally, chance causes adaptation to a new tactic to take too long. In the distribution of adaptation times, these exceptionally long adaptation times are outliers. The present research investigates two countermeasures against the occurrence of outliers, namely penalty balancing and history fallback.

The outline of the remainder of the paper is as follows. Section 2 discusses opponent AI in games and describes dynamic scripting. It also discusses the results achieved with dynamic scripting in a simulation and in the state-of-the-art CRPG NEVERWINTER NIGHTS. Section 3 presents the two countermeasures, and the results obtained by applying them in dynamic scripting. Section 4 discusses the results. Finally, section 5 concludes and points at future work

2 Online Learning of Game Opponent AI with Dynamic Scripting

Online learning of computer game AI entails that the AI is adapted while the game is being played. In subsection 2.1 we present dynamic scripting as a technique that is designed specifically for this purpose. Those interested in a more detailed exposition of dynamic scripting are referred to [10]. Subsection 2.2 discusses online learning requirements for games and how dynamic scripting meets them. Subsection 2.3 presents the results of an evaluation of the effectiveness of dynamic scripting.

2.1 Dynamic Scripting

Dynamic scripting is an unsupervised online learning technique for commercial computer games. It maintains several rulebases, one for each opponent type in the game. The rules in the rulebases are manually designed using domain-specific knowledge. Every time a new opponent of a particular type is generated, the rules that comprise the script that controls the opponent are extracted from the corresponding rulebase. The probability that a rule is selected for a script is influenced by a weight value that is associated with each rule. The rulebase adapts by changing the weight values to reflect the success or failure rate of the corresponding rules in scripts. A priority mechanism can be used to let certain rules take precedence over other rules. The dynamic scripting process is illustrated in figure 1 in the context of a commercial game.

The learning mechanism in our dynamic scripting technique is inspired by rein-

forcement learning techniques [11]. It has been adapted for use in games because regular reinforcement learning techniques are not sufficiently efficient for online learning in games [12]. In the dynamic scripting approach, learning proceeds as follows. Upon completion of an encounter, the weights of the rules employed during the encounter are adapted depending on their contribution to the outcome. Rules that lead to success are rewarded with a weight increase, whereas rules that lead to failure are punished with a weight decrease. The remaining rules get updated so that the total of all weights in the rulebase remains unchanged. The size of the weight change depends on how well, or how badly, a team member performed during the encounter.

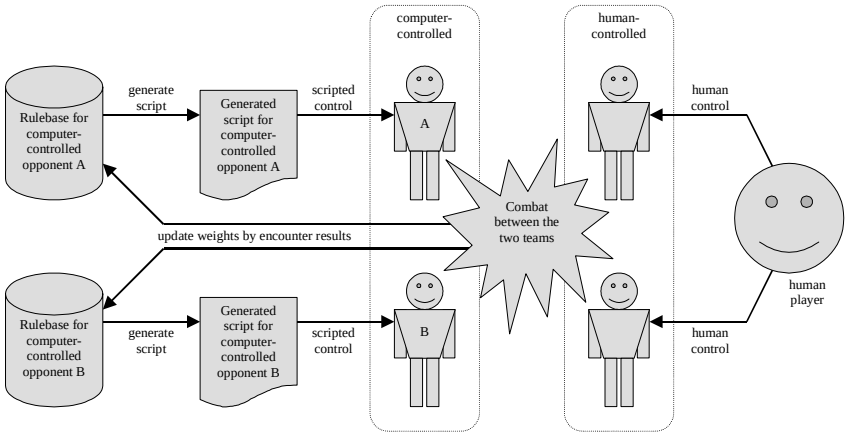

Fig. 1. The dynamic scripting process. For each computer-controlled opponent a rulebase generates a new script at the start of an encounter. After an encounter is over, the weights in the rulebase are adapted to reflect the results of the fight

2.2 Online Learning Requirements and Dynamic Scripting

For unsupervised online learning of computer game AI to be applicable in practice, it must be fast, effective, robust, and efficient. Below we discuss each of these four requirements in detail.

