A Tutoring System for Commercial Games

Pieter Spronck and Jaap van den Herik

Universiteit Maastricht, Institute for Knowledge and Agent Technology
{p.spronck,herik}@cs.unimaas.nl

Abstract In computer games, tutoring systems are used for two purposes: (1) to introduce a human player to the mechanics of a game, and (2) to ensure that the computer plays the game at a level of playing strength that is appropriate for the skills of a novice human player. Regarding the second purpose, the issue is not to produce occasionally a weak move (i.e., a give-away move) so that the human player can win, but rather to produce not-so-strong moves under the proviso that, on a balance of probabilities, they should go unnoticed. This paper focuses on using adaptive game AI to implement a tutoring system for commercial games.[1] We depart from the novel learning technique 'dynamic scripting' and add three straightforward enhancements to achieve an 'even game', viz. high-fitness penalising, weight clipping, and top culling. Experimental results indicate that top culling is particularly successful in creating an even game. Hence, our conclusion is that dynamic scripting with top culling can implement a successful tutoring system for commercial games.

1 Introduction

In computer games, tutoring systems are used for two purposes: (1) to introduce a human player to the mechanics of a game, and (2) to ensure that the computer plays the game at a level of playing strength that is appropriate for the skills of a novice human player. In our view, an 'appropriate' playing strength entails that the computer manages to play an 'even game' against the human player, i.e., a game where both players have an equal chance to win. Of course, winning a game is not a matter of chance, but a matter of applying strategies, and strategies are to be chosen at will by the players involved. To ensure that the game remains interesting, the issue is not for the computer to produce occasionally a weak move (i.e., a give-away move) so that the human player can win, but rather to produce not-so-strong moves under the proviso that, on a balance of probabilities, they should go unnoticed [1]. We refer to the automatic adaptation of the computer's playing strength to the skills of the human player as 'difficulty scaling'. Our present research investigates the second purpose of tutoring systems, i.e., the scaling of a game's difficulty level so that the computer plays an even game against even a novice human player. Our focus is on commercial games.

[1] All software discussed in this paper can be downloaded from the first author's website: http://www.cs.unimaas.nl/p.spronck

Especially for complex commercial games, such as Computer RolePlaying Games (CRPGs) and strategy games, where for every move the human player can choose between hundreds of different actions, tutoring systems are a necessity. When available, a tutoring system usually consists of two parts: (1) one or more 'introductory levels', to make the human player familiar with the game's mechanics, and (2) a 'difficulty setting', a discrete value that allows the human player to determine at what level of difficulty the game will be played. While the state of the art for introductory levels is of high quality, the difficulty setting commonly has some seriously challenging issues.

We indicate three different issues with the difficulty setting in games. First, the setting is *coarse*, with the player having a choice between only a limited number of difficulty levels (usually three or four). Second, the setting is *player-selected*, with the player unable to assess which difficulty level is appropriate for his skills. Third, the setting has a *limited scope*, (in general) only affecting the computer-controlled opponents' strength, and not their strategies. Consequently, even on a 'high' difficulty setting, the opponents exhibit similar behaviour as on a 'low' difficulty setting, despite their greater strength.

We propose to alleviate the three issues mentioned above by replacing the 'difficulty setting' with a tutoring system consisting of adaptive game AI and an adequate difficulty-scaling mechanism. Adaptive game AI changes the computer's strategies to the way a game is played. As such, (1) it makes changes in small steps (i.e., it is not coarse), (2) it makes changes automatically (i.e., it is not player-selected), and (3) it affects the computer's strategies (i.e., it does not have a limited scope). With difficulty scaling, the changes made by the adaptive game AI can be tuned to the human player's skills, effectively enticing an even game at all times. We demonstrate the viability of our proposal by enhancing the online adaptive game AI technique 'dynamic scripting' with difficulty-scaling enhancements, and empirically validating the effectiveness of the resulting tutoring system in a simulated CRPG.

The outline of the remainder of the paper is as follows. Section 2 provides background information on tutoring systems and adaptive game AI. Section 3 describes dynamic scripting. Section 4 deals with three difficulty-scaling enhancements to dynamic scripting. Section 5 presents the experimental results obtained from applying dynamic scripting with difficulty scaling in a simulated CRPG. Section 6 discusses the results. In Section 7, the paper concludes and points at future work.

