Learning Companion Behaviors Using Reinforcement Learning in Games

AmirAli Sharifi, Richard Zhao and Duane Szafron

Department of Computing Science, University of Alberta
Edmonton, AB,
CANADA T6G 2H1
asharifi@ualberta.ca, rxzhao@ualberta.ca, dszafron@ualberta.ca

Abstract

Our goal is to enable Non Player Characters (NPC) in computer games to exhibit natural behaviors. The quality of behaviors affects the game experience especially in story-based games, which rely on player-NPC interactions. We used Reinforcement Learning to enable NPC companions to develop preferences for actions. We implemented our RL technique in BioWare Corp.'s Neverwinter Nights. Our experiments evaluate an NPC companion's behaviors regarding traps. Our method enables NPCs to rapidly learn reasonable behaviors and adapt to changes in the game.

Introduction

Game players have growing expectations about intelligent behavior of agents in story-based games. Non-Player Characters (NPC) lead the Player Character (PC) through the story. The behaviors of NPCs are usually scripted manually, which results in repetitive and artificial looking behaviors. Since there are usually many NPCs in story-based games, the cost of scripting complex behaviors for each NPC is not financially viable. Some researchers (Spronck et al. 2006) and games companies (Booth 2009) have started using learning techniques to generate more realistic and complex behaviors for NPCs. Reinforcement Learning (RL) is a popular adaptive learning technique.

In RL there are several mechanisms and algorithms to learn policies that identify high-reward behaviors for an agent in any given context, by maximizing the expected reward. We can use these techniques to learn policies for various types of agent behaviors in story-based games to derive more natural NPC behaviors.

We use RL to derive appropriate behaviors for a companion NPC that accompanies the PC during the story. For illustration, we investigate the actions a companion selects immediately after detecting a trap and after subsequent verbal communication with the PC. After detecting a trap, an NPC can: disarm it, mark its location, inform the PC about it, or do nothing. The first two actions may cause physical damage to the NPC and/or PC if the action fails critically. The second two actions may cause physical damage if the PC subsequently triggers the trap. The choice of action should depend on the NPC's past experience regarding traps and on how much the NPC cares about the PC. After this initial NPC action, the PC may provide verbal feedback on the action such as "Good job disarming that trap" or "It exploded, but good try." or "Marking was a bad idea, disarm it." Next, the PC may also tell the NPC what further action to take on that specific trap, such as: "You marked it, now disarm it", or "OK, there is a trap, mark it's location". At this point the NPC should decide either to do what the PC asks or refuse, saying something like "Bad idea" or "Disarm it yourself". The PC can request additional actions on this particular trap and the NPC can continue to concur or refuse until the trap is disarmed or the PC decides to move on.

A second example of an NPC decision is whether to pick someone's pocket. This NPC decision depends on various parameters such as potential gain, success probability based on past experience, and again how much the NPC cares about the PC, which in this case would depend on whether the PC shares loot from previous NPC pickpocket actions. There is a set of actions to choose from and the NPC will choose an appropriate action. As in the previous example, the PC can both provide optional verbal feedback and can entreat the NPC to take a different action.

We used ScriptEase (2010) and the Sarsa(λ) algorithm (Sutton and Barto 1998) to generate learned companion behaviors in BioWare Corp.'s Neverwinter Nights (NWN) (BioWare 2010b). The NPC learns natural behaviors that adapt quickly to changes in the environment. Running many experiments in the game takes a long time, since we cannot shut off the graphics. Therefore, we wrote a simulation program that uses NWN mechanics and conducted multiple experiments to evaluate our approach.

