The question of whether computers can think is like the question of whether submarines can swim.

– Edsger W. Dijkstra.

University of Alberta

GENERATING ADAPTIVE COMPANION BEHAVIORS USING REINFORCEMENT LEARNING IN GAMES

by

AmirAli Sharifi

A thesis submitted to the Faculty of Graduate Studies and Research
in partial fulfillment of the requirements for the degree of

Master of Science

Department of Computing Science

Fall 2010
Edmonton, Alberta

Examining Committee

Duane Szafron, Computing Science

Michael Bowling, Computing Science

Sean Gouglas, History & Classics

I dedicate this thesis to my mother, Sedigheh, for her endless love and support.

Abstract

Non-Player Character (NPC) behaviors in today's computer games are mostly generated from manually written scripts. The high cost of manually creating complex behaviors for each NPC to exhibit intelligence in response to every situation in the game results in NPCs with repetitive and artificial looking behaviors. The goal of this research is to enable NPCs in computer games to exhibit natural and human-like behaviors in non-combat situations. The quality of these behaviors affects the game experience especially in story-based games, which rely heavily on player-NPC interactions. Reinforcement Learning has been used in this research for BioWare Corp.'s Neverwinter Nights to learn natural-looking behaviors for companion NPCs. The proposed method enables NPCs to rapidly learn reasonable behaviors and adapt to the changes in the game environment. This research also provides a learning architecture to divide the NPC behavior into sub-behaviors and sub-tasks called decision domains.

Acknowledgements

I owe my deepest gratitude to my supervisor, Dr. Duane Szafron, for his unending support, guidance, and enthusiasm toward my research. I would also like to thank my dissertation committee, Dr. Michael Bowling and Dr. Sean Gouglas, for their invaluable comments and suggestions.

I would like to specially thank Richard Zhao, for his significant help during my research. I would also like to thank my fellow researchers, Neesha Desai, Christopher Kerr, Marcus Trenton, Yi Yang, Levi Lelis, and Sapphire Zhao as well as ScriptEase implementation team, Robin Miller, Jason Duncan, Joshua Friesen, Eric Graves, and Matthew Church. I am pleased to thank Metanat Hooshsadat. It would have been next to impossible to finish this dissertation without her help and great support.

It is a pleasure for me to thank my mother, Sedigheh Mansour, and my sisters, Negar Sharifi and Negin Sharifi, for their significant backing and encouragement through all my life. Words are not enough to express my gratitude for their endless support and believing in me.

Finally, I would like to thank everyone at the University of Alberta and the Department of Computing Science who made the completion of this thesis possible.

Table of Contents

List of Tables

List of Figures

Chapter 1

Introduction

On January 25^{th}, 1947, the patent for the first interactive video game in history, the *Cathode-Ray Tube Amusement Device*, was filed [19]. This system used 8 vacuum tubes for display, and knobs to adjust the parameters, such as speed, for a missile firing game. Games have come a long way since that time. The interactions between the player and the game are far more complicated than they used to be. There are various aspects to these interactions that make the game-play more interesting for the player. In modern games the goal is to maximize fun for the player. Believability is one of the factors that can contribute to this fun and one of the aspects of games which has a high potential to enrich this believability is Artificial Intelligence (AI). AI techniques can be used for many aspects of modern games such as path finding, selecting enemy moves in a combat, discovering good team strategies in sport games, etc. Some of these AI techniques become especially important depending on the type of the game.

In story-based games, the player takes the role of a character in the story. These games usually contain quests and missions that the player should complete in order to make progress in the game. The player is usually led through the story by the non-player characters (NPCs). The behaviors of NPCs are often scripted manually, which results in repetitive and artificial looking behaviors. Many game players have growing expectations about the intelligent behavior of agents, especially in story-based games. Since there are usually many NPCs in these games, the cost of scripting complex behaviors for each NPC is not financially viable [27]. In these games, the huge amount of interaction between the player character (PC) and the non-player characters require intelligent behaviors on the NPCs' side. Unfortunately, current "AI-controlled characters" are repetitive and machine-like so players do not get as much satisfaction from playing these games as they could if agents

exhibited more realistic behaviors.