2 Tutoring Systems and Adaptive Game AI

In analytical computer games, an interesting domain of research is online adapting strategies, i.e., strategies that adapt and learn automatically (unsupervised) while the game is being played. The application areas are (1) learning from the computer (i.e., tutoring systems), (2) teaching the computer, and (3) providing human players with sufficient entertainment that they enjoy the game. For commercial games, 'online adapting strategies' are generally referred to as 'adap-

tive game AI'. We believe that adaptive game AI is a prerequisite for successful commercial games [2].

In the domain of analytical two-player games such as SHOGI, CHESS, and CHECKERS we have seen many learning systems, but not so many online learning systems (apart from opening books). There is an interesting branch of opponent-model search [3] that might suit our research aim; however, in general opponent-modelling techniques are applied offline. Early ideas on tutoring strategies in game-tree search can be ascribed to Iida, Handa, and Uiterwijk [1], with their introduction of loss-oriented search (LO search), that is used to produce an even game. Iida *et al.* [1] acknowledged that their model is possibly too detailed to be realistic, and rather naively replaced the stochastic quality by a numerical value. Yet, the first ideas are there, even though they are based on an idealised opponent.

For analytical games, tutoring systems are based on adding adaptive game AI in minimax search and opponent-model search. So far, most commercial games do not rely on such advanced AI techniques [4]. Consequently, there is only a small basis in the game AI of commercial games to apply our ideas of difficulty scaling to.

The implementation of online adaptive game AI is widely disregarded by commercial game developers [4,5], even though it has been shown to be feasible for simple games [6]. Recently, Spronck, Sprinkhuizen-Kuyper, and Postma [2] introduced a set of four computational requirements for online adaptive game AI to be successful in commercial games. The four requirements are (1) speed, (2) effectiveness, (3) robustness, and (4) efficiency. Moreover, Spronck *et al.* [2] developed an online learning technique that meets the four requirements, called 'dynamic scripting'. Dynamic scripting is a straightforward technique, but nevertheless the first of its kind. It supports online adaptive game AI in an intuitive way. Dynamic scripting is the basis for our enhancements to incorporate difficulty scaling in commercial games.

3 Dynamic Scripting

In this section we present dynamic scripting as a technique that is designed for the implementation of online adaptive game AI in commercial games (henceforth called 'games'). Those interested in a more detailed exposition of dynamic scripting are referred to [2].

Dynamic scripting is an unsupervised online learning technique for 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 controlling the opponent are extracted from the corresponding rulebase. The probability that a rule is selected for a script is proportional to the weight value that is associated with the rule. The rulebase adapts by changing the weight values to reflect the success or failure rate of the associated rules in scripts. A priority mechanism can be used to let certain rules take precedence over other

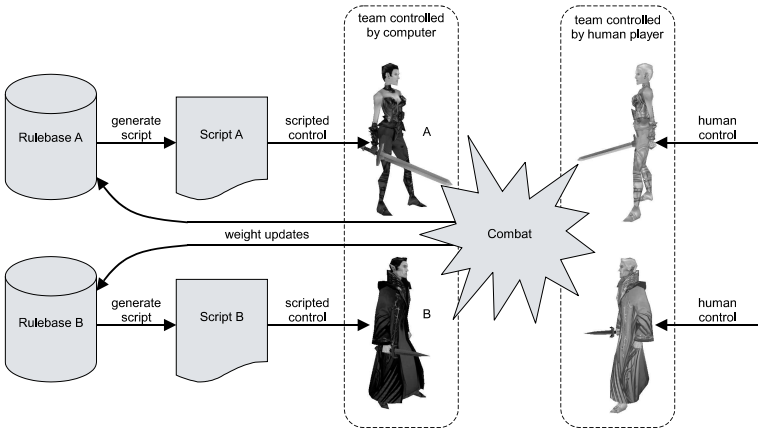

Figure 1. Dynamic scripting.

rules. The dynamic scripting process is illustrated in Figure 1 in the context of a game.