Related Work

There have been a few attempts to use learning methods for NPC behaviors in computer games (e.g., Creatures, and Black & White). However, RL has not been popular since the learning times are often too long for the limited roles that NPCs play (Spronck et al. 2003). Some hybrid methods have been proposed such as a dynamic rule-base (Spronck et al. 2006), where a pre-built set of rules is maintained for each type of NPC. A subset of the existing rule-base is chosen for each NPC and after observing a complete set of actions, the value function for choosing a new subset of rules is updated. However, this method still requires effort to make a logical and ordered rule-base (Timuri et al. 2007) and its adaptation is limited once a policy has been learned (Cutumisu et al. 2008). Sharma et al. (2007) used a hybrid of RL and planning to create high level strategic plans in real time strategy games. Smith, et al. (2008) used the Q-Learning algorithm to learn high-level team strategies in first person shooter games.

Most progress on using RL in games has been on learning high-level strategies rather than behaviors for individual NPCs. However, Cutumisu et al. (2008) and Zhao and Szafron (2009) have shown that individual NPCs can learn behaviors using variations of the Sarsa(λ) algorithm called ALeRT, and ALeRT-AM. These algorithms have dynamic learning rates that support the fast changing environments found in video games. However, these algorithms were only evaluated for combat, where relatively more training episodes are available than the situation for most non-combat behaviors. In addition, the reward function used in combat is not suitable for non-combat situations. Merrick and Maher (2009) have additional references to research on character learning in games.

We show that RL can be used to learn non-combat NPC behaviors. Our goal is to devise a responsive learning system that produces natural behaviors for NPCs, based on their own motivations.

Algorithm

We use Sarsa(λ), an online single agent RL algorithm (Sutton and Barto 1998) with function approximation and binary features to learn agent behaviors. On each time step, the agent performs an action and observes the consequences. Sarsa(λ) maintains an approximation of the optimal action-value function, $Q^*(s,a)$. For each pair of states and actions, it represents the value of taking action a in state s and is used to select the best action to perform in the current state according to a learned policy π. Policy π is a mapping of each pair of state-actions (a, s) to the probability of performing action a in state s. The corresponding action-value function for policy π, $Q^\pi(s,a)$, estimates the expected long-term reward for performing action a in state s and following policy π afterwards. In Sarsa(λ) we start in state s_1, take action a_1, and observe reward r_1 and state s_2. We select action a_2 according to our policy, π, and then update our approximation, $Q^\pi(s,a)$, hence the name Sarsa (state-action-reward-state-action).

Sarsa(λ) is an on-policy algorithm so it learns from experience. Sarsa(λ) uses a temporal-difference updating method in which α is the learning rate, γ is a discount factor, and λ, the trace-decay, propagates rewards for the latest actions to previous actions using the eligibility trace matrix, denoted $e(s, a)$. These parameters can be tuned to adjust the responsiveness of learning:

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

Double Reward System

In the conventional Sarsa(λ) algorithm, updates to the approximation of $Q(s,a)$ are done once for each learning step. However, for the companion-learning problem there are two potential sources of reward in each step. The first reward is the immediate reward, r_i, the NPC observes from the environment after taking an action. It reflects the immediate consequences of the NPC action. If the NPC encounters a critical failure while disarming a trap the NPC takes damage. If the NPC is successful, experience points (XP) are gained. Each consequence must be considered to build an effective reward function, which will inform the NPC's preferences for performing future actions. The second reward is a delayed reward based on feedback of the PC. This feedback could be verbal or physical, such as a gift from the PC. We focus on verbal rewards, which we denote r_v. The delayed reward may or may not materialize since the PC may not provide a verbal reward or gift. As a result, our single update to $Q(s,a)$ is always based on one reward, r_i or two rewards, r_i and r_v.

This technique is different than performing two complete Sarsa(λ) steps, since the NPC does not perform an action between the immediate and delayed reward. If a delayed reward occurs, we perform an update immediately. If no delayed reward occurs before the next action selection is triggered, we update without a delayed reward, so our algorithm is actually a Sarsa/Sarrsa algorithm. The verbal reward function should take into account how much the NPC currently cares about the PC. If the NPC does not care about the PC, the verbal reward is discounted.

