Learning Character Behaviors using Agent Modeling in Games

Richard Zhao, Duane Szafron

Department of Computing Science, University of Alberta
Edmonton, Alberta, Canada T6G 2E8
{rxzhao, duane} @cs.ualberta.ca

Abstract

Our goal is to provide learning mechanisms to game agents so they are capable of adapting to new behaviors based on the actions of other agents. We introduce a new on-line reinforcement learning (RL) algorithm, ALeRT-AM, that includes an agent-modeling mechanism. We implemented this algorithm in BioWare Corp.'s role-playing game, *Neverwinter Nights* to evaluate its effectiveness in a real game. Our experiments compare agents who use ALeRT-AM with agents that use the non-agent modeling ALeRT RL algorithm and two other non-RL algorithms. We show that an ALeRT-AM agent is able to rapidly learn a winning strategy against other agents in a combat scenario and to adapt to changes in the environment.

Introduction

A story-oriented game contains many non-player characters (NPCs). They interact with the player character (PC) and other NPCs as independent agents. As increasingly realistic graphics are introduced into games, players are starting to demand more realistic behaviors for the NPCs. Most games today have manually-scripted NPCs and the scripts are usually simple and repetitive since there are hundreds of NPCs in a game and it is time-consuming for game developers to script each character individually. ScriptEase (ScriptEase 2009), a publicly-available author-oriented developer tool, attempts to solve this problem by generating script code for BioWare Corp.'s *Neverwinter Nights* (NWN 2009), from high-level design patterns. However, the generated scripts are still static in that they do not change over time as the NPCs gain experience. In essence, the NPCs do not learn from their failures or successes. The ALeRT algorithm (Cutumisu et al. 2008) attempted to solve this problem by using reinforcement learning (RL) to automatically generate NPC behaviors that change over time as the NPC learns.

The ALeRT (Action-dependent Learning Rates with Trends) algorithm is based on the single agent online learning algorithm Sarsa(λ). The algorithm was implemented in a combat environment to test its effectiveness. It was demonstrated to achieve the same performance as Spronck's rule-based dynamic scripting (DS-B) algorithm (Spronck et al. 2006) when the environment remains static and adapts better than DS-B when the environment changes.

Although the ALeRT algorithm was successful in a simple situation (combat between two fighter NPCs), we applied it to more complex learning situations: combat between two sorcerer NPCs and combat between mixed teams of sorcerers and fighters. We discovered that when an NPC's "best" action was dependent on the actions of other NPCs, ALeRT sometimes could not find this "best" strategy. We used an "agent model" to predict the other NPCs' actions and used this extra state to learn a better strategy. In this paper, we describe a modified version of the ALeRT algorithm, called ALeRT-AM (ALeRT with Agent Modeling), and we present the results of experiments that we conducted to compare ALeRT-AM to both ALeRT and DS-B.

The experiments were conducted using the NWN game combat module provided by Spronck (NWN Arena 2009). We show that with agent modeling, the augmented algorithm ALeRT-AM is able to achieve equal results to ALeRT in situations with simple winning strategies, and it was able to obtain better results when the winning strategies depend on the opposing agent's actions.

Related Works

It is not common for commercial games to apply reinforcement learning techniques, since RL techniques generally take too long to learn and most NPCs are not around long enough (Spronck et al. 2003). Research has been done to incorporate RL algorithms into various genres of games. A planning/RL hybrid method for real-time strategy games is proposed by Sharma, et al. (2007), where Q-learning, an RL technique, is used to update the utilities of high-level tactics and a planner chooses the tactic with the highest expected utility. Others have explored the use of RL techniques in first-person shooters (Smith et al. 2007). Similarly, a variant of the Q-learning algorithm is used to learn a team strategy, rather than the behaviors of individual characters. We are interested in developing an architecture in which individual NPCs can learn interesting and effective behaviors. The challenge is in defining a learning model with useful features, accurate reward

functions and a mechanism to model other agents. In addition, this leaning model must be capable of being incorporated into behaviors by game authors that have no programming skills (Cutumisu 2009).

