Automatic Rule Ordering for Dynamic Scripting

Timor Timuri and **Pieter Spronck** and **Jaap van den Herik**
Universiteit Maastricht
Maastricht ICT Competence Centre, Institute for Knowledge and Agent Technology
P.O. Box 616, 6200 MD Maastricht, The Netherlands
p.spronck@micc.unimaas.nl

Abstract

The goal of adaptive game AI is to enhance computer-controlled game-playing agents with (1) the ability to self-correct mistakes, and (2) creativity in responding to new situations. Dynamic scripting is a reinforcement learning technique that realises fast and reliable online adaptation of game AI. It employs knowledge bases which contain rules that can be included in game scripts. To be successful, dynamic scripting requires a mechanism to order the rules that are selected for scripts. So far, rule ordering was achieved by a manually-tuned priority value for each rule. In the present research, we propose three mechanisms to order rules automatically for dynamic scripting. We performed experiments in which we let dynamic scripting, using each of the three mechanisms, play against manually-designed tactics. Our results show that dynamic scripting with automatic rule ordering generates game AI that is at least as effective as dynamic scripting with manually-tuned priority values. Moreover, it has the ability to generate novel game AI with significantly increased effectiveness. The costs are a slight decrease in learning efficiency. So, we may conclude that automatic rule ordering is a valuable enhancement for dynamic scripting.

Introduction

The behaviour of computer-controlled agents in modern computer games is determined by so-called 'game AI'. We define 'adaptive game AI' as game AI that employs unsupervised online learning (i.e., during game-play). Adaptive game AI has two main objectives, namely (1) to enhance the agents with the ability to learn from their mistakes (self-correction), and (2) to enhance the agents with the ability to devise new behaviour in response to previously unconsidered situations (creativity). Although recently academic researchers achieved good results in their exploration of adaptive game AI (Demasi & Cruz 2002; Spronck, Sprinkhuizen-Kuyper, & Postma 2004; Graepel, Herbrich, & Gold 2004), game publishers are still reluctant to release games with online-learning capabilities (Funge 2004). Their main fear is that the agents learn inferior behaviour (Woodcock 2002; Charles & Livingstone 2004). Therefore, the few games that contain online adaptation, only do so in a severely limited sense, in order to run as little risk as possible (Charles & Livingstone 2004).

Dynamic scripting (Spronck, Sprinkhuizen-Kuyper, & Postma 2004; Spronck *et al.* 2006) is a reinforcement learning technique aimed at fast and reliable online adaptation of game AI. It employs knowledge bases with rules that can be part of game scripts. Weights attached to the rules indicate the likelihood that a rule is selected for a script. An update mechanism adapts the weight values to increase the effectiveness of the scripts generated from the knowledge base.

Often, rules in a script are only effective when they occur in a specific order. For instance, in a Role-Playing Game (RPG), a fighter's melee attacks usually are most effective when he 'buffs' himself first. However, if melee attack rules occur in the script before the buffing rules, in effect the fighter will perform his attacks without buffing. Therefore, for an automated learning mechanism, determining the order of the selected rules in the script is as important as determining which rules should be in the script.

In previous work, the ordering of rules occurred either by a manually-tuned priority mechanism (Spronck, Sprinkhuizen-Kuyper, & Postma 2004) or by the learned rule weights, where the rules with the highest weights were executed first (Ponsen & Spronck 2004). Neither of these mechanisms exhibited a satisfactory performance. Therefore we propose to enhance dynamic scripting with an automatic rule-ordering mechanism, so that an appropriate rule ordering is learned at the same time as appropriate rule weights.

The outline of this paper is as follows. First, we describe dynamic scripting and related work. Then, we discuss three mechanisms to generate automatically a successful ordering of the rules in a script, and describe the experiments that we performed in two simulated RPGs. Finally, we discuss the achieved results and provide conclusions.