GESM Action Selection Policy

We need an action selection policy to select actions based on the learned values of $Q(s,a)$ for the current state and all available actions. Several action selection policies are widely used. The simplest selection policy is a greedy policy, where the NPC selects the action with the highest $Q(s,a)$ for the current state. This policy is good in a stationary environment when the optimal policy, π^*, has already been learned (Sutton and Barto 1998). A simple alternative is the ε-greedy policy, which selects the action with highest $Q(s,a)$ with probability $1- \varepsilon$ (exploitation) and selects a random action with a probability ε (exploration). Another alternative policy is the Softmax Policy (Sutton and Barto 1998), where the $Q(s,a)$ values for all actions are transformed into probabilities. Actions with higher *Q(s,a)* values have greater probability of being selected.

Each policy has advantages and disadvantages. After gathering empirical results with both the ε -greedy and Softmax policies, we decided to combine them into a new policy we call Greedy Epsilon Softmax or GESM. This policy selects the action with highest *Q(s,a)* value with probability of 1-ε, and uses Softmax with probability of ε, excluding the best action during exploration. We avoid the random exploration of the normal ε -greedy algorithm, since in this application the second best action is usually more appropriate than the rest of the actions. We used a Gibbs distribution for the Softmax part of the policy (where *n* is the number of distinct actions, and *τ* is the temperature parameter controlling the scale of differences in selection probabilities):

$$\frac{e^{Q_t(a)/\tau}}{\sum_{b=1}^{n} e^{Q_t(b)/\tau}}$$

NWN Implementation

One of the responsibilities of companions in story-based games may be to detect and disarm traps. Companions in NWN are scripted manually. They wait for the PC's command instead of initiating behaviors, and they always obey. If the PC tells an NPC to disarm a trap, the NPC always attempts to disarm it regardless of damage. Such NPCs do not look intelligent in the player's eyes. In Dragon Age, a companion can disarm a trap when the player takes on the persona of the companion. In this case, the companion is also forced to disarm the trap. However, an NPC using our learning system will develop preferences for actions after a short period of time and will decide what to do about a trap after detecting it and how to respond to the PC's orders about detected traps.

Figure 1 shows the NWN area we built for our traps experiments. The PC and the companion NPC go counter clockwise around the castle starting from the point designated by Label 3. As they pass the resetting trigger for the first time, they start the experiment and the traps get reset each time they walk over the resetting trigger.

Our learning task for traps is non-episodic, so the NPC can continue to learn as long as there are traps available. A learning step consists of deciding the next action for a trap that has been detected or deciding whether to obey an order from the PC, then performing the selected action and receiving the rewards. The sets of actions available to the companion NPC after detecting a trap or receiving an order from the PC are shown in Table 1.

We model the difficulty-based traps in NWN by three trap categories, easy, medium, and hard. These are not absolute difficulty categories, but instead are relative to the NPCs skill level at some point in the game. In other words, late in the game a trap that we label easy relative to NPC skill could actually be more difficult than a trap we label hard near the start of the game, when the NPC skill level is low. Disarming or marking a trap can result in success, failure, or critical failure. Failure causes no damage and the trap remains active. Critical failure damages the NPC and the trap remains armed. The amount of damage is a range of percentages of maximum hit points (HP) that depends on the trap category. Table 2 shows the properties of actions, their critical failure damage and their success/fail/critical fail probabilities relative to trap difficulty that we used to model traps.