In story-based games, the DS-B dynamic scripting algorithm (Spronck et al. 2006) is an alternative that uses a set of rules from a pre-built rule-base. A rule is defined as an action with a condition, *e.g.* if my health is below 50%, try to heal myself. A "script" containing a sequence of rules is dynamically-generated and an RL technique is applied to the script generation process. In Spronck's test cases, each "script" for a fighter NPC contains five rules from a rule-base of twenty rules. For a sorcerer NPC, each "script" contains ten rules from a rule-base of fifty rules. The problem with this approach is that the rule-base is large and it has to be manually ordered (Timuri et al. 2007). Moreover, once a policy is learned by this technique, it is not adaptable to a changing environment (Cutumisu et al. 2008). Improvements have been made by combining DS-B with "macros" (Szita et al. 2008). Macros are sets of rules specialized in certain situations, *e.g.* an opening macro or a midgame macro. By having a rule-base of learned macros instead of singular rules, it is shown that adaptivity can be increased. However, these techniques do not take into consideration the actions of an opponent agent.

There are attempts at opponent modeling in real-time strategy games, using classifiers on a hierarchically structured model (Schadd et al. 2007), but so far no attempts have been made in story-based games. Researchers have applied opponent-modeling techniques to poker, a classical game (Billings et al. 2002), but the methods are not directly applicable to story-based games, since the rules in poker are too mechanical to be applied to NPC behaviors. The algorithm to be introduced in this paper is based on a non-agent modeling technique called ALeRT (Cutumisu et al. 2008), which we summarize in the next section.

The ALeRT algorithm

The ALeRT algorithm introduced a variation of a single-agent on-line RL algorithm, Sarsa(λ) (Sutton and Barto 1998). A learning agent keeps a set of states of the environment and a set of valid actions the agent can take in the environment. Time is divided into discrete steps and only one action can be taken at any given time step. At each time step t, an immediate reward, r, is calculated from observations of the environment. A policy π is a mapping of each state s and each action a to the probability of taking action a in state s. The value function for policy π, denoted $Q^{\pi}(s,a)$, is the expected total reward (to the end of the game) of taking action a in state s and following π thereafter. The goal of the learning algorithm is to obtain a running estimate, $Q(s,a)$, of the value function $Q^{\pi}(s,a)$ that evolves over time as the environment changes dynamically. The on-policy Sarsa algorithm differs from the off-policy Q-learning algorithm in that with Sarsa, the running estimate $Q(s,a)$ approximates the value function for a chosen policy π, instead of an optimal value function. Our estimate $Q(s,a)$, is initialized arbitrarily and is learned from experience. As with Sarsa(λ), ALeRT uses the Temporal-Difference prediction method to update the estimate $Q(s,a)$, where α denotes the learning rate, γ denotes the discount, and e denotes the eligibility trace:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)] e(s_t, a_t)$$

In Sarsa(λ), α is either a small fixed value or decreasing in order to guarantee convergence. Within a computer game environment, the slow learning rate of Sarsa(λ) poses a serious problem. The ALeRT algorithm analyzes trends in the environment by tracking the change of $Q(s,a)$ at each step. The trend measure is based on the Delta-Bar-Delta measure (Sutton 1992) where a window of previous steps is kept. If the current change follows the trend, then the learning rate is increased; and if the current change differs from the trend, then learning rate is decreased.

Secondly, as opposed to a global learning rate, α, ALeRT establishes a separate learning rate, $\alpha(a)$ for each action, a, of the learning agent. This enables each action to form its own trend so that when the environment is changed, only the actions affected by that change register disturbances in their trends.

Thirdly, as opposed to a fixed exploration rate, ALeRT uses an adjustable exploration rate, changing in accordance with a positive or negative reward. If the received reward is positive, implying the NPC has chosen good actions, the exploration rate is decreased. On the other hand, if the reward is negative, the exploration rate is increased.

The ALeRT algorithm is a general purpose learning algorithm which can be applied to the learning of NPC behaviors. The easiest way to evaluate the effectiveness of a learning algorithm is to test it in a quantitative experiment. It is difficult to test behaviors whose rewards are not well-defined, so this general algorithm was tested in a situation with very concrete and evaluable reward functions – combat.