1. **Fast**. Since online learning takes place during gameplay, the learning algorithm should be computationally fast. This requirement excludes computationally intensive learning methods such as model-based learning. Dynamic scripting only requires the extraction of rules from a rulebase and the updating of weights once per encounter, and is therefore computationally fast.
2. **Effective**. In providing entertainment for the player, the adapted AI should be at least as challenging as manually designed AI (the occasional occurrence of a non-challenging opponent being permissible). This requirement excludes random learning methods, such as evolutionary algorithms. Dynamic scripting extracts the rules for the script from a rulebase, which contains only rules that have been manually designed using domain knowledge. Since none of the rules in the script

will be ineffective, the script as a whole won't be either, although it may be inappropriate for certain situations.

3. **Robust**. The learning mechanism must be able to cope with a significant amount of randomness inherent in most commercial gaming mechanisms. This requirement excludes deterministic learning methods that depend on a gradient search, such as straightforward hill-climbing. Dynamic scripting is robust because it uses a reward-and-penalty system, and does not remove rules immediately when punished.
4. **Efficient**. In a single game, a player experiences a limited number of encounters with similar groups of opponents. Therefore, the learning process should rely on just a small number of trials. This requirement excludes slow-learning techniques, such as neural networks, evolutionary algorithms and reinforcement learning. With appropriate weight-updating parameters dynamic scripting can adapt after a few encounters only. We have evaluated the efficiency of dynamic scripting with experiments that are discussed in subsection 2.3.

2.3 Evaluation of the Efficiency of Dynamic Scripting

To evaluate the efficiency of dynamic scripting, we implemented it in a simulation of an encounter of two teams in a complex CRPG, closely resembling the popular BALDUR'S GATE games (the simulation environment is shown in figure 2). We also implemented dynamic scripting in an actual commercial game, namely the state-of-the-art CRPG NEVERWINTER NIGHTS (NWN). Our evaluation experiments aimed at assessing the adaptive performance of a team controlled by the dynamic scripting technique, against a team controlled by static scripts. If dynamic scripting is efficient, the dynamic team will need only a few encounters to design a tactic that outperforms the static team, even if the static team uses a highly effective tactic. In the simulation, we pitted the dynamic team against a static team that would use one of four, manually designed, basic tactics (named "offensive", "disabling", "cursing" and "defensive") or one of three composite tactics (named "random party", "random character" and "consecutive party"). In NWN we pitted the dynamic team against the AI programmed by the developers of the game.

Of all the static team's tactics the most challenging is the consecutive party tactic. With this tactic the static team starts by using one of the four basic tactics. Each encounter the party will continue to use the tactic employed during the previous encounter if that encounter was won, but will switch to the next tactic if that encounter was lost. This strategy is closest to what human players do: they stick with a tactic as long as it works, and switch when it fails.

To quantify the relative performance of the dynamic team against the static team, after each encounter we calculate a so-called "fitness" value for each team. This is a real value in the range [0,1], which indicates how well the team did during the past encounter. It takes into account whether the team won or lost, and, if the team won, the number of surviving team members and their total remaining health. The dynamic team is said to *outperform* the static team at an encounter if the average fitness over the last ten encounters is higher for the dynamic team than for the static team.