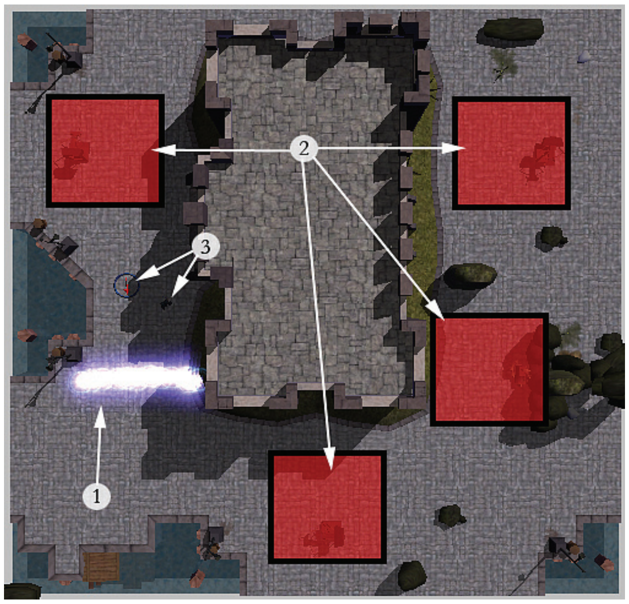

Figure 1 - Traps Area. Label 1 is the resetting trigger, Label 2 is the traps. Label 3 denotes the starting point of PC and NPC.

Decision Making Trigger	**Trap Detection**	**PC orders to Disarm**	**PC orders to Mark**
Possible NPC Actions	Nothing	Disarm	Mark
	Disarm	Refuse	Refuse
	Mark	N/A	
	Inform		

Table 1- Available NPC actions and decision-making triggers

Trap Type	**Outcome \ Action**	**Disarm**	**Mark**	**Inform**	**Nothing**	**Refuse**
Easy 5-10% damage	Success	80%	100%	100%	100%	100%
	Fail	10%	0%	0%	0%	0%
	Critical Failure	10%	0%	0%	0%	0%
Medium 10%-20% damage	Success	50%	70%	100%	100%	100%
	Fail	10%	10%	0%	0%	0%
	Critical Failure	40%	20%	0%	0%	0%
Hard 20%-30% damage	Success	10%	50%	100%	100%	100%
	Fail	10%	10%	0%	0%	0%
	Critical Failure	80%	40%	0%	0%	0%

Table 2 - Action outcomes relative to trap difficulty

To define the reward function, it is necessary to discuss an important concept. The NPC's approval of the PC, denoted *A∈ [0,1]*, plays an important role in the reward. *A* changes as the NPC observes the consequences of actions that are based on PC orders. Dragon Age (BioWare 2010a) displays such an approval as a value between -100 and 100. Similarly, we display this approval as a value between 0 and 100. Therefore, changes in *A*, denoted *ΔA*, are made in discrete steps with a minimum step size of *1/100*. To mirror the real world, *A* does not change linearly. If the NPC currently has a low *A*, it is harder for the PC to gain trust and if *A* is high, the NPC can forgive some mistakes. This means that changes in *A* are smaller when *A* is low (near 0) or high (near 1) and larger when *A* is in the middle (near 0.5). Therefore, we calculate *ΔA* (the change in A) using the following parabolic function (*A* in [0,1]), so that *A* changes most rapidly in the middle of its interval (*ΔA= 5/100*) and most slowly at the ends (*ΔA= 1/100*):

$$\Delta A = (-16A^2 + 16A + 1)/100$$

Note that the NPC's approval of the PC (*A*) may change for other reasons during the game and affect the NPC's willingness to obey trap-related orders.

The immediate reward is parameterized based on the action and the action outcome. Table 3 shows the parameters used to create the reward function. The reward has two positive components, one negative component and one component whose sign is variable:

$$\text{Reward} = \text{XPR} + \text{TRR} + \text{IDR} + \text{AR}$$

Parameter	Formula or Value
XPR	+0.2
TRR	A*(Average Trap Damage)*(Revelation Factor RF) RF = 0 for nothing, 0.3 for inform, 0.8 for mark and 1.0 for disarm
IDR	-(1-A)*(Actual Critical Failure Damage)
AR	AF*A
AF	0.35

Table 3 - Required parameters for building the reward function.

XPR is the reward that represents XP gained by successfully disarming a trap, so it is zero in other situations. XPR is constant and determined by the relative damage a character must usually take to accumulate XP.