To evaluate the effectiveness of the algorithm, it was applied to Neverwinter Nights, in a series of experiments with fighter NPCs, where one fighter used the ALeRT algorithm to select actions and the opponent fighter used other algorithms (default NWN, DS-B, hand-coded optimal). A combat scenario was chosen because the results of the experiment can be measured using concrete numbers of wins and loses. ALeRT achieved good results in terms of policy-finding and adaptivity. However, when we applied the ALeRT algorithm in a duel between two sorcerer NPCs (fighters and sorcerers are common units in NWN), the experimental results were not as good as we expected (see Experiments and Evaluation).

A fighter only has limited actions. For example, a fighter may only: use a melee weapon, use a ranged weapon, drink a healing potion or drink an enhancement potion (speed). However, a sorcerer has more actions, since sorcerers can cast a number of spells. We abstracted the spells into eight categories. This gives a sorcerer eight actions instead of the four actions available to fighters.

In a more complex environment where both sides have a fighter and a sorcerer, ALeRT can also be applied. Each action is appended with a target. The targets are friendly fighter, friendly sorcerer, enemy fighter, and enemy sorcerer, depending on the validity of the action. This system can be extended easily to multiple members for each team.

ALeRT-AM: ALeRT with Agent Modeling

With ALeRT, each action is chosen based on the state of the environment that does not include knowledge of recent actions of other agents. Although there is no proof that ALeRT always converges to an optimal strategy, in practice it finds a strategy that gives positive rewards in most situations. In the case of a fighter, where there are only a limited number of simple actions, a winning strategy can be constructed that is independent of the other agent's actions. For example, in the four-action fighter scenario it is always favorable to take a speed enhancement potion in the first step. Unfortunately, when more complex actions are available the actions of the other agents are important. In the case of sorcerers, the spell system in NWN is balanced so that spells can be countered by other spells. For example, a fireball spell can be rendered useless by a minor globe of invulnerability spell. In such a system, any favorable strategy has to take into consideration the actions of other agents, in addition to the state of the environment. The task is to learn a model of the opposing agent and subsequently come up with a counter-strategy that can exploit such information.

We adapted the ALeRT algorithm to produce ALeRT-AM, by adding features based on the predicted current actions of other agents using opponent modeling Q-learning (Uther and Veloso 1997). We modified the value function to contain three parameters, the current state s, the agent's own action a, and the opposing agent's action a'. We denote the modified value function $Q(s, a, a')$. At every time step, the current state s is observed and action a is selected based on a selection policy, ε-greedy, where the action with the largest estimated $Q(s, a, a')$ is chosen with probability $(1-\varepsilon)$, and a random action is chosen with probability ε. Since the opponent's next action cannot be known in advance, a' is estimated based on a model built using past experience. The ALeRT-AM algorithm is shown in Figure 1.

$N(s)$ denotes the frequency of game state s and $C(s, a)$ denotes the frequency of the opponent choosing action a in game state s. For each action a, the weighted average of the value functions $Q(s, a, a')$ for each opponent action a' is calculated, based on the frequency of each opponent action. This weighted average is used as the value of action a in the ε-greedy policy, as shown on the line marked by **.

Initialize Q arbitrarily, $C(s, a) \leftarrow 0, N(s) \leftarrow 0$ for all s,a

Initialize $\alpha(a) \leftarrow \alpha_{\max}$ for all a

Repeat (for each episode) until s is terminal

$\vec{e} = \vec{0}$

$s, a, a' \leftarrow$ initial state and action of episode

Let F represent the set of features from s, a, a'

Repeat (for each step t of episode)

For all $i \in$ F:

$e(i) \leftarrow e(i) + \frac{1}{\|\mathrm{F}\|}$ (accummulating eligibility traces)

Take action a_t, observe reward, r_t,

opponent's action a'_t and next state, s_{t+1}

With probability $1-\varepsilon$:

** $a_{t+1} \leftarrow argmax_a \sum_{a'} \frac{C(s_t,a')}{N(s_t)} Q(s_t,a,a')$

or with probability ε:

$a_{t+1} \leftarrow$ a random action $\in$ A

$\delta \leftarrow r_t + \gamma \sum_{a'} \frac{C(s_{t+1},a')}{N(s_{t+1})} Q(s_{t+1},a_{t+1},a') - Q(s_t,a_t,a'_t)$