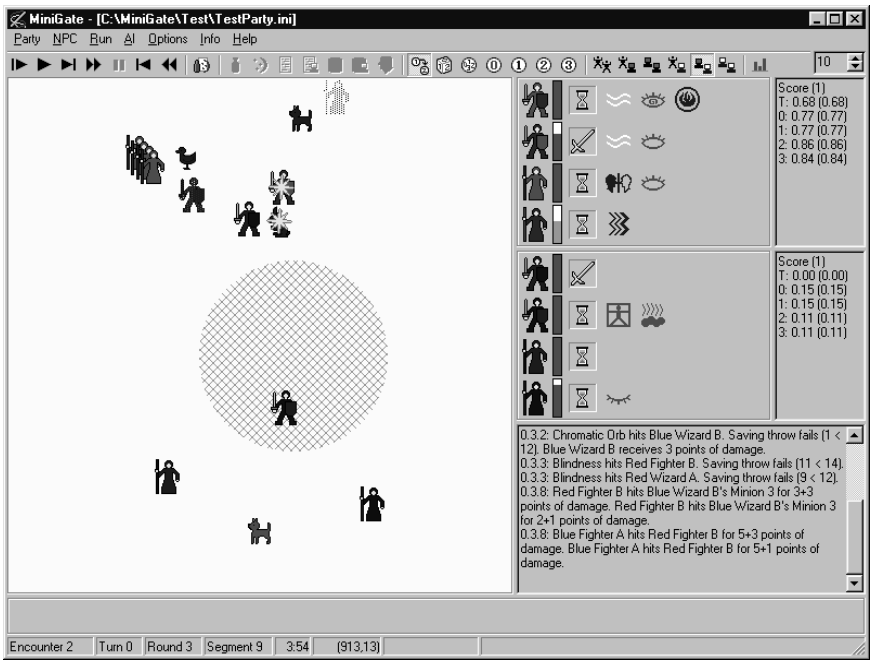

Fig. 2. The simulation environment used to test dynamic scripting

In order to identify reliable changes in strength between parties, we define the *turning point* as the number of the first encounter after which the dynamic team outperforms the static team for at least ten consecutive encounters. A low value for the turning point indicates good efficiency of dynamic scripting, since it shows that the dynamic team consistently outperforms the static team within a few encounters only.

The results of our evaluation experiments are summarised in table 1. Since the opponent AI in NWN was significantly improved between NWN version 1.29 (which we used in earlier research) and version 1.61, turning points have been calculated for both of them. From the results in this table we observe that the turning points achieved are low, especially considering the fact that rulebases started out with equal weights for all rules. We therefore conclude that dynamic scripting is efficient and thus meets all requirements stated in subsection 2.2.

However, from the surprising differences between the average and median values for the turning points, and from the fact that some of the highest turning points found are extremely high, we conclude that, although turning points are low in general, there occasionally are cases where they are too high for comfort. These so-called “outliers” are explained by the high degree of randomness that is inherent to these games. A long run of encounters where pure chance drives the learning process away from an optimum (for instance, a run of encounters wherein the dynamic team is lucky and wins although it employs inferior tactics, or wherein the dynamic team is unlucky and loses although it employs superior tactics) may place the rulebase in a state from which it has difficulty to recover. To resolve the problem of outliers, we investigated two countermeasures, which are discussed in the next section.

Table 1. Turning point values for dynamic scripting pitted against nine different tactics. The columns, from left to right, present the following: (1) the name of the tactic, (2) the number of experiments done with this tactic, (3) the average turning point found, (4) the median turning point found, (5) the standard deviation, (6) the standard error of the mean, (7) the highest turning point found, and (8) the average of the highest five turning points found

Tactic	#exp	$\bar{x}$	median	σ	$\sigma_{\bar{x}}$	highest	$\bar{x}_{top5}$
Offensive	100	58	53	35.0	3.5	314	155
Disabling	100	12	11	5.2	0.5	51	31
Cursing	100	137	35	333.6	33.4	1767	1461
Defensive	100	31	27	18.8	1.9	93	77
Random Party	100	56	34	74.4	7.4	595	310
Random Character	100	53	27	67.0	6.7	398	289
Consecutive Party	100	72	47	100.3	10.0	716	424
NWN AI 1.29	50	21	16	8.8	1.2	101	58
NWN AI 1.61	31	35	32	18.8	3.4	75	65

3 Reducing the Number of Outliers

To reduce the number of outliers, we propose two countermeasures, namely (1) penalty balancing, and (2) history fallback. Subsection 3.1 explains the first countermeasure, and subsection 3.2 the second. The results of the experiments used to test the effectiveness of the countermeasures are presented in subsection 3.3.

3.1 Penalty Balancing

The magnitude of the weight adaptation in a rulebase depends on a measure of the success (or failure) of the opponent whose script is extracted from the rulebase. Typically, the measure of success of an opponent is expressed in the form of an individual fitness function that, besides the team fitness value, incorporates elements of the opponent's individual performance during an encounter. The individual fitness takes a value in the range [0,1]. If the value is higher than a break-even value *b*, the weights of the rules in the script that governed the opponent's behaviour are rewarded, and otherwise they are penalised. The weight adjustment is expressed by the following formula for the new weight value *W*:

$$W = \begin{cases} \max\left(W_{\min}, W_{org} - P_{\max} \cdot \frac{b-F}{b}\right) & \{F < b\} \\ \min\left(W_{org} + R_{\max} \cdot \frac{F-b}{1-b}, W_{\max}\right) & \{F \geq b\} \end{cases} \quad (1)$$

where W_{org} is the original weight value, W_{min} and W_{max} respectively are the minimum and maximum weight values, R_{max} and P_{max} respectively are the maximum reward and penalty, F is the individual fitness, and b is the break-even value.