TRR represents the reward for revealing the existence of a trap. It accounts for reduction in future damage from an armed trap by allowing the PC to avoid it. The total value of TRR that can be obtained for a single trap is the average trap damage, discounted by the approval A. However, this reward can be earned in stages. None of this reward is earned if the NPC does nothing. If the NPC marks a non-revealed trap, the reward is 0.8 (revelation factor of marking) of this total. If the NPC only informs the PC that a trap exists, the reward is only 0.3 (revelation factor of informing) of this total, since there is a higher chance that the PC will get damaged by not knowing the exact location of the trap. However, if the NPC first informs and then marks a trap, the reward for informing already accounts for 0.3 of the total so the reward for marking is (0.8 - 0.3) of the total.

IDR is the negative reward that represents damage taken by the NPC for a critical failure, while disarming or marking a trap. IDR is discounted based on the NPC's approval of the PC, since the NPC may be willing to take damage for a well-liked PC.

AR is the reward that represents positive or negative verbal feedback from the PC. AR depends both on the approval, A, and a scaling factor, AF. The scaling factor is necessary to combine an approval score between 0 and 1 with damage rewards and the XP reward and *A* is used since the amount the NPC cares about the PC approval is dependent on how much the NPC approves of the PC.

The feature vector used for learning contains 5 binary features that represent the state of the environment: 1) the NPC's approval of the PC is higher than 0.5, 2) the damage to an NPC from a critical failure is greater than 10% of the NPC's maximum hit-points, 3) the NPC's skill rank of disarming traps is greater than the NPC's level, 4) the NPC's dexterity skill modifier is greater than 3 and 5) a constant 1 for normalization. We have used linear function approximation, which means we calculate *Q(s,a)* as the dot product of the learned weight vector of an action, ω_a^T (initialized to zero) and the binary feature vector Φ_s. This small feature vector seems to capture all of the necessary information for realistic trap-disarming behavior. Naturally, the feature vector would have some other components for other learning activities. However, there will likely be some shared features such as the NPC's approval of the PC.

Simulation

In order to evaluate our learning system we needed to calculate the *Q(s,a)* averages over a large number of runs. It is impossible to shut off the graphics in the game and the time it takes for the PC to give orders or feedback and the NPC to respond would make the experiments very time consuming. Therefore, we created a simulator program that captures the complexity of traps in story-based games. It generalizes many of the concepts in NWN, such as traps of varying difficulty with respect to NPC skill, critical failure damage, experience point rewards and the problem of leaving un-disarmed traps that can trigger later. All the parameters we need to set in the game are available in the simulator and they work with the game mechanics. The simulator also enabled us to model different common PCs by setting parameters. After running the simulator, we can transfer the learned weight vector to an NPC in NWN and observe the NPC behaviors that are generated by that weight vector. Naturally, as the NPC interacts with a PC, the weights change as the NPC learns in the NWN environment. This is in contrast to the default rogue companion in NWN that always obeys the PC.

Experiments and Evaluation

We conducted many experiments with a variable number of traps and variable trap difficulty. We fixed the learning parameters to $\alpha = 0.1$, $\gamma = 0.95$ $\lambda = 0$, and the policy parameters to $\varepsilon = 0.3$ and $\tau = 0.2$. Each graph in this section is the average of 500 independent learning experiments, where the learning weights and other parameters are reset before each experiment.

An NPC starts with zero knowledge of the traps, knowing only the set of legal actions. We modeled common PC behaviors using four different PC models. An *independent* PC wants the NPC to be independent. This PC never gives orders to the NPC. The *rogue* PC wants to personally disarm all the traps. The *selfish* PC wants the NPC to disarm all the traps, no matter what negative consequences occur for the NPC. The *cautious* PC cares about the NPC and tries to understand the level of the NPC's rogue skills. This PC would never order the NPC to disarm a trap if the NPC failed at the easier task of marking it. For brevity, we present only a representative subset of results. Other results were as expected and appear in Sharifi (2010). When the game is shipped, the designers do not know what the behavior of the PC will be. The player may play similarly to one of these four models, some combination of them or in any arbitrary way. The NPC learns to adapt to whatever style the PC has, even if the PC changes style during the game. The learning algorithm does not depend on these models in any way.

