ONLINE ADAPTATION OF GAME OPPONENT AI WITH DYNAMIC SCRIPTING

Pieter Spronck, Ida Sprinkhuizen-Kuyper and Eric Postma
Universiteit Maastricht / IKAT
P.O. Box 616, NL-6200 MD Maastricht, The Netherlands
E-mail: p.spronck@cs.unimaas.nl

KEYWORDS
Gaming, artificial intelligence, machine learning, unsupervised online learning, opponent AI.

ABSTRACT

Unsupervised online learning in commercial computer games allows computer-controlled opponents to adapt to the way the game is being played, thereby providing a mechanism to deal with weaknesses in the game AI and to respond to changes in human player tactics. For online learning to work in practice, it must be fast, effective, robust, and efficient. This paper proposes a novel technique called "dynamic scripting" that meets these requirements. In dynamic scripting an adaptive rulebase is used for the generation of intelligent opponents on the fly. The performance of dynamic scripting is evaluated in an experiment in which the adaptive players are pitted against a collective of manually designed tactics in a simulated computer roleplaying game and in a module for the state-of-the-art commercial game NEVERWINTER NIGHTS. The results indicate that dynamic scripting succeeds in endowing computer-controlled opponents with successful adaptive performance. We therefore conclude that dynamic scripting can be successfully applied to the online adaptation of computer game opponent AI.

1 INTRODUCTION

The quality of commercial computer games is directly related to their entertainment value (Tozour 2002a). 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 (Schaeffer 2001). Improving the quality of opponent AI (while preserving the characteristics associated with high entertainment value (Scott 2002)) is desired in case human-controlled opponents are not available.

In recent years some research has been performed to endow relatively simple games, such as the action game QUAKE, with advanced opponent AI (Laird 2001). However, for more 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 much more difficult. For these complex games most AI researchers resort to scripts, i.e. lists of rules that are executed sequentially (Tozour 2002b). These scripts are generally static and tend to be quite long and complex (Brockington and Darrah 2002). This leads to two major problems, namely the *problem of complexity* and the *problem of adaptability*.

The problem of complexity entails that because of their complexity, AI scripts are likely to contain weaknesses, which can be exploited by human players to easily defeat supposedly tough opponents. The problem of adaptability entails that 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. These two problems, which are common for the opponent AI of modern CRPGs (Spronck *et al.* 2003), hamper the entertainment value of commercial computer games.

There are two ways to apply machine learning techniques to improve the quality of scripted opponent AI. The first way is to employ *offline learning* prior to the release of a game to deal with the problem of complexity (Spronck *et al.* 2003). The second way is to apply *online learning* after the game has been released to deal with both the problem of complexity and the problem of adaptability. 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 and playing style. While supervised online learning has been sporadically used in commercial games (Evans 2002), unsupervised online learning is widely disregarded by commercial game developers (Woodcock 2000), even though it has been shown to be feasible for simple games (Demasi and Cruz 2002, 2003). The present study shows that unsupervised online learning is of great potential for improving the entertainment value of commercial computer games.

Our research question reads: How can unsupervised online learning be incorporated in commercial computer games to improve the quality of the opponent AI? We propose a novel technique called *dynamic scripting* that realises online adaptation of scripted opponent AI and report on experiments performed in both a simulated and an actual commercial CRPG to assess the adaptive performance obtained with the technique.

The outline of the remainder of the paper is as follows. Section 2 discusses opponent AI in CRPGs. Section 3 describes online learning of computer game AI and the dynamic scripting technique. The experiments performed for evaluating the adaptive performance of dynamic scripting are described in section 4 and 5. In section 4 dynamic scripting is used in a simulated CRPG. In section 5 it is applied in a module for the state-of-the-art CRPG NEVERWINTER NIGHTS. Section 6 discusses the results achieved with dynamic scripting. Section 7 concludes and points at future work.

2 OPPONENT INTELLIGENCE IN CRPGS

In Computer RolePlaying Games (CRPGs) the human player is situated in a virtual world represented by a single character or a party of characters. Each character is of a specific type (e.g., a fighter or a wizard) and has certain

characteristics (e.g., weak but smart). In most CRPGs, the human player goes on a quest, which involves conversing with the world's inhabitants, solving puzzles, discovering secrets, and defeating opponents in combat. During the quest the human-controlled characters gather experience, thereby gaining more and better abilities, such as advanced spell-casting powers. Some examples of modern CRPGs are BALDUR'S GATE, NEVERWINTER NIGHTS and MORROWIND.

While combat in action games generally relies mainly on fast reflexes of the human player, combat in a CRPG usually relies on complex, strategic reasoning. The complexity arises from the fact that in each combat round both the human player and the computer-controlled opponents have a plethora of choices at their disposal. For instance, characters can execute short or long range attacks with different kinds of weapons, they can drink various potions, and they can cast a wide range of magic spells. The probabilistic nature of the results of these actions adds to the complexity of the combat process.

Opponent AI in CRPGs is almost exclusively based on scripts. Scripts are the technique of choice in the game industry to implement opponent AI in CRPGs, because they are understandable, predictable, adaptable to specific circumstances, easy to implement, easily extendable, and useable by non-programmers (Tozour 2002b, Tomlinson *et al.* 2003). Usually scripts are written and represented in a formal language that has special functions to test environmental conditions, to check a character's status, and to express commands. During the game-development phase scripts are manually adapted to ensure that they exhibit the desired behaviour. After a game's release the scripts and associated behaviours remain unchanged (unless they are updated in a game patch).