Penalty balancing is tuning the magnitude of the maximum penalty in relation to the maximum reward to optimise speed and effectiveness of the adaptation process. The experimental results presented in section 2 relied on a maximum reward that was substantially larger than the maximum penalty (namely, P_{max}=30 for the simulation experiments, and P_{max}=50 for the NWN experiments, while R_{max}=100 for both). The argument for the relatively small maximum penalties is that, as soon as an optimum is found, the rulebase should be protected against degradation. This argument seems to be intuitively correct, since for a local optimum a penalty can be considered equivalent to a mutation as used in an evolutionary learning system, and the effectiveness of a learning system improves if the mutation rate is small in the neighbourhood of an optimum [13]. However, if a sequence of undeserved rewards occurs, the relatively low maximum penalty will have problems reducing the unjustly increased weights. Penalty balancing, whereby P_{max} is brought closer to the value of R_{max}, gives dynamic scripting better chances to recover from undeserved weight increases, at the cost of higher chances to move away from a discovered optimum.

3.2 History Fallback

In the original formulation of dynamic scripting [10], the old weights of the rules in the rulebase are erased when the rulebase adapts. With history fallback all previous weights are retained in so-called "historic rulebases". When learning seems to be stuck in a sequence of rulebases that have inferior performance, it can "fall back" to one of the historic rulebases that seemed to perform better.

However, caution should be taken not to be too eager to fall back to earlier rulebases. The dynamic scripting process has shown to be quite robust and learns from both successes and failures. Returning to an earlier rulebase means losing everything that was learned after that rulebase was generated. Furthermore, an earlier rulebase may have a high fitness due to chance, and returning to it might therefore have an adverse effect. We confirmed the wisdom of this caution by implementing dynamic scripting with an eager history-fallback mechanism in NWN, and found its performance to be much worse than that of dynamic scripting without history fallback. Therefore, any history-fallback mechanism should only be activated when there is a high probability that a truly inferior rulebase is replaced by a truly superior one.

Our implementation of history fallback is as follows. The current rulebase R is used to generate scripts that control the behaviour of an opponent during an encounter. After each encounter i, before the weight updates, all weight values from rulebase R are copied to historic rulebase R_i. With R_i are also stored the individual fitness F_i, the team fitness T_i, and a number representing the so-called "parent" of R_i. The parent of R_i is the historic rulebase whose weights were updated to generate R_i (usually the parent of R_i is R_{i-1}). A rulebase is considered "inferior" when both its own fitness values, and the fitness values of its N immediate ancestors, are low. A rulebase is considered "superior" when both its own fitness values, and the fitness values of its N immediate ancestors, are high. If at encounter i we find that R_i is inferior, and in R_i's ancestry we find a historic rulebase R_j that is superior, the next parent used to generate the current rulebase R will not be R_i but R_j. In our experiments we used N=2.

Though unlikely, with this mechanism it is still possible to fall back to a historic rulebase that, although it seemed to perform well in the past, actually only did so by being lucky. While this will be discovered by the learning process soon enough, we don't want to run the risk of returning to such a rulebase over and over again. We propose two different ways of alleviating this problem. The first is by simply not allowing the mechanism to fall back to a historic rulebase that is "too old", but only allow it to fall back to the last M ancestors (in our experiments we used M=15). We call this "limited distance fallback" (LDF). The second is acknowledging that the individual fitness of a rulebase should not be too different from that of its direct ancestors. By propagating a newly calculated fitness value back through the ancestry of a rulebase, factoring it into the fitness values for those ancestors, a rulebase with a high individual fitness that has children that have low fitness values, will also have its fitness reduced fast. We call this "fitness propagation fallback" (FPF). Both versions of history fallback allow dynamic scripting to recover earlier, well-performing rulebases.