Our action selection policy, GESM, selects the highest action during exploitation and selects one of the other actions probabilistically based on relative state-action value during exploration. The relative scores play an important role in marking NPC preferences and contribute to the NPC's more natural behavior. The main obstacle in using RL in computer games is the speed of adaptation. In order to understand how well an NPC adapts to the changes in both the emotional environment and the physical environment, we need to test the NPC's responses to both trap difficulty changes and PC approval changes.

Figure 2 illustrates the speed of adaptation for changing trap difficulties (the physical environment). Figure 2 shows results for a *cautious* PC with high approval (0.8), while traps change from easy to hard and back to easy every 5 traps. This graph shows that as the NPC becomes aware of the danger from hard traps, marking becomes top choice and disarming becomes second choice. The cautious PC is coaching the NPC by giving verbal approval for success and disapproval for failure. This verbal approval speeds up the learning process. We do not expect this kind of cyclic trap difficulty in the game. However, we constructed this scenario specifically to validate fast adaptability. Although trap difficulty does not actually cycle in a game, it is common to face a range of trap difficulties at any point in the game.

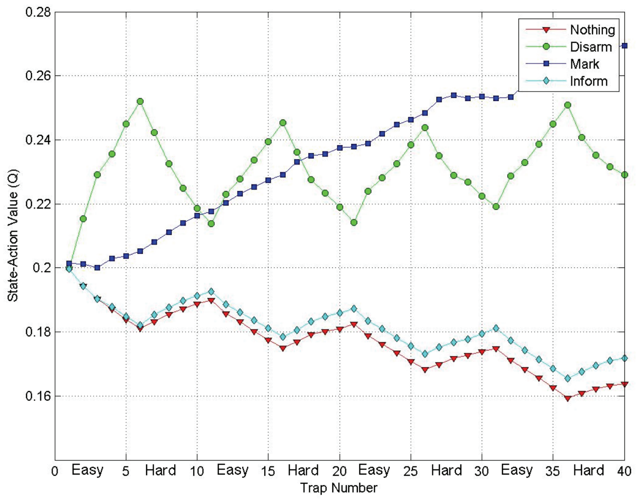

Figure 2 - Adapting to trap difficulty with a high approval cautious PC

Figure 3 shows alternating easy/hard traps for a *selfish* PC with low approval (0.2). The preferred action is based solely on the difficulty of the traps. For hard traps, the learned action preference order is to do nothing, then mark, then inform and then disarm. The NPC learns that it is best to do nothing with hard traps, since if the NPC informs the selfish PC that a trap is present, the NPC will be ordered to disarm it. For easy traps the order of preferences is: disarm, then nothing, then mark and finally to inform. The reason the NPC prefers to disarm rather than mark is that disarming yields XP while marking does not.

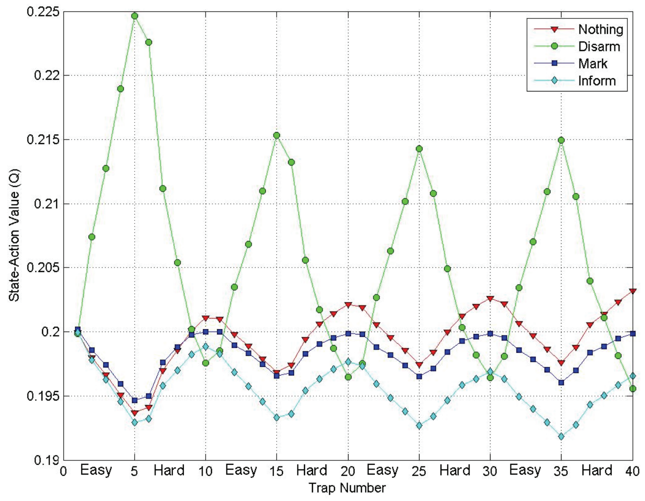

Figure 3 - Adapting to trap difficulty with a low approval selfish PC

Figure 4 illustrates the NPC responses to PC commands. It is for the same easy/hard traps, low approval selfish PC experiment shown in Figure 3. It shows what the NPC would do in response to being commanded to disarm a trap. The NPC is quite willing to disarm easy traps to earn the XP. For hard traps, the NPC learns to refuse.