To deal with all possible choices and all possible consequences of actions the scripts controlling the opponents are of relatively high complexity. In contrast to classic CRPGs, such as the ULTIMA series, in modern CRPGs the human and opponent parties are often of similar composition (see figure 1 for an example), which entails that the opponent AI should be able to deal with the same kind of complexities as the human player faces. The challenge such an encounter offers can be highly enjoyable for human players. However, as already mentioned in the introduction, there are two major problems with application of complex, static scripts to implement opponent AI, namely the problem of complexity and the problem of adaptability. Unsupervised online learning has the potential to solve these problems. This is discussed in the following sections.

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

3 ONLINE LEARNING OF GAME AI

Unsupervised online learning of computer game AI entails that automatic learning techniques are applied that adapt the AI while the game is being played. In order for unsupervised online learning to be applicable in practice, it must meet four requirements, which are discussed in subsection 3.1. In subsection 3.2 we present dynamic scripting as an unsupervised online learning technique that meets these requirements.

3.1 Requirements for Online Learning

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.
2. **Effective**. In providing entertainment for the player, the adapted scripts should be at least as challenging as manually designed ones (the occasional occurrence of a non-challenging opponent being permissible). This requirement excludes random learning methods, such as evolutionary algorithms.
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.
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.

To meet these four requirements, we need a learning algorithm of high performance. The two main factors of importance when attempting to achieve high performance for a learning mechanism are the exclusion of randomness and the addition of domain-specific knowledge (Michalewicz and Fogel 2000). Since randomness is inherent in commercial computer games it cannot be excluded, so in this case it is imperative that the learning process is based on domain-specific knowledge.

3.2 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. These rulebases are used to create new scripts that control opponent behaviour every time a new opponent is generated. The rules that comprise a script that controls a particular opponent are extracted from the rulebase corresponding to the opponent type. The probability that a rule is selected for

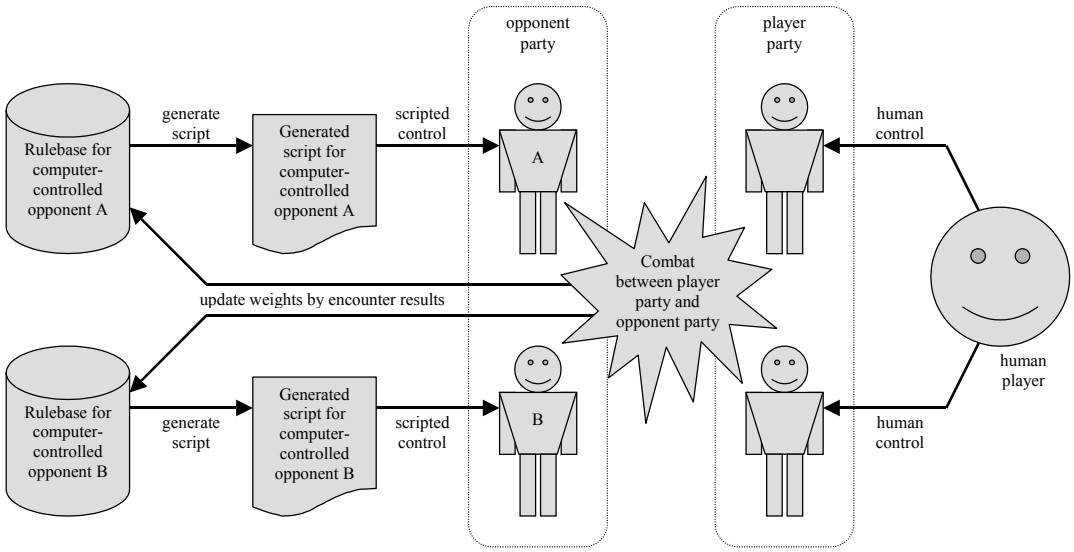

Figure 2: The dynamic scripting process. For each computer-controlled opponent a rulebase generates a new script at the start of an encounter. After an encounter is over, the weights in the rulebase are adapted to reflect the results of the fight.

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. The size of the weight changes is determined by a weight-update function.

The dynamic scripting process is illustrated in figure 2 in the context of a commercial game. The rulebase associated with each opponent contains manually designed rules that use domain-specific knowledge. At the start of an encounter, a new script is generated for each opponent by randomly selecting a specific number of rules from its associated rulebase. There is a linear relationship between the probability a rule is selected and its associated weight. The order in which the rules are placed in the script depends on the application domain. A priority mechanism can be used to let certain rules take precedence over other rules.

The learning mechanism in our dynamic scripting technique is inspired by reinforcement learning techniques (Russell and Norvig 2002). It has been adapted for use in games because regular reinforcement learning techniques do not meet the requirement of efficiency (Manslow 2002). 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 dynamic scripting technique meets at least three of the four requirements listed in 3.1. First, it is computationally fast, because it only requires the extraction of rules from a rulebase and the updating of weights once per encounter. Second, it is effective, because all rules in the rulebase are based on domain knowledge (although they may be inappropriate for certain situations). Third, it is robust because rules are not removed immediately when punished.