The learning mechanism of dynamic scripting is inspired by reinforcement-learning techniques [7,8]. 'Regular' reinforcement learning techniques, such as TD-learning, in general need large amounts of trials, and thus do not meet the requirement of efficiency [9,10]. Reinforcement learning may be suitable for online learning of game AI when the trials occur in a short time-span. Such may be the case on an operational level of intelligence, as in, for instance, the work by Graepel, Herbrich, and Gold [11], where fight movements in a fighting game are learned. However, for the learning on a tactical or strategic level of intelligence, a trial consists of observing the performance of a tactic over a fairly long period of time. Therefore, for the online learning of tactics in a game, reinforcement learning will take too long to be particularly suitable. In contrast, dynamic scripting has been designed to learn from a few trails only.

For the rules, weight values are bounded by a range $[W_{min}, W_{max}]$. The size of the weight changes depends on how well, or how badly, a team member behaved during the encounter. It is determined by a fitness function that rates a team member's performance as a number in the range $[0, 1]$. The fitness function is composed of four indicators of playing strength, namely (1) whether the member's team won or lost, (2) whether the member died or survived, (3) the member's remaining health, and (4) the amount of damage done to the member's enemies. The new weight value is calculated as $W + \triangle W$, where W is the original weight value, and the weight adjustment $\triangle W$ is expressed by the following equation:

$$\triangle W = \begin{cases} -\lfloor P_{max} \dfrac{b - F}{b} \rfloor & \{F < b\} \\ \lfloor R_{max} \dfrac{F - b}{1 - b} \rfloor & \{F \geq b\} \end{cases} \quad (1)$$

In Equation 1, $R_{max} \in \mathbb{N}$ and $P_{max} \in \mathbb{N}$ are the maximum reward and maximum penalty respectively, F is the agent fitness, and $b \in \langle 0, 1 \rangle$ is the break-even value. At the break-even point the weights remain unchanged.

4 Difficulty Scaling

This section describes how dynamic scripting can be used to create new opponent strategies while scaling the difficulty level of the game AI to the experience level of the human player. Specifically, it describes three different enhancements to the dynamic scripting technique that let opponents learn how to play an even game, namely (1) high-fitness penalising in Subsection 4.1, (2) weight clipping in Subsection 4.2, and (3) top culling in Subsection 4.3. These enhancements have been discussed before by Spronck, Sprinkhuizen-Kuyper, and Postma [12].

4.1 High-Fitness Penalising

The weight adjustment expressed in Equation 1 gives rewards proportional to the fitness value: the higher the fitness, the higher the reward. To elicit mediocre instead of superior behaviour, the weight adjustment can be changed to give highest rewards to mediocre fitness values, and lower rewards or even penalties to high fitness values. With high-fitness penalising weight adjustment is expressed by Equation 1, where F is replaced by F' defined as follows.

$$F' = \begin{cases} \dfrac{F}{p} & \{F \leq p\} \\ \dfrac{1-F}{p} & \{F > p\} \end{cases} \tag{2}$$

In Equation 2, F is the calculated fitness value, and $p \in [0.5, 1]$, $p > b$, is the reward-peak value, i.e., the fitness value that should get the highest reward. The higher the value of p, the more effective opponent behaviour will be. Figure 2 illustrates the weight adjustment as a function of the original fitness (left) and the high-fitness-penalising fitness (right), with the mapping of F to F' in between. Angles α and β are equal.

Since the optimal value for p depends on the strategy that the human player uses, we decided to let the value of p adapt to the perceived difficulty level of a game, as follows. Initially p starts at a value p_{init}. After every fight that is lost by the computer, p is increased by a small amount p_{inc}, up to a predefined maximum p_{max}. After every fight that is won by the computer, p is decreased by a small amount p_{dec}, down to a predefined minimum p_{min}.

4.2 Weight Clipping

During the weight updates, the maximum weight value W_{max} determines the maximum level of optimisation that a learned strategy can achieve. A high value for W_{max} allows the weights to grow to large values, so that after a while the

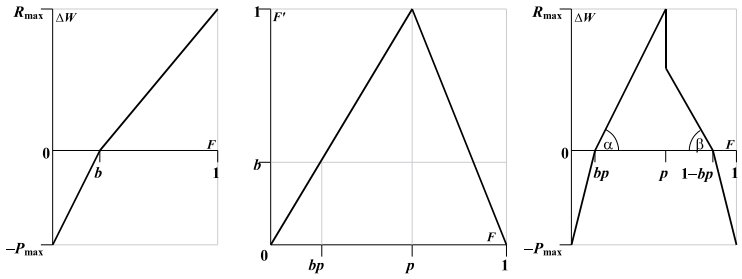

Figure 2. Comparison of the original weight-adjustment formula (left) and the high-fitness-penalising weight-adjustment formula (right), by plotting the weight adjustments as a function of the fitness value F. The middle graph displays the relation between F and F'.

most effective rules will almost always be selected. This will result in scripts that are close to a presumed optimum. A low value for W_{max} restricts weights in their growth. This enforces a high diversity in generated scripts, most of which will not be that good.