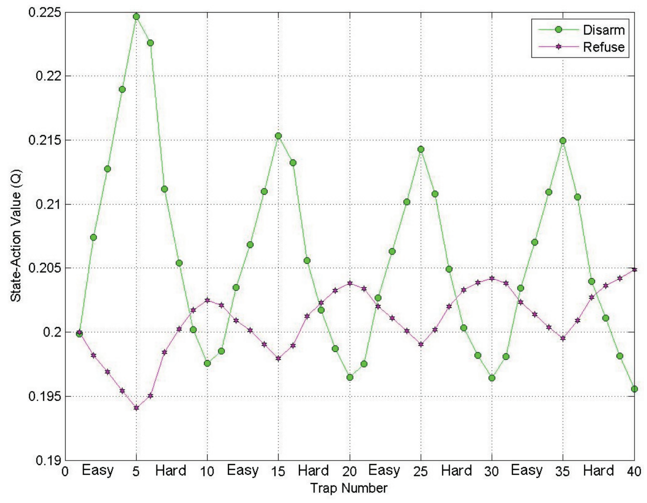

Figure 4 –Command action Adaption to trap difficulty for a low approval of a selfish PC

Figure 5 shows alternating easy/hard traps for a *rogue* PC with low approval. A rogue PC wants the NPC to only inform about traps so that the PC can disarm/mark all the traps personally. After 7 traps, the NPC learns to inform the PC about all traps. Since XP for traps is shared between both players, no matter who disarms them, the NPC is fine with allowing the PC to take all the risks. However, it takes 7 traps to convince the NPC, since the first 5 traps are easy and the low approval means that the NPC does not respect the PC commands. Once the NPC realizes that there are some hard traps (traps 6 and 7), the PC is allowed to disarm/mark all the rest. The results are similar for a high approval PC, except that it only takes a single trap for the PC to convince the NPC, due to the high approval.

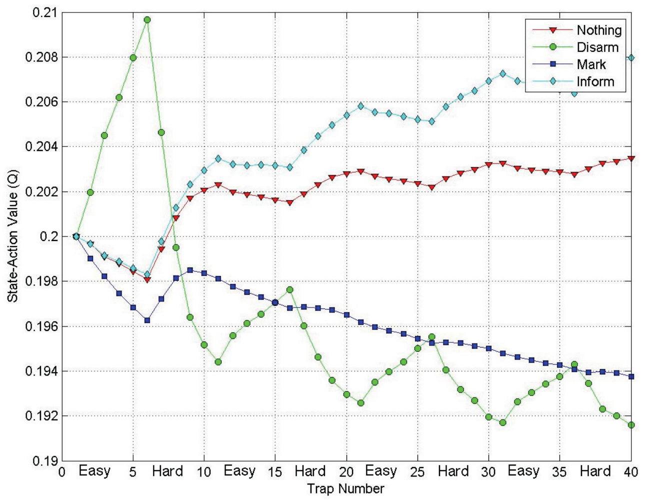

Figure 5 - Adapting to trap difficulty with a low approval rogue PC

Figure 6 shows tests with 40 traps with fixed hard difficulty. We use a cautious PC that starts with a low approval (0.2) and then switch to high approval (0.8) after 5 traps. We then switch back and forth for every 5 traps. This simulates changes in approval due to other events occurring in the game that drive changes in the emotional state of the NPC. We want to see if the NPC behavior changes accordingly. Since the traps are hard, the first choice is to inform the PC. Since the PC is cautious, the NPC is not commanded to disarm. When the approval is high, the second choice is to mark the traps to prevent the PC from being damaged. However, when the approval is low, the second choice is to do nothing since the NPC does not care about damage to the PC from an unmarked trap. Note that our GESM policy is to explore 30% of the time and in the exploration case, the first choice is never selected. The second choice (mark or nothing) is then selected most of the time, since $\tau = 0.2$.