1.1 Behaviors In Games

Non-player characters in computer games can be friendly, neutral, or hostile. The hostile characters are usually referred to as the *enemy*, *mob*, etc. In late 1970s and early 1980s video games like *Ultima* started to use friendly and neutral NPCs. In early games like *Ultima I: The Age of Darkness*[1] [8], the NPCs appeared in their simplest way. The early NPCs had very simple or random movement with almost no intelligence. Some of them could have single-line dialogues with the PC. Figure 1.1 shows that after the PC started transacting (*Ultima I* terminology for interacting) with the king, the king responded with a single line of dialogue.

Figure 1.1: A single-line dialogue in *Ultima I: The First Age of Darkness*

Later in the 1980s and 1990s, games started to have more complicated behaviors for the NPCs, such as multiple-choice dialogues with the NPCs. However, despite the advances in hardware, the improvements in NPC behaviors were very small compared to the improvements in other aspects of commercial games such as more realistic physics or graphics. There are some exceptions such as the game *Façade* [13], which was developed by researchers. In this game the player takes the role of a guest of a married couple. The player

[1] http://www.uo.com/archive/

can initiate or respond to conversations by typing sentences. The NPCs in this game act, respond, and interact with each other with a much more realistic manner than most commercial games. However, manually developing such complicated behaviors for each character in a story-based game with hundreds of characters is not currently practical because of the high cost. Recent commercial story-based games such as *Dragone Age*[2] [1], have more complicated behaviors for companion NPCs compared to their predecessors such as *Neverwinter Nights*[3] [17]. However, these behaviors are still machine-like and very predictable. For example, in both games the player can order the NPC to disarm a trap and the NPC will always obey. In *Dragon Age*, the companion NPCs have an *approval* that shows how much they like the PC. This approval can change in response to PC actions or when the PC gives a gift to the companion. Low approval can cause the companion to leave the player's party or attack the PC, while a companion with high approval might unlock some hidden quests for the PC. Although these behaviors are more realistic than the behaviors of companions in *Neverwinter Nights*, they do not reduce the amount of manual scripting required and they are static behaviors, which will soon become predictable for the player. In *Dragon Age*, the player can take control of the companion to perform actions, regardless of the approval value. Therefore the player can always force the companion to obey the PC.

In *The Elder Scrolls IV: Oblivion*[4] [18] the NPCs exhibit more complicated and natural looking behaviors than the games released in the previous ten years, such as NWN. However, these behaviors are also scripted manually for many different NPCs. These behaviors are based on a daily schedule that are set for each NPC. Consequently, sometimes some NPCs in *Oblivion* stay somewhere and stare at a wall or an empty field for 5 in-game hours [33]. The official campaign does not contain any companions. The *Oblivion* community has constructed some "mods" that add companions [3]. However, all these companions exhibit static manually-scripted behaviors.

1.2 Behavior Learning

One way to tackle the static and unnatural looking behavior problem is to use learning techniques to generate acceptable behaviors. By using learning techniques, it is possible to eliminate the need for extensive manual scripting for each single NPC in order to generate natural looking behaviors. It is also possible to eliminate the developer's need to predict

[2] http://dragonage.bioware.com
[3] http://nwn.bioware.com
[4] http://www.elderscrolls.com/games/oblivion_overview.htm

every possible situation in the game to handle it. Since the game industry wants to have complete control over NPCs and to avoid any unpredicted behavior, the use of learning techniques has been very limited in commercial games. In fact, the term Artificial Intelligence in commercial games mostly refers to using manual scripting to give the appearance of intelligent behavior. For example in *Left 4 Dead*[5] [26], when the companion NPCs fall behind the PC, instead of learning to run, they are simply teleported near the PC when they are out of view of the player. It is also common in strategy games to allow a computer player (NPC) to access private information of the PC to appear more intelligent.