$Q \leftarrow Q + \alpha(a)\delta\vec{e}$

$\vec{e} = \gamma\lambda\vec{e}$

$C(s_t,a'_t) \leftarrow C(s_t,a'_t) + 1$

$N(s_t) \leftarrow N(s_t) + 1$

$\Delta\alpha = \frac{\alpha_{\max} - \alpha_{\min}}{\alpha_{\text{steps}}}$

if $\overline{\delta(a)}\delta > 0 \ \wedge \left|\overline{\delta(a)} - \mu_{\overline{\delta(a)}}\right| > f\sigma_{\overline{\delta(a)}}$

$\alpha(a) \leftarrow \alpha(a) + \Delta\alpha$

else if $\overline{\delta(a)}\delta \le 0$

$\alpha(a) \leftarrow \alpha(a) - \Delta\alpha$

end of step

$\Delta\varepsilon = \frac{\varepsilon_{\max} - \varepsilon_{\min}}{\varepsilon_{\text{steps}}}$

if $\sum_{step} r_{step} = 1$

$\varepsilon = \varepsilon - \Delta\varepsilon$

else

$\varepsilon = \varepsilon + \Delta\varepsilon$

end of episode

Figure 1. The ALeRT-AM algorithm.

NWN Implementation

We used combat to test the effectiveness of the ALeRT-AM algorithm, for all the same reasons as it was used to test ALeRT. We conducted a series of learning experiments in NWN using Spronck's arena-combat environment. An agent is defined as an AI-controlled character (A non-player character, or NPC), and a player character (PC) is controlled by the player. Each agent responds to a set of events in the environment. For example, when an agent is attacked, the script associated with the event *OnPhysicalAttacked* is executed.

An episode is defined as one fight between two opposing teams, starting when all agents of both teams have been created in the arena and ending as soon as all agents from one team are destroyed. A step is defined as one round of combat, which lasts six seconds of game time.

Every episode can be scored as a zero-sum game. When each team consists of one agent, the total score of an episode is 1 if the team wins and -1 if the team loses. The immediate reward function of one step is defined as:

$$r = 2[\frac{\hat{H}_s}{\hat{H}_s + \hat{H}_o} - \frac{H_s}{H_s + H_o}]$$

H represents the hit points of an agent where the subscript s denotes the hit points of the agent whose actions are being selected and the subscript o denotes the hit points of the opposing agent. An H with a hat (^) denotes the hit points at the current step and an H without a hat denotes the hit points at the previous step.

When each team consists of two agents, the total score can be defined similarly. The hit points of all team members are added together. The total score of an episode is 1 if the team wins and -1 if the opposing team wins. The immediate reward function of one step is defined as:

$$r = 2[\frac{\hat{H}_{s1} + \hat{H}_{s2}}{\hat{H}_{s1} + \hat{H}_{o1} + \hat{H}_{s2} + \hat{H}_{o2}} - \frac{H_{s1} + H_{s2}}{H_{s1} + H_{o1} + H_{s2} + H_{o2}}]$$

The subscripts 1 and 2 denote team members 1 and 2.

The feature vector contains combinations of states from the state space and actions from the action space. Available states and actions depend on the properties of the agent. Two types of agents are used in our experiment, a fighter and a sorcerer. The state space of a fighter consists of three Boolean states: 1) the agent's hit points are lower than half of the initial hit points; 2) the agent has an enhancement potion available; 3) the agent has an active enhancement effect. The action space of a fighter consists of four actions: *melee*, *ranged*, *heal*, and *speed*. The state space of a sorcerer consists of four Boolean states: 1) the agent's hit points are lower than half of the initial hit points; 2) the agent has an active combat-protection effect; 3) the agent has an active spell-defense effect; 4) the opposing sorcerer has an active spell-defense effect. The action space of a sorcerer consists of the eight actions shown in Table 1. When agent modeling is used by the sorcerer, there is also an opponent action space, consisting of equivalent actions for the opposing sorcerer agent.