Dynamic Scripting

In reinforcement learning problems, an adaptive agent interacts with its environment and iteratively learns a policy, based on a scalar reward signal that it receives from the environment (Kaelbling, Littman, & Moore 1996; Sutton & Barto 1998). Dynamic scripting (Spronck, Sprinkhuizen-Kuyper, & Postma 2004; Spronck *et al.* 2006) is a reinforcement learning technique designed for creating adaptive agents for commercial computer games. It employs on-policy value iteration to learn state-action values based

exclusively on a reward signal that is only concerned with maximising immediate rewards. Action selection is implemented with a softmax method (Sutton & Barto 1998). The reward signal is typically designed with prior expert knowledge of how to achieve a certain goal.

In commercial computer games, an opponent is an agent (or a team of agents) which opposes the human player within a game world. In most games, the opponent's behaviour is determined by scripts (Tozour 2002). A script consists of a sequence of rules, which connect a condition to an action. To select an action, the rules in the script are examined sequentially, and in a straightforward design the action of the first rule for which the condition holds true is activated.

The default implementation of dynamic scripting is aimed at learning scripts for opponents. The implementation is as follows. Each opponent type is represented by a knowledge base that contains a list of rules that may be inserted in a game script. Every time a new opponent is placed in the game, the rules that comprise the script controlling the behaviour are extracted from the corresponding knowledge base. Each rule in the knowledge base has a so-called 'rule weight'. The probability for a rule to be selected for a script is proportional to the associated rule weight.

After an encounter (i.e., a fight) between the human player and an opponent, the opponent's knowledge base adapts by changing the rule-weight values in accordance with the success or failure rate of the rules that were activated during the encounter (i.e., rules that occurred in the opponent's script). The new rule weight value is calculated as $W + \Delta W$, where W is the original rule weight value. The weight adjustment ΔW is expressed by the following formula:

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

In Equation 1, R_{max} and P_{max} are the maximum reward and maximum penalty respectively, $b \in \langle 0, 1 \rangle$ is the break-even value, and $F \in [0, 1]$ is the opponent's fitness. The fitness is a relative measure of the opponent's success, which is high for good results (victories) and low for bad results (defeats).

Dynamic scripting has been designed so that adaptive opponents (1) start exploiting knowledge already after a few trials, and (2) explore new knowledge continuously. It allows balancing exploitation and exploration by maintaining a minimum and maximum selection probability for all actions (i.e., rules). Comparable solution methods such as temporal-difference learning or Monte-Carlo learning update state-action values only after they have been executed (Sutton & Barto 1998). In contrast, dynamic scripting updates all state-action values in a specific state through a redistribution process (Spronck *et al.* 2006), so that the sum of the state-action values remains constant. The consequence is that dynamic scripting cannot guarantee convergence. Non-convergence actually is essential for successful use in games, since the learning task continuously changes (e.g., the human player may choose to switch tactics). Dynamic scripting is capable of generating a variety of behaviours, and responding quickly to changing game dynamics.

In its original implementation, the knowledge bases used by dynamic scripting were manually designed, i.e., programmers used their own knowledge of the game engine to create a list of rules which should allow the construction of effective game scripts in a variety of circumstances. A manually-tuned priority mechanism was used to order the rules in a script. In later work, Ponsen & Spronck (2004) indicated that using an offline learning mechanism to design the knowledge bases would, in general, lead to adaptive game AI that has increased effectiveness and efficiency. In their research, rule ordering was effectuated by the learned rule weights: rules with higher weights occurred earlier in the scripts than rules with lower weights (a similar mechanism was used by Dahlbom & Niklasson, 2006). Ponsen & Spronck also showed that this ordering mechanism was flawed, in the sense that certain successful combinations of rules could not be discovered.

Below we propose that rule ordering can be learned automatically, at the same time as the rule weights are learned. We discuss three automatic-ordering mechanisms, followed by a discussion of the experiments executed to test their performance.

Rule ordering

Based on the work of Timuri (2006), we implemented three different automatic rule-ordering mechanisms for dynamic scripting.

The first is *Weight Ordering* (WO). By this mechanism, rules are ordered according to their rule-weight values: the rule with the highest weight will be first in the script, the rule with the next-highest weight second, etc. This mechanism is analogous to the ordering implementation used by Ponsen & Spronck (2004).

The second is *Relation-Weight Ordering* (RWO). By this mechanism, a relation-weights table is maintained during the game. It indicates for each combination of two rules whether they have a beneficial or a detrimental effect on each other, for both possible rule orderings, including the size of the effect. The rules are ordered in such a way that it reflects the highest possible benefit according to the relation-weights table. The relation-weights table will be explained in detail below.