3.3 Experimental Results

To test the effectiveness of penalty balancing and history fallback, we ran a series of experiments in our simulation environment. We decided to use the "consecutive party tactic" as the tactic employed by the static team, since this tactic is the most challenging for dynamic scripting. We compared nine different configurations, namely learning runs using maximum penalties P_{max}=30, P_{max}=70 and P_{max}=100, combined with the use of no fallback (NoF), limited distance fallback (LDF), and fitness propagation fallback (FPF).

We also ran some experiments with NWN. In these experiments we used for the static team the standard AI of NWN version 1.61, and we ran also some experiments using so-called "cursed AI". With cursed AI in 20% of the encounters the game AI deliberately misleads dynamic scripting into awarding high fitness to purely random tactics, and low fitness to tactics that have shown good performance during earlier encounters. We did NWN experiments both with no fallback and fitness propagation fallback. We did not change the maximum penalties since in our original experiments for NWN we already used higher maximum penalties than for the simulation.

Table 2 gives an overview of both the simulation and the NWN experiments. Figure 3 shows histograms of the turning points for each of the series of simulation experiments. From these results we make the following four observations: (1) Penalty balancing is a necessary requirement to reduce the number of outliers. All experiments that have a higher maximum penalty than our original P_{max}=30 reduce the number and magnitude of outliers. (2) If penalty balancing is *not* applied, history fallback seems to have no effect or even an adverse effect. (3) If penalty balancing *is* applied, history fallback has no adverse effect and may actually have a positive effect. (4) In the NWN environment history fallback has little or no effect.

As a final experiment, we applied a combination of penalty balancing with P_{max}=70 and limited distance fallback to all the different tactics available in the simulation

Table 2. Turning point values for dynamic scripting pitted against the consecutive party tactic in the simulation and against the NWN AI 1.61, in different circumstances, which are specified in column 1. Columns 2 to 8 present equal information as in table 1

Situation	#exp	$\bar{x}$	median	σ	$\sigma_{\bar{x}}$	highest	$\bar{x}_{top5}$
Sim, P_{max}=30, NoF	100	72	47	100.3	10.0	716	424
Sim, P_{max}=30, LDF	100	99	49	229.3	22.9	2064	837
Sim, P_{max}=30, FPF	100	80	54	145.0	14.5	971	605
Sim, P_{max}=70, NoF	100	62	44	69.4	6.9	336	301
Sim, P_{max}=70, LDF	100	52	37	56.2	5.6	393	238
Sim, P_{max}=70, FPF	100	60	32	57.3	5.7	391	245
Sim, P_{max}=100, NoF	100	66	59	59.5	6.0	322	246
Sim, P_{max}=100, LDF	100	68	60	56.7	5.7	271	225
Sim, P_{max}=100, FPF	100	57	53	50.6	5.1	331	202
NWN, NoF	31	35	32	18.8	3.4	75	65
NWN, FPF	30	32	24	26.7	4.9	104	71
NWN cursed, NoF	21	33	24	21.8	4.8	92	64
NWN cursed, FPF	21	32	18	28.1	6.1	115	69