Experiments and Evaluation

The experiments were conducted with Spronck's NWN arena combat module, as shown in Figure 2. For the two opposing teams, one team is scripted with our learning algorithm, while the other team is scripted with one strategy from the following set. NWN is the default Neverwinter Nights strategy, a rule-based static probabilistic strategy. DS-B represents Spronck's rule-based dynamic scripting method. ALeRT is the unmodified version of the online-learning strategy, while ALeRT-AM is the new version that includes agent modeling.

Action Category	Description
Attack melee	Attack with the best melee weapon available
Attack ranged	Attack with the best ranged weapon available
Cast combat-enhancement spell	Cast a spell that increases a person's ability in physical combat, *e.g.* Bull's strength
Cast combat-protection spell	Cast a spell that protects a person in physical combat, *e.g.* Shield
Cast spell-defense spell	Cast a spell that defends against hostile spells, *e.g.* Minor Globe of Invulnerability
Cast offensive-area spell	Cast an offensive spell that targets an area, *e.g.* Fireball
Cast offensive-single spell	Cast an offensive spell that targets a single person, *e.g.* Magic Missile
Heal	Drink a healing potion

Table 1. The actions for a sorcerer

Figure 2. The arena showing combat between two teams.

Each experiment consisted of ten trials and each trial consisted of either one or two phases of 500 episodes. All agents started with zero knowledge of themselves and their opponents, other than the set of legal actions they can take. At the start of each phase, each agent was equipped with a specific set of equipment and in case of a sorcerer, a specific set of spells. In a one-phase experiment, we evaluated how quickly our agents could find a winning strategy. In a two-phase experiment, we evaluated how well our agents could adapt to a different set (configuration) of sorcerer spells. It should be noted that the NWN combat system uses random numbers, so there is always an element of chance.

Motivation for ALeRT-AM

The original ALeRT algorithm, with a fighter as the agent, was shown to be superior to both traditional Sarsa(λ) and NWN in terms of strategy-discovering and more adaptive to environmental change than DS-B (Cutumisu et al. 2008). We begin by presenting the results of experiments where ALeRT was applied to more complex situations.

Figure 3 shows the result of ALeRT versus the default NWN algorithm, for three different teams, one fighter team (Cutumisu et al. 2008), a new sorcerer team and a new sorcerer-fighter team. The Y-axis represents the average number of episodes that the former team has won. ALeRT is quick to converge on a good counter-strategy that results in consistent victories for all teams.

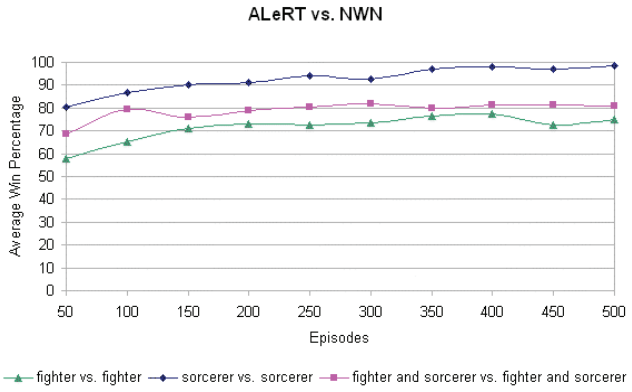

Figure 3. ALeRT against NWN for several teams.

Figure 4 shows the results of ALeRT vs. DS-B, for the same teams. For the fighter team and the fighter-sorcerer team, both ALeRT and DS-B are able to reach an optimal strategy fairly quickly, resulting in a tie.

In the more complex case of the sorcerer team, the results have higher variance with an ALeRT winning rate that drops to about 50% as late as episode 350. These results depend on whether an optimal strategy can be found that is independent of actions of the opposing team. A lone fighter has a simple optimal strategy, which does not depend on the opponent. In the fighter-sorcerer team, the optimal strategy is for both the fighter and sorcerer to focus on killing the opposing sorcerer first, and then the problem is reduced to fighter vs. fighter.

However, with a sorcerer team, there is no single static best strategy, as for every action, there is a counter-action. For example, casting a minor globe of invulnerability will render a fireball useless, but the minor globe is ineffective against a physical attack. The best strategy depends on the opponent's actions and the ability to predict the opponent's next action is crucial.