The third is *Relation-Weight Ordering with Selection Bonus* (RWO+B). This mechanism functions similar to RWO, but differs during the selection of rules for a script. During selection, for each selected rule the relation-weights table is used to find the rule with which it has the largest relation weight. If the relation weight is greater than a specific 'threshold value', and the second rule is not yet selected, it will receive a 'selection bonus' to its rule weight so that is has a substantially increased chance to be selected too.

The Relation-Weights Table

The relation-weights table stores two values for each combination of two rules of a knowledge base, one value for each of the two possible orderings of the two rules. The values are an indication for the effect that the rules have on the performance of the script, when they occur in the specified order.

Rule	R_1	R_2	R_3	...	R_N
R_1	-	w_{12}	w_{13}	...	w_{1N}
R_2	w_{21}	-	w_{23}	...	w_{2N}
R_3	w_{31}	w_{32}	-	...	w_{3N}
...	...	...	...	...	...
R_N	w_{N1}	w_{N2}	w_{N3}	...	-

Table 1: Relations-weights table

The relation-weights table is visualised in Table 1. R_i represents rule number i. The knowledge base contains a total of N rules. By $R_i \prec R_j$ we denote that R_i occurs earlier than rule R_j in the script. Moreover, w_{ij} represents the relation weight that indicates the effect of $R_i \prec R_j$.

Usually relation weights w_{ij} and w_{ji} are not equal. Many rules have a 'natural' location in the script; e.g., rules that deal with very specific circumstances should occur earlier in the script than more general rules. Still, it is possible that both relation weights are positive, when both have a beneficial effect regardless of where they occur in the script. In the same vein, it is possible that both are negative, when they always have a detrimental effect.

Updating the Table

After each fight, first the rule weights are updated, followed by the relation-weights table. The same weight-adjustment value ΔW as used for updating the rule weights, is used to update the relation weights. This is effectuated through the following update rule:

for all $R_i, R_j \in S$ **do**
  if $A(R_i), R_i \prec R_j$ **then**
    $w_{ij} \leftarrow w_{ij} + \Delta W$

where S is the generated script, and $A(R_i)$ indicates that rule R_i was executed at least once during the fight. The effect of the update rule is that if the outcome of a fight was positive (i.e., the learning opponent won), ΔW is positive and therefore the relation weights of all rule combinations in the script will be rewarded (as long as the first rule in a combination was actually used). Conversely, if the outcome of a fight was negative (i.e., the learning opponent lost), ΔW is negative and therefore the relation weights will be penalised.

Using the Table

The relation-weights table is used to calculate the rule ordering weight O_m for each rule R_m that is selected for the script, according to the following equation:

$$O_m = \sum_{i=1, R_i \in S}^{N} w_{mi} \quad (2)$$

The rules are ordered in the script according to their ordering weights: the rule with the highest ordering weight comes first in the script, followed by the second-highest, etc. Note that usually this mechanism will automatically order relationships spanning more than two rules effectively. The exception is when the ordering of two rules depends on the presence or absence of a third rule, e.g., when the preferred ordering is $R_i \prec R_j$ when R_k is present in the script, but $R_j \prec R_i$ when R_k is absent. In practice, however, this is a rare occurrence.

Two Experiments

To assess empirically the performance of the proposed rule-ordering mechanisms, we performed two experiments in simulated RPGs, namely (1) a Wizard Battle Experiment, and (2) a Team Battle experiment. In the experiments, a dynamic opponent (i.e., an agent or a team of agents using dynamic scripting) is pitched against a static opponent (i.e., an agent or team of agents using a static, manually-designed tactic). One fight between the two opponents consists of a battle that continues until one of them is defeated. For Equation 1, we use $R_{max} = 100$, $P_{max} = 70$, and $b = 0.3$. For RWO+B, we set the threshold value to 300 and the selection bonus to 200. The function we use to calculate the fitness is the same as used by Spronck *et al.* (2006).