The dynamic scripting technique is believed to meet the fourth requirement of efficiency because with appropriate weight-updating parameters it can adapt after a few encounters only. To determine whether the belief is warranted, we performed two experiments with dynamic scripting. The first of these experiments, described in section 4, tested the efficiency of dynamic scripting in a simulated CRPG situation. The second experiment, described in section 5, tested dynamic scripting in an actual state-of-the-art CRPG to confirm that the achieved results can be repeated in practice.

4 SIMULATION EXPERIMENTS

This section describes the experiments used to test the efficiency of dynamic scripting in a simulated CRPG. It describes the problem situation to which dynamic scripting is applied (4.1), the scripts and rulebases (4.2), the weight-update function (4.3), the actual experiments (4.4) and the achieved results (4.5).

4.1 The CRPG Simulation

The gameplay mechanism in our CRPG simulation, illustrated in figure 3, was designed to resemble the popular BALDUR'S GATE games (see figure 1). These games (along

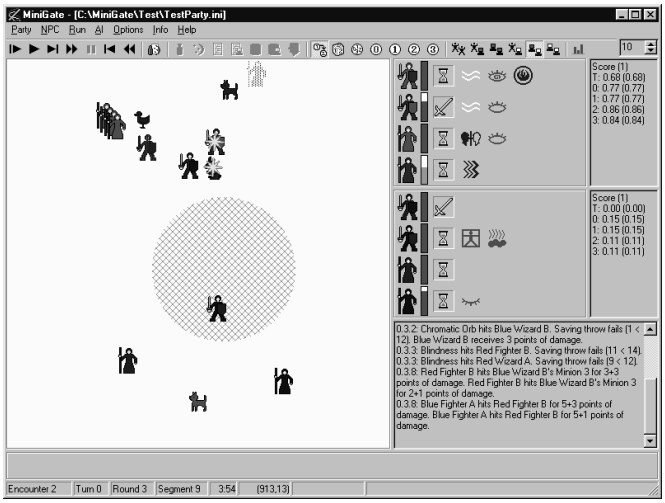

Figure 3: A screenshot of the testing environment. To the left is the combat area, the upper right shows the status of all characters in the encounter, and to the lower right a report is shown of the current effects.

	**Magic Missile** creates missiles of magical energy that unerringly strike a target creature. Each missile inflicts 1 to 5 points of damage. Starting with one missile at level 1, the wizard gains an additional missile every two levels, up to five missiles at level 9.
	**Charm Person** affects a single humanoid, who receives a saving throw vs. spells to avoid the effect. A charmed individual fights for the caster's party. The spell is broken after 5 combat rounds, when an ally tries to harm the target, or through a second charm spell.
	**Mirror Image** creates one duplicate of the caster for each caster level up to nine duplicates at level 9. Each duplicate absorbs the damage of one attack against the wizard, after which it disappears. The duplicates last a maximum of 3 rounds for each caster level.
	**Deafness** affects a single target. If the target fails a saving throw vs. spells, he or she becomes totally deaf for the duration of one day. Deafened spellcasters have a 50% chance to miscast any spell. The only cure is a *Dispel Magic* cast onto the target.
	**Stinking Cloud** creates a cloud of nauseous vapours with a radius of 20 feet around a target location. Any creature caught within the cloud must, each round, save vs. poison or be unable to move or act for 1 to 5 rounds. The cloud remains one round for each caster level.
	**Fireball** generates an explosion of flames that does 1 to 6 points of fire damage for every level of the caster (up to level 10) to any creature caught in the blast. Affected creatures receive a saving throw vs. spells for half damage.
	**Monster Summoning I** conjures 1 to 3 level 3 woodland animals which fight under the command of the caster's party. The animals remain until they are slain or the spell expires, which happens after 3 rounds plus 1 round per caster level.

Table 1: Seven spell examples.

with a few others) contain the most complex and extensive gameplay system found in modern CRPGs, closely resembling classic non-computer roleplaying games (Cook *et al.* 2000). Our simulation entails an encounter between player and opponent parties of similar composition. Each party consists of two fighters and two wizards of equal experience level. The armament and weaponry of the party is static; each character is allowed to select two (out of three possible) magic potions; and the wizards are allowed to memorise seven (out of 21 possible) spells. The spells incorporated in the simulation are of varying types, amongst which damaging spells, blessings, curses, charms, area-effect spells and summoning spells. Table 1 lists seven spell examples.

Instead of having the choices of spells and potions for opponents adapt in a separate process, we made them depend on the (generated) scripts as follows. Before the encounter starts the script is scanned to find rules containing actions that refer to drinking potions or casting spells. When such a rule is found, a potion or spell that can be used in that action is selected. If the character controlled by the script is allowed to possess the potion or spell, it is added to the character's inventory.