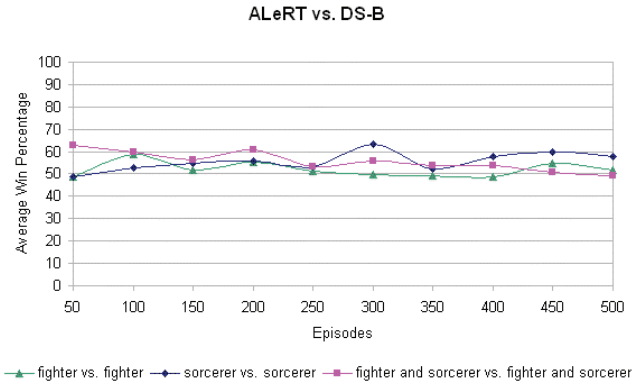

Figure 4. Motivation for ALeRT-AM: ALeRT versus DS-B for several teams

Agent modeling

With ALeRT and agent modeling, ALeRT-AM achieves an approximately equal result with ALeRT when battling against the default NWN strategy (Figure 5). The sorcerer team defeats NWN with a winning rate above 90% while the fighter-sorcerer team achieves an 80% winning rate.

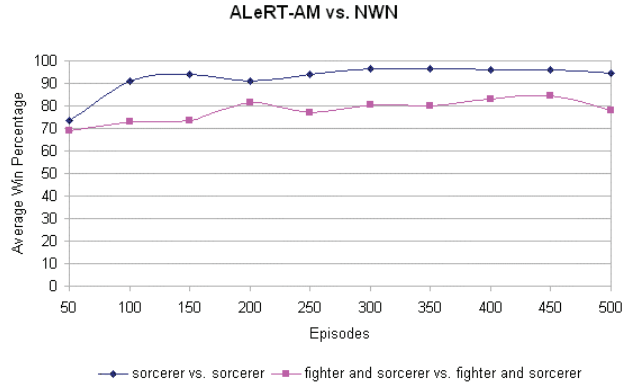

Figure 5. ALeRT-AM against NWN for several teams.

With ALeRT-AM versus DS-B, the results are much more consistent (Figure 6). For the sorcerer team, ALeRT-AM derives a model for the opponent in less than 100 episodes and is able to keep its winning rate consistently above 62%. For the fighter-sorcerer team, ALeRT-AM does better than ALeRT against DS-B by achieving and maintaining a winning rate of 60% by episode 300.

The ALeRT-AM algorithm was also tested again the original ALeRT algorithm. Figure 7 shows the results for the sorcerer teams and the fighter-sorcerer teams. ALeRT-

AM has an advantage over ALeRT at adapting to the changing strategy, generally keeping the winning rate above 60% and quickly recovering from a disadvantaged strategy, as shown near episode 400 in the sorcerer vs. sorcerer scenario (a turning point on the blue line in the graph).