An experiment consists of a series of tests, which are initialised with a particular rule-ordering mechanism and a particular static tactic. We defined four different static tactics for the first experiment, and eight for the second. A test consists of a series of fights, in which the dynamic opponent uses the rule-ordering mechanism, and the static opponent uses the static tactic. At the start of a test, the knowledge bases are initialised with all rule weights set to 100, and, when appropriate, all relation weights set to zero. The test continues for a fixed number of fights, during which the dynamic opponent learns to become more effective than the static opponent.

The performance of the dynamic opponent during a test is expressed by two measures. The first measure is the 'turning point', which is the number of the fight after which the average fitness over the last ten fights is higher for the dynamic opponent than for the static opponent, and remains higher for at least the next ten fights. Thus, the turning point gives an indication for the efficiency of the learning process (the lower the turning point, the higher the efficiency). The second measure is the 'number of wins', which is the total number of victories of the dynamic opponent in the last 100 fights of the test. Thus, the second measure gives an indication of the achieved level of effectiveness of the dynamic opponent (the higher the number of victories, the higher the effectiveness). Each test is repeated 50 times, and the averages for the two measures are reported.

Wizard Battle Experiment

The first experiment concerns a battle between two high-level wizards (magic users), based on the implementation of 15th-level wizards in the RPG *Neverwinter Nights*. The knowledge base for a wizard contains 40 rules, most of which consist of simply the casting of a specific magic spell (a complete list is given by Timuri (2006)). A script consists of a selection of ten of these 40 rules. Each spell can only be cast once during a fight. For rule ordering, we employed the three different mechanisms discussed earlier in this paper (WO, RWO, and RWO+B).

For the static opponent, we devised four different tactics, namely the following.

	Best Spell		Offensive		Disabling		Combined	
	TP	Wins	TP	Wins	TP	Wins	TP	Wins
WO	138 (112)	45 (23)	125 (102)	48 (20)	39 (36)	54 (8)	82 (62)	59 (10)
RWO	148 (104)	60 (24)	140 (122)	65 (21)	42 (37)	55 (11)	87 (57)	66 (12)
RWO+B	153 (135)	68 (23)	143 (106)	70 (23)	43 (41)	63 (9)	102 (88)	71 (13)

Table 2: Results of the Wizard Battle experiment.

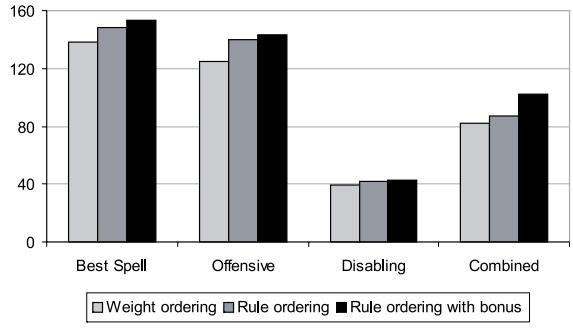

Figure 1: Average turning points achieved in the Wizard Battle experiment.

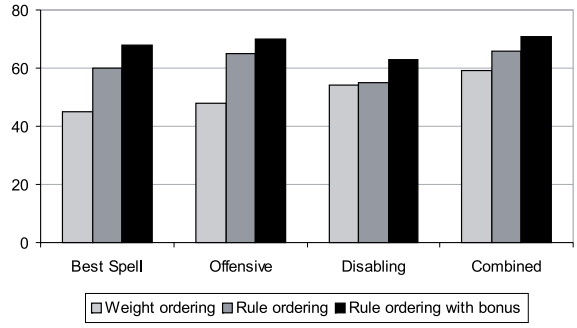

Figure 2: Average wins achieved in the Wizard Battle experiment.

1. *Best Spell*: The agent always casts a spell of the highest level available. If there are multiple spells available of the highest level, one is chosen at random.
2. *Offensive*: The agent uses only damaging spells. It starts with the highest-level damaging spell, and works its way downwards. When it runs out of spells, it uses a weapon.
3. *Disabling*: The agent attempts to incapacitate its opponent by using disabling spells. When it runs out of disabling spells, it switches to the offensive tactic.
4. *Combined*: The agent chooses one of the first three tactics at random for each fight.