4.2 Scripts and Rulebases

The scripting language is designed to enable the expression of rules composed of an optional conditional statement and a single action. The conditional statement consists of one or more conditions combined with logical ANDs and ORs. Conditions can refer to a variety of environmental variables, such as the distances separating characters, the characters' health, and the spells that are suffered or benefited from. There are five basic actions: (1) attacking an enemy, (2) drinking a potion, (3) casting a spell, (4) moving, and (5) passing. In the scripting language, spells, potions, locations and characters can be referenced specifically (e.g., "cast spell 'magic missile' at closest enemy wizard"), generally (e.g., "cast any offensive spell at a random enemy") or somewhere in-between (e.g., "cast the strongest damaging spell available at the weakest enemy"). Rules in the scripts are executed in sequential order. For each rule the condition (if present) is checked. If the condition is fulfilled (or absent), the action is executed if it is both possible and useful in the situation at hand. If no action is selected when the final rule is checked, the default action 'pass' is used.

In dynamic scripting rules for a script are selected with a probability determined by the rule weights. To determine the rule order in the CRPG simulation we have assigned each rule a priority value, whereby rules with a higher priority take precedence over rules with a lower priority. For rules with equal priority we let the rules with higher weights take precedence. If two rules have both equal priorities and equal weights, their order is determined randomly.

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

4.3 The Weight-update Function

The weight-update function is based on two so-called "fitness functions": a fitness function for the party as a whole, and a fitness function for each individual character.

The fitness of a party is a value in the range [0,1], which is zero if the party has lost the fight, and 0.5 plus half the average remaining health of all party members if the party has won the fight. The fitness F for the party p (consisting of four party members) is formally defined as:

$$F(p)=\begin{cases} 0 & \{\forall n\in p \mid h(n)\le 0\} \\ 0.5+0.125\sum_{n\in p}\frac{h(n)}{mh(n)} & \{\exists n\in p \mid h(n)>0\} \end{cases}$$

where $mh(n)$ is a function that returns the health of character n at the start of the encounter (as a natural number that is greater than zero) and $h(n)$ is a function that returns the health of character n at the end of the encounter (as a natural number between zero and $mh(n)$).

The fitness of a character c is a value in the range [0,1], that is based on four factors, namely (1) the average remaining health of all party members (including character c), (2) the average damage done to the opposing party, (3) the remaining health of character c (or, if c died, the time of death) and (4) the party fitness. The fitness F for character c (who is a member of party p) is formally defined as:

$$F(p,c) = 0.05\sum_{n}\begin{cases} 0 & \{n\in p \wedge h(n)\le 0\} \\ 0.5+0.5\frac{h(n)}{mh(n)} & \{n\in p \wedge h(n)>0\} \\ 1 & \{n\notin p \wedge h(n)\le 0\} \\ 0.5-0.5\frac{h(n)}{mh(n)} & \{n\notin p \wedge h(n)>0\} \end{cases}$$

$$+ \begin{cases} \frac{\min(dc(c),100)}{1000} & \{h(c)\le 0\} \\ 0.2+0.1\frac{h(c)}{mh(c)} & \{h(c)>0\} \end{cases} + 0.3F(p)$$

where n is any of the characters in the encounter (for a total of eight characters), $dc(c)$ is the timer count at the time of death of character c and the other functions are as in the party fitness calculation. The fitness function for individual characters assigns a large reward to a victory of its party (even if the individual itself did not survive), a smaller reward to the individual's own survival, and an even smaller reward to the survival of its comrade party members and the

damage they inflicted to the opposing party. As such the character fitness function is a good measure of the success rate of the script that controls the character.

The weight-update function translates the character fitness into weight adaptations for the rules in the script. Only the rules in the script that are actually executed during an encounter were rewarded or penalised. The weight-update function is formally defined as follows:

$$W = \begin{cases} \max\left(0, W_{org} - MP \cdot \dfrac{b - F(p,c)}{b}\right) & \{F(p,c) < b\} \\ \min\left(W_{org} + MR \cdot \dfrac{F(p,c) - b}{1 - b}, MW\right) & \{F(p,c) \geq b\} \end{cases}$$

where W is the new weight value, W_{org} is the original weight value, MP is the maximum penalty, MR is the maximum reward, MW is the maximum weight value, and b is the break-even point. In our simulation we set MP to 30, MR to 100, MW to 2000 and b to 0.3. At the break-even point, weights remain unchanged. To keep the sum of all weight values in a rulebase constant, weight changes are executed through a redistribution of all weights in the rulebase. The weights in the rulebase were initialised with a value of 100.

4.4 The Experiments

The experiments aim at assessing the adaptive performance of an opponent party controlled by the dynamic scripting technique, against a player party controlled by static scripts. We defined four different basic tactics and three composite tactics for the player party. The four basic tactics, implemented as a static script for each party member, are as follows.

1. **Offensive**: The fighters always attack the nearest enemy with a melee weapon, while the wizards use the nastiest damaging spells at the most susceptible enemies.
2. **Disabling**: The fighters start by drinking a potion that frees them of any disabling effect, then attack the nearest enemy with a melee weapon. The wizards use all kinds of spells that disable enemies for a few rounds.
3. **Cursing**: The fighters always attack the nearest enemy with a melee weapon, while the wizards use all kinds of spells that harm enemies in some way. They try to charm enemies, physically weaken enemy fighters, deafen enemy wizards, summon minions in the middle of the enemy party, etc.
4. **Defensive**: The fighters start by drinking a potion that reduces fire damage, after which they attack the closest enemy with a melee weapon. The wizards use all kinds of defensive spells, to deflect harm from themselves and from their comrades, including the summoning of minions.