Weight clipping automatically changes the value of W_{max}, with the intent to enforce an even game. It aims at having a low value for W_{max} when the computer wins often, and a high value for W_{max} when the computer loses often. The implementation is as follows. After the computer won a fight, W_{max} is decreased by W_{dec} per cent (with a lower limit equal to the initial weight value W_{init}). After the computer lost a fight, W_{max} is increased by W_{inc} per cent.

Figure 3 illustrates the weight-clipping process and the associated parameters. The shaded bars denote weight values for arbitrary rules on the horizontal axis. Before the weight adjustment, W_{max} changes by W_{inc} or W_{dec} per cent, depending on the outcome of the fight. After the weight adjustment, in Figure 3 the weight value for rule 4 is too low, and will be increased to W_{min} (arrow 'a'), while the weight value for rule 2 is too high, and will be decreased to W_{max} (arrow 'b').

4.3 Top Culling

Top culling is quite similar to weight clipping. It employs the same adaptation mechanism for the value of W_{max}. The difference is that top culling allows weights to grow beyond the value of W_{max}. However, rules with a weight greater than W_{max} will not be selected for a generated script. Consequently, when the computer-controlled opponents win often, the most effective rules will have weights that exceed W_{max}, and cannot be selected, and thus the opponents will use relatively weak strategies. Alternatively, when the computer-controlled opponents lose often, rules with high weights will be selectable, and the opponents will use relatively strong strategies.

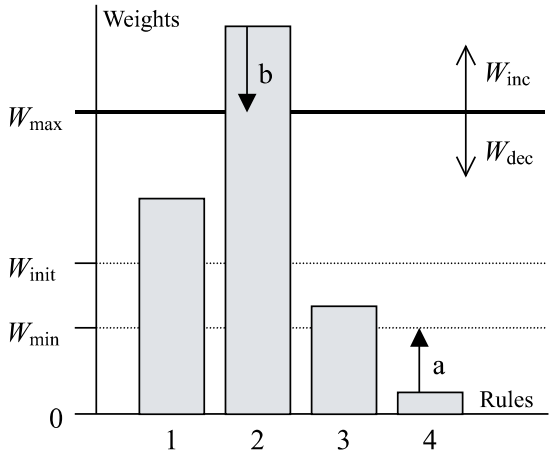

Figure 3. Weight clipping and top culling process and parameters.

In Figure 3, contrary to weight clipping, top culling will leave the value of rule 2 unchanged (the action represented by arrow 'b' will not be performed). However, rule 2 will be unavailable for selection, because its value exceeds W_{max}.

5 Experimental Results

To evaluate the effect of the three difficulty-scaling enhancements to dynamic scripting, we employed a simulation of an encounter of two teams in a complex Computer RolePlaying Game (CRPG), closely resembling the popular Baldur's Gate games. We used this environment in earlier research to demonstrate the efficiency of dynamic scripting [2]. Our evaluation experiments aimed at assessing the performance of a team controlled by the dynamic scripting technique using a difficulty-scaling enhancement, against a team controlled by static scripts. If the difficulty-scaling enhancements work as intended, dynamic scripting will balance the game so that the number of wins of the dynamic team is roughly equal to the number of losses. In the simulation, we pitted the dynamic team against a static team that uses one of five, manually designed, basic strategies (named 'offensive', 'disabling', 'cursing', 'defensive', and 'novice'), or one of three composite strategies (named 'random team', 'random agent' and 'consecutive').

Of the eight static team's strategies the most interesting in the present context is the 'novice' strategy. This strategy resembles the playing style of a novice Baldur's Gate player (for whom a tutoring system is most needed). While the 'novice' strategy normally will not be defeated by arbitrarily picking rules from the rulebase, many different strategies exist that can be employed to defeat it, which the dynamic team will quickly discover. Without difficulty-scaling, the dynamic team's number of wins will greatly exceed its losses.