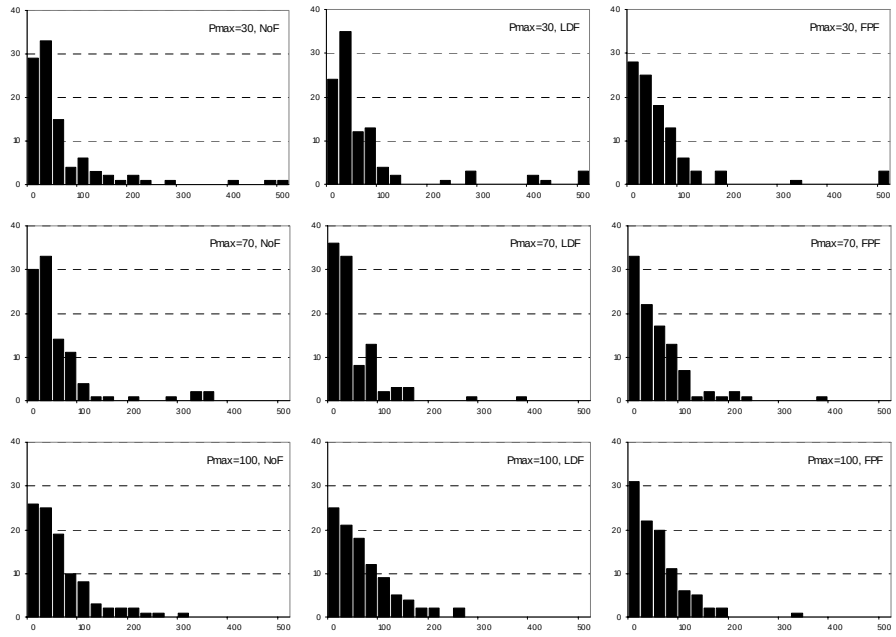

Fig. 3. Histograms of the turning points for the simulation experiments in table 2. The turning points have been grouped in ranges of 25 different values. Each bar indicates the number of turning points falling within a range. Each graph starts with the leftmost bar representing the range [0,24]. The rightmost bars in the topmost three graphs represent all turning points of 500 or greater (the other graphs do not have turning points in this range)

environment. The results are summarised in table 3. A comparison of table 3 and table 1 shows a significant, often very large reduction of the both the highest turning point and the average of the highest five turning points, for all tactics except for the "disabling" tactic (however, the "disabling" tactic already has the lowest turning points in both tables). This clearly confirms the positive effect of the two countermeasures.

Table 3. Turning point values for dynamic scripting pitted against different tactics, using P_{max}=70 and limited distance fallback. The columns present equal information as in table 1

Tactic	#exp	$\bar{x}$	median	σ	$\sigma_{\bar{x}}$	highest	$\bar{x}_{top5}$
Offensive	100	53	52	24.8	2.5	120	107
Disabling	100	13	11	8.4	0.8	79	39
Cursing	100	44	26	50.4	5.0	304	222
Defensive	100	24	17	15.3	1.5	79	67
Random Party	100	51	29	64.5	6.5	480	271
Random Character	100	41	25	40.7	4.1	251	178
Consecutive Party	100	52	37	56.2	5.6	393	238

4 Discussion

In this section we discuss the results presented in the previous section. Subsection 4.1 examines the experimental results obtained using the countermeasures. Subsection 4.2 discusses the usefulness of dynamic scripting enhanced with the countermeasures.

4.2 Interpretation of the results

The results presented in table 2 indicate that penalty balancing has an undeniable positive influence on dynamic scripting, especially in reducing the number of outliers. In combination with penalty balancing, history fallback can have an extra positive impact. A qualitative explanation of the history fallback effect is the following. In subsection 3.1 we stated that penalty balancing runs the risk of losing a discovered optimum due to chance. History fallback counteracts this risk, and may therefore improve dynamic scripting even further.

In the NWN environment we observed that history fallback had little or no effect. This may be due to the following three reasons. (1) The effect of history fallback is small compared to the effect of penalty balancing. (2) Since even static opponents that use cursed AI do not cause significantly increased turning points, it seems that dynamic scripting in NWN is so robust that remote outliers do not occur, and therefore countermeasures are not needed. (3) Dynamic scripting in the NWN environment has two extra enhancements compared to the implementation in the simulation, namely the ability to decrease script length, and a rulebase that contains more general tactics as rules. These enhancements may also reduce the occurrence of outliers.

4.2 Usefulness

It is clear from the results in table 2 that the number of outliers has been significantly reduced with the proposed countermeasures. However, occasionally exceptionally long learning runs still occur in the simulation experiments, even though they are rare. Does this mean that dynamic scripting needs to be improved even more before it can be applied in a commercial game?

We argue that it does not. Dynamic scripting is ready to be applied in commercial games. Our argument is twofold. (1) Because dynamic scripting is a non-deterministic technique, outliers can never be prevented completely. However, entertainment value of a game is guaranteed even if an outlier occurs, because of the domain knowledge in the rulebase (this is the requirement of effectiveness from subsection 2.2). (2) Exceptionally long learning runs mainly occur because early in the process chance increases the wrong weights. This is not likely to happen in a rulebase with pre-initialised weights. When dynamic scripting is implemented in an actual game, the weights in the rulebase will not all start out with equal values, but they will be initialised to values that are already optimised against commonly used tactics. This will not only prevent the occurrence of outliers, but also increase the speed of weight optimisation, and provide history fallback with a likely candidate for a superior rulebase.