To assess the ability of the dynamic scripting technique to cope with sudden changes in tactics, we defined the following three composite tactics.

5. **Random party tactic**: Each encounter one of the four basic tactics is selected randomly.
6. **Random character tactic**: Each encounter each opponent randomly selects one of the four basic tactics, independent from the choices of his comrades.
7. **Consecutive party tactic**: The party starts by using one of the four basic tactics. Each encounter the party will continue to use the tactic used 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.

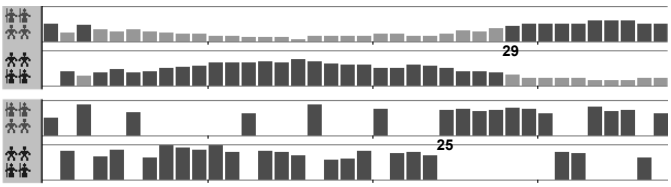

Figure 4: Two charts comparing the fitness values of two parties during a sequence of encounters. In each chart the top graph represents the opponent party, which uses the dynamic scripting technique, and the bottom graph the player party. The upper chart shows the average fitness over the last 10 encounters for both parties. In this example, from encounter 29 on, the opponent party outperforms the player party, so the average turning point is 29. The lower chart shows the absolute fitness for the two parties. This chart shows an absolute turning point of 25.

To quantify the relative performance of the opponent party against the player party, after each encounter we calculate the average fitness for each of the parties over the last ten encounters. The opponent party is said to *outperform* the player party at an encounter if the average fitness over the last ten encounters is higher for the opponent party than for the player party.

In order to identify reliable changes in strength between parties, we define two notions of the *average turning point* and the *absolute turning point* (illustrated in figure 4). The *average turning point* is the number of the first encounter after which the opponent party outperforms the player party for at least ten consecutive encounters. The *absolute turning point* is defined as the first encounter after which a consecutive run of encounters in which the opponent party wins is never followed by a longer consecutive run in which the opponent party loses. Low values for the average and absolute turning points indicate good efficiency of dynamic scripting, since they indicate that the opponent party (using dynamic scripting) consistently outperforms the player party within a few encounters only.

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

4.5 Results

Table 2 presents the results of the experiments in the simulated CRPG environment. The table lists, for each of the tactics employed by the player party, the achievements by the opponent party, which uses dynamic scripting, with respect to the average and absolute turning points. We make the following three observations.

First, the disabling tactic is easily defeated. Apparently the

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

Table 2: Results of the experiments described in section 4. For each tactic the lowest, highest, average and median average and absolute turning points are shown.

disabling tactic is not a good tactic, because dealing with it does not require adaptation of the rulebase.

Second, it is striking that for both turning points in most cases the average is significantly higher than the median. The explanation is the rare occurrence of extremely high turning points. During early encounters chance can cause potentially successful rules to get a low rating or unsuccessful rules to get a high rating. As a result, the rulebase diverges from a good weight distribution from which it has trouble recovering. Our experiments contained no mechanism to reduce the effect of early divergence, but it is clear such a mechanism is needed to make dynamic scripting a practically useful technique.

Third, the consecutive tactic, which in subsection 4.4 we argued is closest to human player behaviour, is overall the most difficult to defeat with dynamic scripting. Nevertheless, our dynamic scripting technique is capable of defeating this tactic rather quickly, especially considering the fact that the rulebase started out with all weights being equal, while in an actual game the weights would be biased from the start to give the objectively better rules a higher selection probability.

The results of our first series of experiments indicated that dynamic scripting can successfully be applied as an unsupervised online learning technique for commercial computer games. To confirm that the results achieved in the simulation are applicable to actual state-of-the-art CRPGs, in a second experiment we implemented dynamic scripting in a module for the CRPG NEVERWINTER NIGHTS. This experiment is described in the next section.

5 EXPERIMENTS IN A COMMERCIAL GAME

This section describes the experiments used to test the effectiveness of dynamic scripting in an actual commercial CRPG. It describes the game selected for these experiments and the problem situation to which dynamic scripting is applied (5.1), the scripts and rulebases (5.2), the weight-update function (5.3), the actual experiments (5.4) and the achieved results (5.5).

5.1 Commercial Game Situation

To test out dynamic scripting in practice we chose the game NEVERWINTER NIGHTS (NWN; 2002), developed by BioWare Corp. NWN is a popular, state-of-the-art CRPG. One of the reasons for its popularity, and a major reason for selecting this game for evaluating the dynamic scripting technique, is that the game is easy to modify and extend. The game comes with a toolset that allows the user to develop completely new game modules and provides access to the scripting language and all the scripted game resources, including the opponent AI. While the scripting language is not as powerful as modern programming languages, we found it to be sufficiently powerful to implement dynamic scripting.

We implemented a small module in NWN similar to the simulated CRPG detailed in section 4. The module contains an encounter between a player party and an opponent party of similar composition. This is illustrated in figure 5. Each party consists of a fighter, a rogue, a priest and a wizard of equal experience level. In contrast to the opponents in the simulated CRPG the inventory and spell selections in the NWN module can not be changed. Hence, the opponent party in the NWN module is more constrained than the opponent party in the simulation.