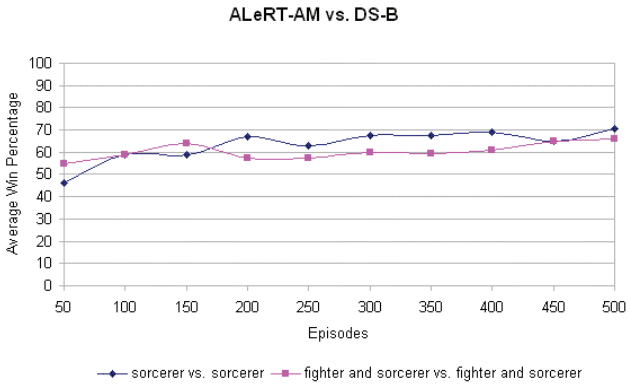

Figure 6. ALeRT-AM versus DS-B for several teams

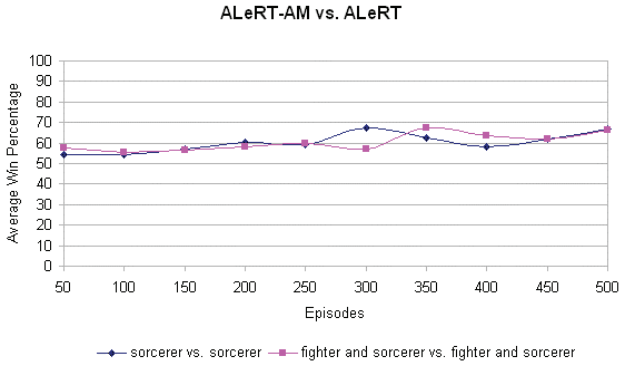

Figure 7. ALeRT-AM versus ALeRT for several teams

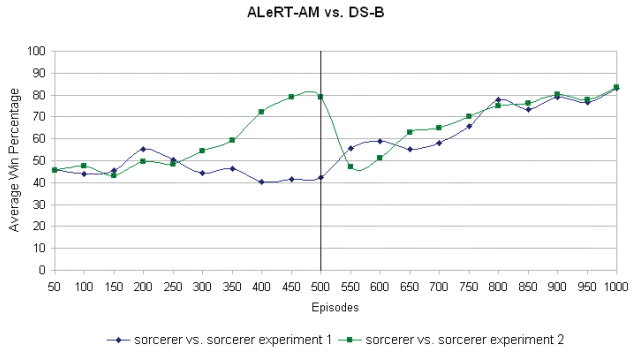

Figure 8. ALeRT-AM versus DS-B in a changing environment

Adaptation in a dynamic environment

ALeRT was shown to be adaptable to change in the environment for the fighter team (Cutumisu et al. 2008), by changing the configuration at episode 501 (the second phase). For a fighter team, a better weapon was given in the second phase. We demonstrate that the adaptability remains even with agent modeling (Figure 8). For a sorcerer team, the new configuration has higher-level spells. We are interested in the difficult sorcerer case and two sets of experiments were performed. In the first set of experiments, for the first 500 episodes, the single optimal strategy is to always cast fireball, since no defensive spell is provided that is effective against the fireball, and both ALeRT-AM and DS-B find the strategy quickly, resulting in a tie. DS-B has a slightly higher winning rate due to the epsilon-greedy exploratory actions of ALeRT-AM and the fact that in this first phase, no opponent modeling is necessary, since there is a single optimal strategy. After gaining a new defensive spell against the fireball at episode 501, there is no longer a single optimal strategy. In a winning strategy, the best next action depends on the next action of the opponent. The agent model is able to model its opponent accurately enough so that its success continuously improves and by the end of episode 1000, ALeRT-AM is able to defeat DS-B at a winning rate of approximately 80%. In the second set of experiments, both the first 500 episodes and the second 500 episodes require a model of the opponent in order to plan a counter-strategy, and ALeRT-AM clearly shows its advantage over DS-B in both phases.

Observations

Although ALeRT-AM has a much larger feature space than ALeRT (with sixty-four extra features for a sorcerer, representing the pairs of eight features for the agent and eight features for the opposing agent), its performance does not suffer. In a fighter-sorcerer team, the single best strategy is to kill the opposing sorcerer first, regardless of what the opponent is trying to do. In this case, ALeRT-AM performs as well as ALeRT in terms of winning percentage against all opponents we have experimented with. Against the default static NWN strategy, both ALeRT and ALeRT-AM perform exceptionally well, quickly discovering a counter-strategy to the opposing static strategy, if one can be found, as is the case with the sorcerer team and the fighter-sorcerer team. We have also shown that ALeRT-AM does not lose the adaptivity of ALeRT in a changing environment.

In the sorcerer team, where the strategy of the sorcerer depends heavily on the strategy of the opposing agent, ALeRT-AM has shown its advantages. Both against the rule-based learning agent DS-B and the agent running the original ALeRT algorithm, ALeRT-AM emerges victorious and it is able to keep its victory by quickly adapting to the opposing agent's strategy.

When implementing the RL algorithm for a game designer, the additional features required for agent

modeling will not cause additional work. All features can be automatically generated from the set of actions and the set of game states. The set of actions and the set of game states are simple and intuitive, and they can be reused across different characters in story-based games.