The experiment comprised twelve tests, combining each of the three rule-ordering mechanisms for the dynamic opponent, with each of the four tactics for the static opponent. Each test consisted of 500 fights, and was repeated 50 times. The resulting averages of the turning points (TP) and the victories of the last 100 fights (Wins) are presented in Table 2 (with the standard deviation for each result between brackets). They are visually represented in Figures 1 and 2. From these results we make the following four observations.

First, we see that the efficiency (derived from the turning points) of the weight-ordering (WO) mechanism seems to be the best of all tests, while the relation-weights ordering with selection bonus (RWO+B) performs worst in this respect. However, the differences are not statistically significant.

Second, we see that the effectiveness (derived from the number of wins) achieved by WO seems to be the worst in all tests, while RWO+B performs best. This result is statistically significant for all of the differences between WO and RWO+B, and for some of them between RWO and RWO+B.

Third, we point out that while WO has a high efficiency, the resulting effectiveness for the Best-Spell and the Offensive tactics are unsatisfactory, as they are less than 50%.

Fourth, we note that all rule-ordering mechanisms achieve a very high efficiency against the disabling tactic. The turning points are low because, on average, fights against the disabling tactic take longer than fights against the other tactics. Therefore, they generate more updates in the knowledge base and in the relation-weights table per fight.

Team Battle Experiment

The second experiment concerns a battle between two low-level teams. The goals of this experiment were (1) to determine whether the results achieved with individual opponents would transfer to opposing teams, and (2) to compare the effectiveness of learned rule orderings with manually-tuned rule orderings. The teams consisted each of two wizards and two fighters, based on the implementation of 5th-level characters in the RPG *Baldur's Gate*. The knowledge base for the wizard type contains 50 rules, and for the fighter type 20 rules. (a complete list is given by Spronck (2005)). A script consists of ten rules for a wizard, and five rules for a fighter.

We did not test WO further, because the previous experiment demonstrated its inferiority decisively. So, for rule ordering, we employed three different mechanisms, namely the following: (1) Priority-based Ordering (PO), where rules are ordered according to manually-tuned priority values (i.e., as rules were ordered in the original implementation of dynamic scripting), (2) the previously-discussed RWO, and (3) the previously-discussed RWO+B. For the static opponent, we used eight different tactics, namely the following.

1. *Offensive* (Off): The agents focus on damaging their opponents.

	Off		Dis		Curse		Def	
	TP	Wins	TP	Wins	TP	Wins	TP	Wins
PO	50 (21)	73 (13)	11 (2)	89 (12)	42 (46)	61 (13)	23 (20)	74 (14)
RWO	71 (42)	79 (14)	13 (7)	74 (15)	51 (52)	67 (13)	20 (12)	75 (13)
RWO+B	71 (39)	81 (12)	14 (9)	78 (13)	39 (34)	62 (14)	23 (17)	74 (13)
	Begin		RP		RC		CON	
	TP	Wins	TP	Wins	TP	Wins	TP	Wins
PO	19 (11)	79 (11)	32 (33)	58 (11)	32 (30)	66 (12)	51 (42)	61 (14)
RWO	43 (36)	67 (14)	48 (48)	61 (15)	42 (33)	64 (13)	64 (44)	65 (12)
RWO+B	45 (44)	64 (16)	54 (62)	59 (16)	56 (49)	62 (13)	61 (42)	58 (15)

Table 3: Results of the Team Battle experiment

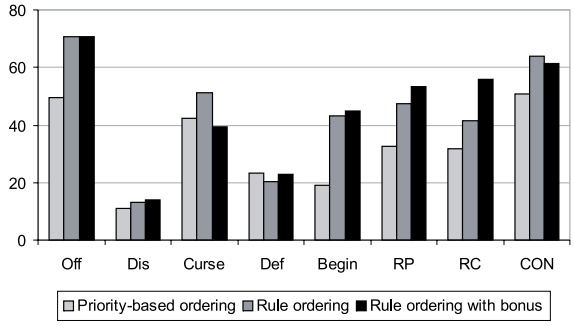

Figure 3: Average turning points achieved in the Team Battle experiment.