We note that, besides as a target for the history fallback mechanism, historic rulebases can also be used to store tactics that work well against a specific tactic employed by a human player. If human player tactics can be identified, these rulebases can simply be reloaded when the player starts to use a particular tactic again after having employed a completely different tactic for a while.

5 Conclusion and Future Work

Dynamic scripting is a technique that realises unsupervised online adaptation of opponent AI in complex commercial computer games such as CRPGs. It is based on the automatic online generation of AI scripts for computer game opponents by means of an adaptive rulebase. Although dynamic scripting has been shown to perform well, exceptionally long learning runs ("outliers") tend to occur occasionally. In this paper we investigated two countermeasures against the outliers, namely penalty balancing and history fallback. We found that penalty balancing has a significant positive effect on the occurrence of outliers, and that history fallback may improve the effect of penalty balancing even further. We conclude that the performance of dynamic scripting is enhanced by these two countermeasures, and that dynamic scripting can be successfully incorporated in commercial games.

Our future work aims at applying dynamic scripting in other game types than CRPGs, such as Real-Time Strategy games. We will also investigate whether offline machine learning techniques, which can be very effective in designing tactics [14], can be used to "invent" completely new rules for the dynamic scripting rulebase. Finally, since our main aim is to use online learning against human players, it is essential that we extend our experiments to assess if online learning actually increases the enter-

tainment value of a game for human players. After all, for commercial game developers entertainment value is of primary concern when deciding whether or not to incorporate online learning in their games.

References

1. Tozour, P.: The Evolution of Game AI. In: Rabin, S. (ed.): AI Game Programming Wisdom. Charles River Media (2002) 3–15
2. Schaeffer, J.: A Gamut of Games. In: AI Magazine, Vol. 22, No. 3 (2001) 29–46
3. Scott, B.: The Illusion of Intelligence. In: Rabin, S. (ed.): AI Game Programming Wisdom. Charles River Media (2002) 16–20
4. Tozour, P.: The Perils of AI Scripting. In: Rabin, S. (ed.): AI Game Programming Wisdom. Charles River Media (2002) 541–547
5. Brockington, M. and Darrah, M.: How *Not* to Implement a Basic Scripting Language. In: Rabin, S. (ed.): AI Game Programming Wisdom. Charles River Media (2002) 548–554
6. Evans, R.: Varieties of Learning. In: Rabin, S. (ed.): AI Game Programming Wisdom. Charles River Media (2002) 567–578
7. Woodcock, S.: Game AI: The State of the Industry. In: Game Developer Magazine, August (2002)
8. Demasi, P. and Cruz, A.J. de O.: Online Coevolution for Action Games. In: Gough, N. and Mehdi, Q. (eds.): International Journal of Intelligent Games and Simulation, Vol. 2, No. 2 (2003) 80–88
9. Demasi, P. and Cruz, A.J. de O.: Anticipating Opponent Behaviour Using Sequential Prediction and Real-Time Fuzzy Rule Learning. In: Mehdi, Q., Gough, N. and Natkin, S. (eds.): Proceedings of the 4th International Conference on Intelligent Games and Simulation (2003) 101–105
10. Spronck, P., Sprinkhuizen-Kuyper, I. and Postma, E.: Online Adaptation of Game Opponent AI in Simulation and in Practice. In: Mehdi, Q., Gough, N. and Natkin, S. (eds.): Proceedings of the 4th International Conference on Intelligent Games and Simulation (2003) 93–100
11. Russell, S. and Norvig, P.: Artificial Intelligence: A Modern Approach, Second Edition. Prentice Hall, Englewood Cliffs, New Jersey (2002)
12. Manslow, J.: Learning and Adaptation. In: Rabin, S. (ed.): AI Game Programming Wisdom. Charles River Media (2002) 557–566
13. Bäck, T.: Evolutionary Algorithms in Theory and Practice. Oxford University Press, New York (1996)
14. Spronck, P., Sprinkhuizen-Kuyper, I. and Postma, E.: Improving Opponent Intelligence Through Offline Evolutionary Learning. In: International Journal of Intelligent Games and Simulation, Vol. 2, No. 1 (2003) 20–27