Reinforcement Learning (RL) is a popular adaptive learning technique. In RL there are several mechanisms and algorithms to learn policies that identify the optimal behavior for an agent in any given context, by maximizing the expected reward. It will be shown that the optimal behavior is not necessarily what the story teller wants. However, it is possible to use these techniques to learn high-reward policies for various types of agent behaviors in story-based games to derive more natural NPC behaviors. In this research, I use RL to derive appropriate behaviors for a companion NPC that accompanies the PC during the story. An appropriate behavior can be defined as a rapidly adaptive and human-like behavior.

In story-based games, most NPCs will have a short role in the game. However, companion NPCs are the ones that have the most interactions with the PC during game play. For illustration, let's 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: attempt to disarm it, try to 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 might 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.". 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 "Forget it" 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

[5] http://www.l4d.com

NPC decision may depend 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 the best action. As in the previous example, the PC can both provide optional verbal feedback and can entreat the NPC to take a different action.

Generally speaking, there are two categories of learning that can be employed for NPCs in games. The first one is off-line learning. Off-line learning happens during the development time. The AI agents can be trained against experts, each other, etc. Using this type of learning, it is possible to learn an optimal or semi-optimal policy for the AI controlled agents in the game. After the learning process is finished the learned values will be hard-coded in the game and no more learning will happen during the game play. For example, in the *ReVolt* racing game, the AI controlled opponents are trained using genetic algorithms to find the best racing paths [11]. The second category is on-line learning. This type of learning happens during game-play time. Using this method enables the AI controlled NPCs to adapt to the changes in the dynamic environment and player behavior. The game *Black & White*[6] [32] uses on-line learning to enable the PC to train the NPC creature while playing the game, by providing positive and negative feedback. Reinforcement learning can be used in both on-line and off-line learning categories[7]. It is possible to passively train the NPCs during the development time to find a single optimal policy. This policy then can be plugged into the game and the NPC will use the learned values to pick actions later. In contrast, it is possible to employ RL techniques to actively learn from the experience in the game. This will allow the NPC to adapt to the changes in the environment and build a model of the player while the player is actually playing the game. It is also possible to employ both techniques together. The developers can train the NPCs and then when a reasonable behavior has been learned the NPC can continue learning during the game-play to adapt or fine tune the learned values for a specific player and changing game situations.

There are many challenges in using learning for AI controlled character behaviors. First a particular learning technique (Q-Learning, Sarsa(λ), Decision Trees) must be selected. Second, good training data must be created and developers must decide when the NPCs have learned enough (in off-line learning). Third, the parameters must be tuned, which can affect the final outcome of the learning and ultimately affect the game experience [29].

[6] http://lionhead.com/Games/BW/Default.aspx

[7] The terms off-line and on-line here are not referring to on-line and off-line updating in reinforcement learning context. They refer to the concept explained earlier in this paragraph.

For reinforcement learning these challenges become: selecting an algorithm, finding the relevant state and its right representation, finding the right reward function and avoiding infrequent rewards, dealing with slow learning rates, dealing with dynamic worlds, setting algorithm parameters, and balancing exploration and exploitation [29].

ScriptEase [14][8] and reinforcement learning are used in this research to generate adaptive and human-like companion behaviors in BioWare Corp.'s Neverwinter Nights (NWN) [17]. Chapter 2 talks about the previous efforts in using learning for generating NPC behaviors both in research and industry. Companion NPC learning and the hypothesis of this thesis, a new approach to companion learning is discussed in Chapter 3. The results of the experiments and discussion about the results are presented in Chapter 4. Chapter 5 provides conclusions and some directions for future research.

[8] http://webdocs.cs.ualberta.ca/∼script/

Chapter 2

Related Work

ALTHOUGH most commercial games use very little of the potential of AI techniques, there are a few exceptions. AI techniques have been used for different aspects in these exceptional games. For example, the games *Left 4 Dead* and *Left 4 Dead 2* [10][26] use a technique called the "AI Director" that controls the game drama by manipulating some aspects of the game such as the music and the game pace. These games also have systems to spawn the NPCs in strategic places based on the level of drama that should be created. As a result, the game experience varies between low tension and high tension in an entertaining way and the game experience also changes each time the player plays the game.