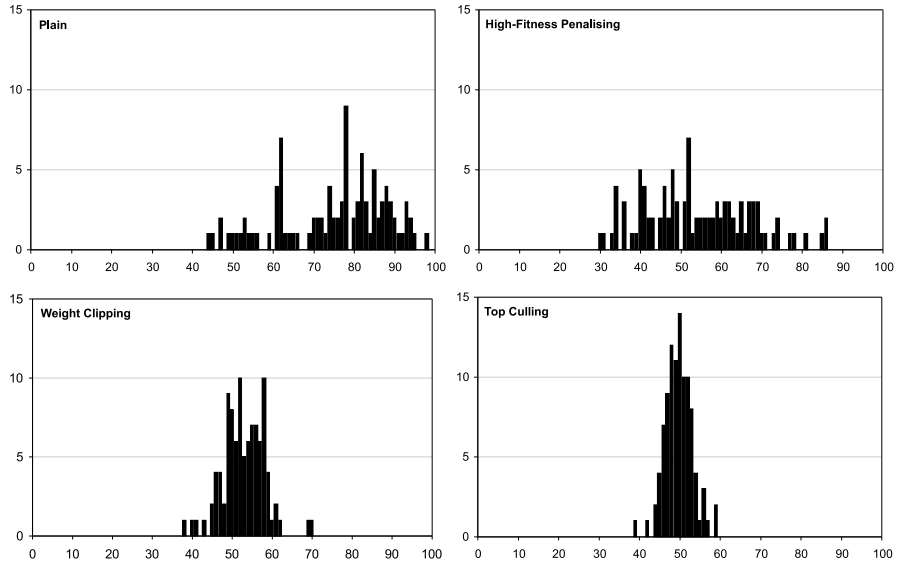

Figure 4. Histograms of 100 tests of the achieved number of wins in 100 fights, against the 'novice' strategy.

In our experiments we initialised $W_{max} = 2000$. We set $W_{init} = 100$, $W_{min} = 0$, $W_{inc} = W_{dec} = 10\%$, $p_{init} = 0.7$, $p_{min} = 0.65$, $p_{max} = 0.75$, $p_{inc} = p_{dec} = 0.01$, $R_{max} = P_{max} = 100$, and $b = 0.3$. We employed the same fitness function as in previous research [2], and dynamic scripting with fitness-propagation fallback [13].

For each of the static strategies, we ran 100 tests in which dynamic scripting was enhanced with each of the three difficulty-scaling enhancements, and, for comparison, also without difficulty-scaling enhancements ('plain'). Each test consisted of a sequence of 150 encounters between the dynamic team and the static team. Because in each of the tests the dynamic scripting process starts with a rulebase with all weights equal, the first 50 encounters were used for finding a balance of well-performing weights (in an actual game, weights would be biased to prefer the best rules from the start, so this 'training period' would not be needed). We recorded the number of wins of the dynamic team for the last 100 encounters. The results of these tests are displayed in Table 1. Histograms for the tests with the 'novice' strategy are displayed in Figure 4. The length of each bar in the histograms indicates the number of tests that resulted in the number of wins (out of 100) that is displayed on the horizontal axis.

To be recognised as an even game, we decided that the average number of wins over all tests must be close to 50. To take into account random fluctuations, in this context "close to 50" means "within the range [45,55]". In Table 1, all

Table 1. Experimental results of testing the difficulty-scaling enhancements to dynamic scripting on eight different strategies, averaged over 100 tests. The results achieved against the 'novice' strategy (marked with an asterisk) are detailed in the histograms in Figure 4.