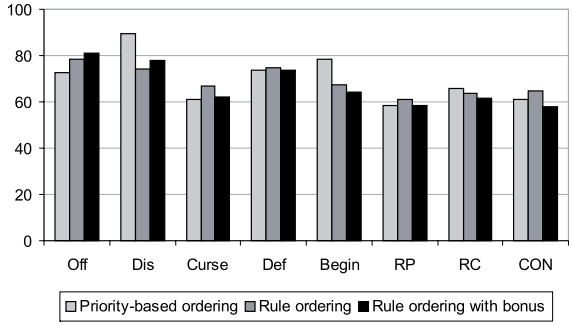

Figure 4: Average wins achieved in the Team Battle experiment.

2. *Disabling* (Dis): The wizard agents focus on first disabling their opponents, before damaging them.
3. *Cursing* (Curse): The wizard agents use a mixture of magic spells to hinder the opponents.
4. *Defensive* (Def): The wizard agents focus on deflecting damage from themselves and their comrades.
5. *Beginner* (Begin): The wizard agents mainly use random spells.
6. *Random Party* (RP): The agents choose one of the first four tactics at random for each fight.
7. *Random Character* (RC): Each of the agents chooses his own tactic from the first four tactics for each fight.
8. *Consecutive* (CON): The agents use the first four tactics in sequence, switching to the next tactic only when they lose a fight.

The experiment comprised 24 tests, combining each of the three rule-ordering mechanisms for the dynamic team, with each of the eight tactics for the static team. Each test consisted of 250 fights, and was repeated 50 times. The resulting averages of the turning points (TP) and the victories of the last 100 fights (Wins) are presented in Table 3 (with the standard deviation for each result between brackets). They are visually represented in Figures 3 and 4. We found that in less than 1% of the fights no turning point was reached after a sequence of 250 fights. In these cases, the turning point was set to 250. The influence of this choice on the average turning points is negligible. From the results we make the following two observations.

First, we see that the efficiency (derived from the turning points) of the priority-based ordering (PO) mechanism is the best in six out of eight tests, and not significantly different from the other results in the remaining two. The explanation is that PO needs to update the least number of weights.

Second, we see that the effectiveness (derived from the number of wins) achieved in most cases does not differ significantly between the mechanisms. Only in the cases of the Disabling tactic and the Beginner tactic, PO leads to significantly better results than the other two mechanisms. However, these two static tactics are the weakest of all, and are very easy to defeat, even without learning, by the supplied knowledge bases with the manually-tuned priorities. Therefore we may conclude that when learning needs to take place to defeat a static tactic, the two mechanisms which automatically learn a rule ordering manage to generate tactics as effective as those generated with manually-tuned priorities.

Effectiveness and Efficiency

In this paper, we showed that a knowledge base with an automatically-determined rule ordering can reach at least the same level of effectiveness as a knowledge base with manually-tuned priority values. Furthermore, we point out that if two rules R_a and R_b exist, which need to be in the

order $R_a \prec R_b$ in one set of circumstances, and in the order $R_b \prec R_a$ in another, automatic rule ordering will be able to generate the appropriate rule ordering for each of these circumstances, while manually-tuned priority values can place the rules in an order appropriate for only one of them.

We also showed that automatic rule ordering seems to be slightly less efficient than manually-tuned rule ordering. We offer the following calculations on the efficiency.

With a knowledge base that contains N rules, without a relation-weights table N rule weights must be tuned. When a script contains M rules, after a fight (at most) M rule weights can be updated in the knowledge base. To update all the rule weights U_1 times, $\frac{NU_1}{M}$ fights are needed.

The size of the relation-weights table is $N(N-1)$. After a fight, $\frac{1}{2}M(M-1)$ relation weights can be updated. To update all the relation weights in the table U_2 times, $\frac{N(N-1)U_2}{\frac{1}{2}M(M-1)}$ fights are needed.