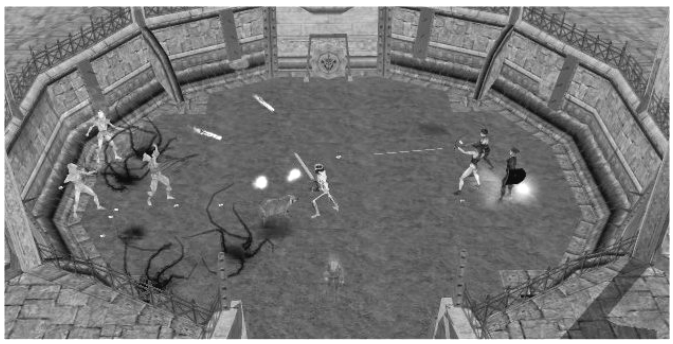

Figure 5: A battle between two parties in NEVERWINTER NIGHTS.

5.2 Scripts and Rulebases

The basic opponent AI in NWN is very general in order to facilitate the development of new game modules. It distinguishes between about a dozen opponent types and for each opponent type it sequentially checks a number of environmental variables and attempts to generate an appropriate response. The behaviour generated by NWN's AI is not completely predictable because the checking sequence and the selection of the responses is partly probabilistic.

For the implementation of the dynamic scripting process, we first extracted the rules employed by the basic opponent AI and entered them in every appropriate rulebase. To these standard NWN rules we added three types of new rules. First, we added rules that are similar to the standard rules, but slightly more specific. For instance, when a rule's action would be "attack closest enemy" we might change that to "attack closest enemy wizard". Second, we added a small number of rules that fire only in very specific circumstances, e.g., when the enemy is first spotted. Third, we added a few empty rules. Selection of the empty rules allows the opponent AI to decrease the number of effective rules.

In the generation of scripts a priority mechanism was used to order the rules. Priorities were set according to their specificity. The most specific rules had the highest priority, and the most general rules the lowest priority. Within a priority group, the rules with the largest weights were assigned the highest priority.

The size of the scripts for both a fighter and a rogue were set to five rules, which were selected out of rulebases containing 21 rules. The size of the scripts for both a priest and a wizard were set to ten rules, the rulebase for the priest containing 53 rules and the rulebase for the wizard containing 49 rules. To the end of each script a call to the basic NWN opponent AI was added, that is, if no rule could be executed the basic opponent AI would determine the actions.

5.3 The Weight-update Function

The weight adaptation mechanism we used in the NWN module made use of a party fitness function and a separate fitness function for each character, just as in the simulated CRPG (see subsection 4.3). Since the precise

implementation of these functions is not critical for the dynamic scripting technique, we decided to differ slightly from the implementation of these functions in the simulation, mainly to avoid problems with the NWN scripting language and to allow varying party sizes.

The fitness of the party is a value in the range [0,1], which is based on three factors, namely (1) whether the party has won the fight, (2) the number of party members surviving, and (3) the remaining health of the surviving party members. The fitness F for the party p is formally defined as:

$$F(p)=\begin{cases} 0 & \{\forall n\in p \mid h(n)\le 0\} \\ 0.2+0.4\dfrac{count(n\in p \mid h(n)>0)+\sum_{n\in p}\dfrac{h(n)}{mh(n)}}{count(n\in p)} & \{\exists n\in p \mid h(n)>0\} \end{cases}$$

where *count* is a function that counts the number of instances of its parameter, $mh(n)$ is a function that returns the health of character n at the start of the encounter (as a natural number that is greater than zero) and $h(n)$ is a function that returns the health of character n at the end of the encounter (as a natural number between zero and $mh(n)$).

The fitness of a character is a value in the range [0,1], that is based on three factors, namely (1) whether the character survived or not, (2) the remaining health of the character (or, if the character died, the time of death), (3) the party fitness. The fitness F for character c (who is a member of party p) is formally defined as:

$$F(p,c)=\begin{cases} \dfrac{\min(dc(c),30)}{100} & \{h(c)\le 0\} \\ 0.3+0.2\dfrac{h(c)}{mh(c)} & \{h(c)>0\} \end{cases} + \; 0.5F(p)$$

where $dc(c)$ is the timer count at the time of death of character c and the other functions are as in the party fitness calculation. The fitness function for individual characters assigns a large reward to a victory of their party (even if the individual itself did not survive), a smaller reward to the individual's own survival and an even smaller reward to the size of the remaining health.

The weight-update function in the NWN module was equal to the weight-update function of the simulation, as defined in subsection 4.3, except that the maximum penalty MP was set to 50. Furthermore, rules in the script that were *not* executed during the encounter, instead of being treated as not being in the script at all, we assigned half the reward or penalty received by the rules that were executed. The main reason for this is that if there were no rewards and penalties for the non-executed rules, the empty rules would never get rewards or penalties.

5.4 The Experiments

Since the simulation experiments already showed that dynamic scripting is an efficient technique, the NWN experiments were mainly aimed at evaluating whether dynamic scripting works as well in a practical situation as in the simulation. We used the same notions of average turning point and absolute turning point (see subsection 4.4) to evaluate the performance of the dynamic scripting technique in these experiments.