	Plain		High-Fitness Penalising		Weight Clipping		Top Culling	
Strategy	Average	St.Dev.	Average	St.Dev.	Average	St.Dev.	Average	St.Dev.
Offensive	61.2	16.4	**46.0**	15.1	**50.6**	9.4	**46.3**	7.5
Disabling	86.3	10.4	56.6	8.8	67.8	4.5	**52.2**	3.9
Cursing	56.2	11.7	42.8	9.9	**48.4**	6.9	**46.4**	5.6
Defensive	66.1	11.9	39.7	8.2	**52.7**	4.2	**49.2**	3.6
Novice*	75.1	13.3	**54.2**	13.3	**53.0**	5.4	**49.8**	3.4
Random team	55.8	11.3	37.7	6.5	**50.0**	6.9	**47.4**	5.1
Random agent	58.8	9.7	44.0	8.6	**51.8**	5.9	**48.8**	4.1
Consecutive	**51.1**	11.8	34.4	8.8	**48.7**	7.7	**45.0**	7.3

cell values indicating an even game are marked in bold font. From Table 1 the following four results can be derived.

First, dynamic scripting used without a difficulty-scaling enhancement results in wins significantly exceeding losses for all strategies except for the 'consecutive' strategy (with a reliability $> 99.9\%$ [14]). The 'consecutive' strategy is the most difficult strategy to defeat [2]. Note that the fact that, on average, dynamic scripting plays an even game against the 'consecutive' strategy even without difficulty scaling, is not because it is unable to consistently defeat this strategy, but because dynamic scripting continues learning after it has reached a local optimum. Therefore, it can "forget" what it previously learned, especially against an superior strategy like the 'consecutive' strategy.

Second, high-fitness penalising performs considerably worse than the other two enhancements. It cannot achieve an even game against six of the eight strategies.

Third, weight clipping is successful in enforcing an even game against seven out of eight strategies. It does not succeed against the 'disabling' strategy. This is caused by the fact that the 'disabling' strategy is so easy to defeat, that even a rulebase with all weights equal will, on average, generate a script that defeats this strategy. Weight clipping can never generate a rulebase worse than "all weights equal".

Fourth, top culling is successful in enforcing an even game against all eight strategies.

From the histograms in Figure 4 we derive the following result. While all three difficulty-scaling enhancements manage to, on average, enforce an even game against the 'novice' strategy, the number of wins in each of the tests is much more "spread out" for the high-fitness-penalising enhancement than for

the other two enhancements. This indicates that the high-fitness penalising results in a higher variance of the distribution of won games than the other two enhancements. The top-culling enhancement seems to yield the lowest variance. This is confirmed by an approximate randomisation test [14], which shows that against the 'novice' strategy, the variance achieved with top culling is significantly lower than with the other two enhancements (reliability $> 99.9\%$). We observed similar distributions of won games against the other strategies, except that against some of the stronger strategies a few exceptional outliers occurred with a significantly lower number of won games. The rare outliers were caused by dynamic scripting occasionally needing more than the first 50 encounters to find well-performing weights against a strong static strategy.

6 Discussion

Of the three different difficulty-scaling enhancements we conclude the top-culling enhancement to be the best choice. It has the following three advantages: (1) it yields results with a very low variance, (2) it is easily implemented, and (3) of the three enhancements, it is the only one that manages to force an even game against inferior strategies, which is of crucial importance for tutoring systems.

We further validated the results achieved with top culling, by implementing dynamic scripting with the top-culling enhancement in a state-of-the-art computer game, NEVERWINTER NIGHTS (version 1.61). We tested it against the game AI implemented by the game developers, with the same experimental procedure as used in the simulation environment. Ten tests without difficulty scaling resulted in an average number of wins of 79.4 out of 100, with a standard deviation of 12.7. Ten tests with the top-culling enhancement resulted in an average number of wins of 49.8 out of 100, with a standard deviation of 3.4. Therefore, our simulation results are supported by the NEVERWINTER NIGHTS tests.

Obviously, the worst difficulty-scaling enhancement we tested is high-fitness penalising. In an attempt to improve high-fitness penalising, we performed some tests with different ranges and adaptation values for the reward-peak value p, but these worsened the results. However, we cannot rule out the possibility that with a different fitness function high-fitness penalising will give better results.

An additional possibility with weight clipping and top culling is that they can be used to set a desired win-loss ratio, simply by changing the rates with which the value of W_{max} fluctuates. For instance, by using top culling with W_{dec}=30% instead of 10%, leaving all other parameters the same, after 100 tests against the 'novice' strategy, we derived an average number of wins of 35.0 with a standard deviation of 5.6.