If the number of required fights to update the relation-weights table is less than the number of fights needed to update the rule weights, dynamic scripting with automatic rule ordering will, in general, need about the same number of fights as regular dynamic scripting (since the rule weights must be updated in both cases). If, however, the relation-weights table needs more fights than the rule weights, the proportion of the required updates is:

$$\frac{\frac{NU_1}{M}}{\frac{N(N-1)U_2}{\frac{1}{2}M(M-1)}} \approx \frac{MU_1}{2NU_2} \tag{3}$$

This proportion is linear. In the Wizard Battle experiment, a knowledge base was used with $N = 40$ rules, and a script was of size $M = 10$. If $U_1 = U_2$, the proportion is $\frac{1}{8}$. Therefore, if the relation weights need as many updates as the rule weights, we would expect that dynamic scripting with a relation-weights table would need eight times as many fights as dynamic scripting without. This was not what we found. The explanation is that the required number of updates for relation weights U_2 can be expected to be much smaller than U_1. Basically, one correct update for a relation weight is sufficient to determine the correct order for two rules, while multiple updates are needed for a rule weight of a strong rule to allow the rule to be selected with a sufficiently high probability that the turning point can be reached.

Conclusion

From our experiments, we may conclude that automatic rule ordering, using our relation-weights table with selection bonus (RWO+B) approach, is a valuable enhancement for dynamic scripting. It results in game scripts that are at least at the same level of effectiveness as scripts generated with manually-tuned priorities, at only a slight loss of efficiency. Furthermore, it has the ability to adapt more successfully to changing circumstances. Combined with the research of Ponsen *et al.* (2006), we have now enhanced dynamic scripting with the ability to generate complete knowledge bases, including rule ordering, fully automatically.

Acknowledgements

This research is supported by a grant from the Dutch Organisation for Scientific Research (NWO grant 612.066.406).

References

Charles, D., and Livingstone, D. 2004. AI: The missing link in game interface design. In Rauterberg, M., ed., *Entertainment Computing – ICEC 2004*, LNCS 3166, 351–354. Berlin, Germany: Springer-Verlag.

Dahlbom, A., and Niklasson, L. 2006. Goal-directed hierarchical dynamic scripting for RTS games. In *Proceedings of the Second Artificial Intelligence and Interactive Digital Entertainment Conference*, 21–28. Menlo Park, CA: AAAI Press.

Demasi, P., and Cruz, A. 2002. Online coevolution for action games. *International Journal of Intelligent Games and Simulation* 2(2):80–88.

Funge, J. 2004. *Artificial Intelligence for Computer Games*. Wellesley, MA: A K Peters, Ltd.

Graepel, T.; Herbrich, R.; and Gold, J. 2004. Learning to fight. In Mehdi, Q.; Gough, N.; Natkin, S.; and Al-Dabass, D., eds., *Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004)*, 193–200. Wolverhampton, UK: University of Wolverhampton.

Kaelbling, L.; Littman, M.; and Moore, A. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research* 4:237–285.

Ponsen, M., and Spronck, P. 2004. Improving adaptive game AI with evolutionary learning. In Mehdi, Q.; Gough, N.; Natkin, S.; and Al-Dabass, D., eds., *Computer Games: Artificial Intelligence, Design and Education (CGAIDE 2004)*, 389–396. Wolverhampton, UK: University of Wolverhampton.

Ponsen, M.; Muñoz-Avila, H.; Spronck, P.; and Aha, D. 2006. Automatically generating game tactics with evolutionary learning. *AI Magazine* 27(3):75–84.

Spronck, P.; Ponsen, M.; Sprinkhuizen-Kuyper, I.; and Postma, E. 2006. Adaptive game AI with dynamic scripting. *Machine Learning* 63(3):217–248.

Spronck, P.; Sprinkhuizen-Kuyper, I.; and Postma, E. 2004. Online adaptation of game opponent AI with dynamic scripting. *International Journal of Intelligent Games and Simulation* 3(1):45–53.

Spronck, P. 2005. *Adaptive Game AI*. Ph.D. thesis, Universiteit Maastricht. Maastricht, The Netherlands: Universitaire Pers Maastricht.

Sutton, R., and Barto, A. 1998. *Reinforcement Learning: An Introduction*. Cambridge, MA: MIT Press.

Timuri, T. 2006. *Automatic Rule Ordering for Dynamic Scripting*. M.Sc. thesis. Universiteit Maastricht.

Tozour, P. 2002. The perils of AI scripting. In Rabin, S., ed., *AI Game Programming Wisdom*, 541–547. Hingham, MA: Charles River Media, Inc.

Woodcock, S. 2002. AI roundtable moderator's report. www.gameai.com/cgdc02notes.html.