While in the simulation experiments the learning opponent party was pitted against several manually programmed strategies employed by the player party, in the NWN experiments we pitted the learning party against the basic NWN opponent AI. The behaviour of the basic opponent AI is somewhat unpredictable and tries to adapt to the circumstances of an encounter. We observed that a party using the basic AI outperforms an unadapted opponent (i.e. an adaptive opponent that has all weights in the rulebase set to the same value).

We ran eleven tests, starting with newly initialised rulebases for each of the characters. Each test continued until the average turning point was reached. The results of the tests are presented in the next subsection.

5.5 Results

The results of the NWN experiments are presented in table 3. The table shows that the results achieved in these experiments are similar to the results achieved in the simulation experiments. Apparently, dynamic scripting can be successfully applied in a state-of-the-art CRPG.

	Average Turning Point				Absolute Turning Point			
Tactic	Low	High	Avg.	Med.	Low	High	Avg.	Med.
Basic AI	10	101	34	27	6	96	33	29

Table 3: Results of the experiments described in section 5. There is only one opponent tactic, namely the basic AI as implemented by the NWN developers. The lowest, highest, average and median average and absolute turning points are shown.

6 DISCUSSION

Our experimental results show that dynamic scripting is capable of adapting rapidly to static or changing tactics. Hence, dynamic scripting is efficient and meets the four requirements stated in section 3 (fast, effective, robust and efficient). The results achieved with the NWN experiments detailed in section 5 clearly show that the implementation is commercially feasible, although some improvements are needed. In this section we discuss the following issues: improving dynamic scripting (6.1), offline dynamic scripting (6.2), generalisation within a game (6.3) and to other games (6.4), and the point-of-view of game developers (6.5).

6.1 Improving Dynamic Scripting

The results of the experiments, presented in subsections 4.5 and 5.5, show that in some exceptional cases the adaptation process of the rulebases can become excessively long. We examined some of the rulebases that were generated in these cases, and found them to contain high weight values for rules that represent undesirable behaviour. Such behaviour, once learned (supposedly through chance), evidently can be difficult to unlearn. A straightforward solution is to store successful copies of the rulebase and to revert to an earlier rulebase when the performance seems to deteriorate.

We noted that if we let our experiments continue even after an average turning point was discovered, it sometimes happened after a while that the rulebase started to generate inferior scripts. This is because the rulebase continues to learn new behaviour, even when it is already successful. Simply stopping the learning process when it has reached an optimum is not a good solution, because our goal is to let the rulebase adapt to *changing* player tactics. A better solution is

to develop a mechanism that protects the rulebase from degrading, such as the previously suggested storing of copies of successful rulebases.

6.2 Offline Dynamic Scripting

In our online learning experiments we only adapt weight values, rather than changing existing rules or adding new rules. In our view, such techniques would severely reduce the efficiency of the process and might interfere with the effectiveness of the generated scripts. However, during an offline training phase, which optimises the rulebase before a game is released, such techniques are certainly possible and can even be successful (Spronck *et al.* 2003).

6.3 Generalisation within a Game

Our experiments were limited to optimising the opponent AI in one specific encounter. In practice, games usually challenge the player with many different encounters, rather than with one specific encounter that is repeated over and over again. Especially in strategic CRPGs, such as BALDUR'S GATE and NEVERWINTER NIGHTS, different instances of an opponent type will have different characteristics, different spells and different equipment. Is it possible to use dynamic scripting to optimise a rulebase for an opponent type in these ever changing circumstances? We argue that it is, provided the rules in the rulebase are generalised. For instance, rules should not refer to the casting of specific spells, but instead to the casting of spells of a certain type (e.g., instead of stating "cast a fireball" a rule should state "cast a high-level damaging area-effect spell"). While, for instance, each opponent wizard in the game might be different, successful generalised tactics for one wizard will also work for most other wizards. Therefore, with generalised rules any wizard encounter can be used to update weights for the wizard opponent type. Our argument is supported by the fact that the basic opponent AI in NWN employs such generalised rules to ensure that the AI can adequately control any opponent instance that can be created.

6.4 Generalisation to Other Games

Although dynamic scripting turns out to be surprisingly efficient and effective for implementing online learning in some commercial games, the question remains whether it can be made sufficiently efficient for application in every type of commercial game. For action games the answer would be an unequivocal yes, because action games typically pit the player against hundreds of copies of one particular opponent instance. For strategic games where there is much variety in opponents it depends on how many different opponent types can be distinguished and how many instances of an opponent type can be found in the game. For practical purposes game designers will usually place only a limited number of different opponent types in a game, and many instances of each type, each slightly different from the rest. Most games, therefore, will contain enough encounters with each opponent type to optimise the associated dynamic scripting rulebase. If the weights in the rulebase do not all start at equal values, but are biased to give the objectively better rules a greater selection probability (for instance through an offline training process), the adaptation process will need even less trials than in our experiments, while the ability to adapt to novel tactics is preserved.

6.5 The Point-of-view of Game Developers

To the four requirements we defined for online learning (fast, effective, robust and efficient) commercial game developers would add two extra ones, namely that (5) the resulting AI should be understandable (which will make it easier for them to place their trust in it), and (6) the resulting AI should be non-repetitive (so it will not be too predictable, which detracts from the entertainment value). We argue that dynamic scripting meets these two additional requirements as well. First, dynamic scripting generates scripts, and therefore the results are understandable by definition. Second, scripts are always generated at random for each new encounter and thus the AI is non-repetitive. The differences between scripts will be greater if the maximum weight values are set low enough so that a considerable number of rules will end up with large enough weights to be selected often.