Conclusions

We modified the general purpose learning algorithm ALeRT to incorporate agent modeling and evaluated the new algorithm by modeling opponent agents in combat. While ALeRT was able to find a winning strategy quickly, when the winning strategy depended only on the actions of the agent itself, it did not give consistent results when the winning strategy included actions that are dependent on the opponent's actions. ALeRT-AM corrected this problem by constructing a model for the opponent and using this model to predict the best action it could take against that opponent. The ALeRT-AM algorithm exhibits the same kind of adaptability that ALeRT exhibits when the environment changes. Although the experiments focused on modeling opponent agents, the same algorithm can be used to predict the actions of allied agents to improve cooperative actions. The algorithm can also be applied beyond combat since the designer of the game needs only to supply a set of states for the game, and a set of legal actions the NPCs can take and a reward function. An initial learning phase can be done off-line so that the initial behaviors are reasonable and as the game progresses, the NPCs will be able to adapt quickly to the changing environment. With a typical story-based game consisting of hundreds of NPCs, the learning experience can also be shared among NPCs of the same type, thus greatly reducing the time required to adapt for the learning algorithm.

As future work, experiments can be conducted with human players to get an evaluation of the human conception of the learning agents. The player would control a character in the game and a companion would be available to the player. Two sets of experiments would be run, one with an ALeRT-AM learning companion and one with a different companion. Players would be asked which one they prefer. Ultimately, the goal of the learning algorithm is to provide NPCs with more realistic behaviors to present a better gaming experience for players.

Acknowledgements

This research was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) and the Alberta's Informatics Circle of Research Excellence (iCORE). We thank the three anonymous reviewers for their feedbacks and suggestions. We also thank the rest of the ScriptEase research team, past and present.

References

Billings, D., Davidson, A., Schaeffer, J. and Szafron, D. 2002. The challenge of poker. *Artificial Intelligence,* 134 (1-2), 201-240.

Cutumisu, M. 2009. Using Behavior Patterns to Generate Scripts for Computer Role-Playing Games. PhD thesis, University of Alberta.

Cutumisu, M., Szafron, D., Bowling, M., Sutton, R.S. 2008. Agent Learning using Action-Dependent Learning Rates in Computer Role-Playing Games. *4th Annual Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-08)*, 22-29.

NWN. 2009. http://nwn.bioware.com.

NWN Arena. 2009. http://ticc.uvt.nl/~pspronck/nwn.html.

Schadd F., Bakkes, S., and Spronck, P. 2007. Opponent Modeling in Real-Time Strategy Games. *8th International Conference on Intelligent Games and Simulation (GAME-ON 2007),* 61-68.

ScriptEase. 2009. http://www.cs.ualberta.ca/~script/.

Sharma, M., Holmes, M., Santamaria, J.C., Irani, A., Isbell, C., Ram, A. 2007. Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL. In *International Joint Conference on Artificial Intelligence (IJCAI-07),* 1041-1046.

Smith, M., Lee-Urban, S., Muñoz-Avila, H. 2007. RETALIATE: Learning Winning Policies in First-Person Shooter Games. In *Proceedings of the Nineteenth Innovative Applications of Artificial Intelligence Conference (IAAI-07)*, 1801-1806.

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. 2003. Online Adaptation of Computer Game Opponent AI. *Proceedings of the 15th Belgium-Netherlands Conference on AI*. 291-298.

Sutton, R.S., and Barto, A.G. eds. 1998. *Reinforcement Learning: An Introduction*. Cambridge, Mass.: MIT Press.

Sutton, R.S. 1992. Adapting Bias by Gradient Descent: An Incremental Version of Delta-Bar-Delta. In *Proceedings of the 10th National Conference on AI*, 171-176.

Szita, I., Ponsen, M., and Spronck, P. 2008. Keeping Adaptive Game AI Interesting. In *Proceedings of CGAMES 2008,* 70-74.

Timuri, T., Spronck, P., and van den Herik, J. 2007. Automatic Rule Ordering for Dynamic Scripting. In *Proceedings of the 3rd AIIDE Conference*, 49-54, Palo Alto, Calif.: AAAI Press.

Uther, W., and Veloso, M. 1997. Adversarial reinforcement learning. Tech. rep., Carnegie Mellon University. Unpublished.