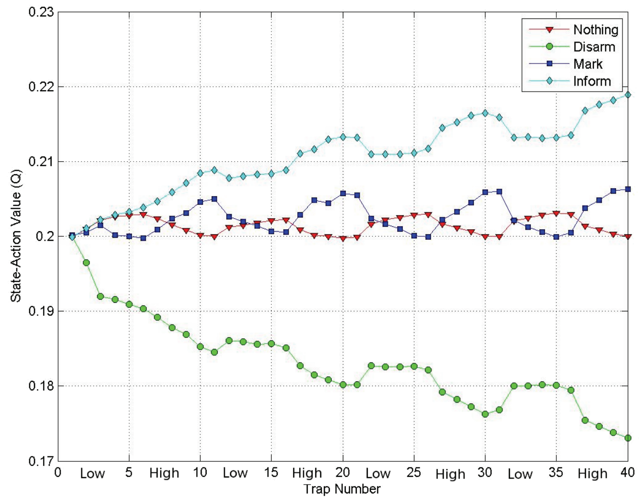

Figure 6 - Adapting to Low and High approval changes of a cautious PC with hard traps

With easy traps (not shown) the NPC disarms all traps as first choice since the XP is desirable and there is a very little chance of damage.

Conclusion

Techniques such as behavior trees (Isla 2005) and rule based (Spronck et al. 2006) methods have been used in games. Recently, RL has been used to enable NPCs to learn behavior strategies for combat scenarios (Cutumisu et al.) (Zhao and Szafron 2009). However, there have been no successful attempts to enable companion NPCs to learn more flexible behaviors that are responsive to changes in emotional and physical state.

We created a mechanism that enables adaptive companion NPC behavior. Players have individual goals, treat their companions differently and have varying companion expectations in different game situations. Our experiments show that an NPC using our learning mechanism can respond differently based on NPC approval of the PC and the changing environmental circumstances (trap difficulty).

When RL is applied to the behavior of companion agents, the companion may decide to do things that are not usually available in hard-coded behaviors. These behaviors are the ones that make the NPC's behavior more natural ("Disarm it yourself"). For example, sometimes the NPC might decide to remain silent about a detected trap, since the NPC suspects that the PC will give a disarm order if the PC is informed about it.

The mechanism that we created is not limited to trap actions. For example, this mechanism can be used by the NPC to decide when to pick pockets. The NPC would learn from experience whether picking pocket is beneficial for the party or not, by considering the changing environment such as the PC's generosity towards the companion NPC, the type of target, and the evaluated risk of detection. Companion NPCs using adaptive learning systems exhibit more realistic behaviors, which can be specifically tuned, controlled, and limited by game designers.

References

BioWare 2010a. Dragon Age. http://dragonage.bioware.com.

BioWare. 2010b. Neverwinter Nights. 2010b. http://nwn.bioware .com.

Booth, M. 2009. The AI Systems of Left 4 Dead, AIIDE 2009 Keynote (http://www.valvesoftware.com/publications/2009/ai_systems_of_l4d_mike_booth.pdf).

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.

Isla, D. 2005. Handling complexity in the Halo 2 AI. In *Proceedings of Game Developers Conference*, San Francisco.

Merrick, K., and Maher, M. 2009. Motivated Reinforcement Learning. Berlin. Springer-Verlag.

ScriptEase. 2010. http://webdocs.cs.ualberta.ca/~script/.

Sharifi, A. 2010. Generating Adaptive Companion Behaviours Using Reinforcement Learning In Games. MSc thesis, University of Alberta, Edmonton, Canada.

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*, 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*, 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.

Timuri, T., Spronck, P., and van den Herik, J. 2007. Automatic Rule Ordering for Dynamic Scripting. *3rd Annual Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-07)*, 49-54.

Watkins, C.J.C.H. 1989. Learning from Delayed Rewards. PhD thesis, Cambridge University, Cambridge, England.

Zhao, R., Szafron, D. 2009. Learning Character Behavior Using Agent Modeling in Games. *5th Annual Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE-09)*, 179-185.