Commercial game developers would, however, not agree to have opponents attempt to learn to defeat the human player at all costs, which is what our fitness criterion, that relies heavily on winning and losing encounters, actually promotes. In commercial games, the human player should (because of entertainment purposes) and will (because of saving and reloading functionalities) always win an encounter. Therefore, in an actual commercial game fitness should rely more on the amount of damage done and the length of the fights. It might even be useful to punish a rulebase for winning a fight or damaging the player party too much, so that the entertainment value of the game for weaker players is protected.

Finally, it should be noted that if online learning is implemented in a game, the game will learn different things from different players and therefore different players will have a different playing experience. Publishers will insist that the Quality Assurance team will cover these different playing experiences while testing the game, which will obviously make the team's task harder. This issue should be addressed in the design phase of a game, when the decision to use an online learning technique is made.

7 CONCLUSIONS AND FUTURE WORK

In this paper we proposed dynamic scripting as a technique to deal with unsupervised online adaptation of opponent AI, suitable for implementation in complex commercial computer games such as CRPGs. Dynamic scripting is based on the automatic online generation of AI scripts for computer game opponents by means of an adaptive rulebase. From our experimental results, we conclude that dynamic scripting is fast, effective, robust, and efficient and therefore has the potential to be successfully incorporated in commercial games. We tested the technique in a module for a state-of-the-art commercial CRPG, BioWare's NEVERWINTER NIGHTS, which showed that the technique works as well in practice as it does in the simulation. However, some changes are needed before the technique is ready to be implemented in actual commercial games.

Specifically, the algorithm should be augmented with a technique that protects the adaptation mechanism against learning ineffective behaviour by chance and then having difficulty to unlearn this inferior behaviour.

Our future work aims at optimising the dynamic scripting technique to make commercial implementation viable. In particular, we focus on tweaking the learning parameters, seeking ways to store successful rulebases to recover from inferior performance, studying methods to optimise the ranking of rules in the scripts, and experimenting with offline learning to optimise a rulebase before online learning takes place. Furthermore, 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 entertainment 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

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

Cook, M., J. Tweet and S. Williams. 2000. *Dungeons & Dragons Player's Handbook*. Wizards of the Coast.

Demasi, P. and A.J. de O. Cruz. 2002. "Online Coevolution for Action Games." *International Journal of Intelligent Games and Simulation* (eds. N.E. Gough and Q.H. Mehdi), Vol. 2, No. 2. University of Wolverhampton and EUROSIS , pp. 80-88.

Demasi, P. and A.J. de O. Cruz. 2003. "Anticipating Opponent Behaviour Using Sequential Prediction and Real-Time Fuzzy Rule Learning." *Proceedings of the 4th International Conference on Intelligent Games and Simulation (GAME-ON 2003)* (eds. Q. Mehdi, N. Gough and S. Natkin). EUROSIS, Belgium, pp. 101-105.

Evans, R. 2002. "Varieties of Learning." *AI Game Programming Wisdom* (ed. S. Rabin). Charles River Media, pp. 567-578.

Laird, J.E. 2001. "It Knows What You're Going To Do: Adding Anticipation to a Quakebot." *Proceedings of the Fifth International Conference on Autonomous Agents*, pp. 385-392.

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

Michalewicz, Z. and D.B. Fogel. 2000. *How To Solve It: Modern Heuristics*. Springer Verlag, 2000.

Russell, S. and P. Norvig. 2002. *Artificial Intelligence: A Modern Approach*, Second Edition. Prentice Hall, Englewood Cliffs, New Jersey.

Schaeffer, J. 2001. "A Gamut of Games." *AI Magazine*, Vol. 22 No. 3, pp. 29-46.

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

Spronck, P., I. Sprinkhuizen-Kuyper and E. Postma. 2002. "Improving Opponent Intelligence Through Offline Evolutionary Learning." *International Journal of Intelligent Games and Simulation* (eds. N.E. Gough and Q.H. Mehdi), Vol. 2, No. 1. University of Wolverhampton and EUROSIS, pp. 20-27.

Tomlinson, S.L., A. Davies and S. Assadourian. 2003. "Working at Thinking About Playing or A Year in the Life of a Games AI Programmer." *Proceedings of the 4th International Conference on Intelligent Games and Simulation (GAME-ON 2003)* (eds. Q. Mehdi, N. Gough and S. Natkin). EUROSIS, Belgium, pp. 5-11.

Tozour, P. 2002a. "The Evolution of Game AI." *AI Game Programming Wisdom* (ed. S. Rabin). Charles River Media, pp. 3-15.

Tozour, P. 2002b. "The Perils of AI Scripting." *AI Game Programming Wisdom* (ed. S. Rabin). Charles River Media, pp. 541-547.

Woodcock, S. 2000. "Game AI: The State of the Industry." *Game Developer Magazine,* August 2000.

ACKNOWLEDGEMENTS

The authors wish to express their gratitude to the University of Alberta GAMES Group and the Netherlands Organization for Scientific Research (NWO) for their support of this research, and to BioWare Corp. for their enlightening commentary.