In previous research we concluded that dynamic scripting is suitable to be applied in real commercial games to improve strategies automatically [13]. With a difficulty-scaling enhancement, dynamic scripting becomes a tutoring system for novice players, improving its usefulness significantly.

7 Conclusions and Future Work

In this paper we proposed to implement a tutoring system for commercial games by enhancing online adaptive game AI with difficulty scaling. We demonstrated the viability of our proposal by testing three different enhancements to the online adaptive game AI technique 'dynamic scripting' that allow scaling of the difficulty level of game AI. These three enhancements are (1) high-fitness penalising, (2) weight clipping, and (3) top culling. Of the three difficulty-scaling enhancements tested, top culling gave the best results. We also discovered that both weight clipping and top culling, besides forcing an even game, can be used to set a different win-loss ratio, by tuning a single parameter. We conclude that dynamic scripting, using top culling, can be used as a tutoring system for commercial games.

In future work, we intend to apply dynamic scripting, including difficulty scaling, in other game types than CRPGs. We will also investigate whether offline machine learning techniques can be used to "invent" completely new rules for the dynamic scripting rulebase. First results for this research are reported by Ponsen and Spronck [15]. Finally, we will aim to investigate the effectiveness of the proposed tutoring system in games played against actual human players. While such a study requires many subjects and a careful experimental design, the game-play experiences of human players are important to convince game developers to adopt the proposed tutoring system in their games.

References

1. Iida, H., Handa, K., Uiterwijk, J.: Tutoring strategies in game-tree search. ICCA Journal **18** (1995) 191–204
2. Spronck, P., Sprinkhuizen-Kuyper, I., Postma, E.: Online adaptation of game opponent AI with dynamic scripting. International Journal of Intelligent Games and Simulation **3** (2004) 45–53
3. Donkers, H.: Nosce Hostem: Searching with Opponent Models. Ph.D. thesis. Universitaire Pers Maastricht, Maastricht, The Netherlands (2003)
4. Rabin, S.: Promising game AI techniques. In Rabin, S., ed.: AI Game Programming Wisdom 2, Hingham, MA, Charles River Media, Inc. (2004) 15–27
5. Woodcock, S.: The future of game AI: A personal view. Game Developer Magazine **7** (2000)
6. Demasi, P., Cruz, A.: Online coevolution for action games. International Journal of Intelligent Games and Simulation **2** (2002) 80–88
7. Sutton, R., Barto, A.: Reinforcement Learning: An Introduction. MIT Press, Cambridge, MA (1998)
8. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Second edition edn. Prentice Hall, Pearson Education, Upper Saddle River, NJ (2003)
9. Manslow, J.: Learning and adaptation. In Rabin, S., ed.: AI Game Programming Wisdom, Hingham, MA, Charles River Media, Inc. (2002) 557–566
10. Madeira, C., Corruble, V., Ramalho, G., Ratitch, B.: Bootstrapping the learning process for the semi-automated design of challenging game AI. In Fu, D., Henke, S., Orkin, J., eds.: Proceedings of the AAAI-04 Workshop on Challenges in Game Artificial Intelligence, Menlo Park, CA, AAAI Press (2004) 72–76

11. Graepel, T., Herbrich, R., Gold, J.: Learning to fight. In Mehdi, Q., Gough, N., Natkin, S., Al-Dabass, D., eds.: Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004), Wolverhampton, UK, University of Wolverhampton (2004) 193–200
12. Spronck, P., Sprinkhuizen-Kuyper, I., Postma, E.: Difficulty scaling of game AI. In El Rhalibi, A., van Welden, D., eds.: GAME-ON 2004 5th International Conference on Intelligent Games and Simulation. (2004)
13. Spronck, P., Sprinkhuizen-Kuyper, I., Postma, E.: Enhancing the performance of dynamic scripting in computer games. In Rauterberg, M., ed.: Entertainment Computing – ICEC 2004. Lecture Notes in Computer Science 3166, Berlin, Germany, Springer-Verlag (2004) 296–307
14. Cohen, P.: Empirical Methods for Artificial Intelligence. MIT Press, Cambridge, MA (1995)
15. Ponsen, M., Spronck, P.: Improving adaptive game AI with evolutionary learning. In Mehdi, Q., Gough, N., Natkin, S., Al-Dabass, D., eds.: Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004), Wolverhampton, UK, University of Wolverhampton (2004) 389–396