In games where the player has a significant amount of communication with the NPCs, enhancing the NPC behaviors can improve the game-play. However, in games like *Left 4 Dead*, despite the aforementioned efforts applied to the "AI Director", the "AI-controlled" companions use static and deterministic behaviors generated from scripts. These scripts check for different game situations and generate a static behavior based on the encountered situations. There are two approaches to the NPC behaviors in games. One approach is to use static behaviors like *Left 4 Dead*, *Oblivion* [18], and Neverwinter Nights [17]. The second approach is to incorporate learning to generate adaptive and more natural-looking behaviors.

Some researchers such as Spronck [23] and some games companies [32] have started using learning techniques to generate more realistic and complex behaviors for NPCs. There are a very limited number of commercial games that use learning techniques to generate NPC behaviors. The game *Black & White* [32] is one of the games that uses learning for NPC behaviors. In this game, the player who takes the role of God, can train a creature to perform various actions for the player. This game uses techniques from reinforcement

learning to train the creature. If the creature performs the action desired by the player, the creature can be encouraged and otherwise it can be punished. Using this simple reward system enables the player to teach the desired behavior to the creature. For example, this creature can be trained to be good, evil, or something in between, which then lead to different actions being selected by this creature regarding various game situations. Although the creature in this game is trained by the PC to exhibit certain behaviors, the nature of this game does not demand human-like behaviors from this creature. In *Black & White* the creature is the avatar of the player in the game but is not required to display human-like behavior. Generating human-like behaviors is harder than generating behaviors for artificial creatures since players have many learned preconceptions about the behavior of human NPCs, based on real life experience. Players have fewer preconceptions about the behaviors of artificial creatures since they do not encounter them in real life.

In the *Forza Motorsport*[1] [16] driving game series, the player can train other (NPC) drivers. This game uses reinforcement learning to learn the way the player is driving. Using this technique enables the player to train other drivers and they can compete for the player's team. The goal of learning used in this game is to replicate the way the player behaves and build a model of the player's behavior. In this game the AI techniques do not generate independent behaviors for the NPCs that are based on independent motives [5]; the drivers are simply replicas of the PC. This may be fine for a racing game where team replica drivers are desired, but the technique does not apply to creating varied behaviors for companions.

Although there have been a few attempts to use learning methods for NPC behaviors in computer games, RL has not been popular since the learning times are often too long for the limited roles that NPCs play [24]. Some hybrid methods have been proposed such as a dynamic rule-base [23], 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 [28] and its adaptation is limited once a policy has been learned [6].

Conati *et al.* [4] used bayesian networks to give adaptive feedback to the player through an NPC in educational games. The game is for evaluating the knowledge of the students about factorization. The agent gives more detailed help if the player makes more mistakes. The agent in this model is trying to understand the players level of knowledge about factorization in order to provide the player with more constructive and insightful comments.

[1] http://forzamotorsport.net

Schrum *et al.* [20] used a combination of Neural Networks and Genetic algorithms called *Neuroevolution* to learn multiple policies for a group of NPCs in combats. The team of NPCs learn different tactics (modes) for the combat depending on the game situation. For example, a different policy is used when the NPCs are behind the player than the policy used for the situation in which the NPCs are in front of the player. This research learns high level tactics for combat and does not learn individual behaviors. Although this approach may be fine for team tactics, it is not clear this approach will produce interesting behaviors for individual companions.

Sharma *et al.* [21] used a hybrid of RL and Case-Based Reasoning (CBR) to create high level strategic plans in real time strategy games. CBR provides a way to solve new problems using reasoning from previously seen problems. This paper provides a multi-layered architecture for task decomposition that enable agents to learn generalized tactics which can be used later in different instances of the problem. These problems should have the same characteristics. Since this type of learning focuses on better team work, it does not tackle the individual NPC behavior problem.

Smith, *et al.* [22] used a variant of the Q-Learning algorithm [30] called *RETALIATE* to learn high level team strategies in first person shooter games. Their work uses NPCs with static scripts as team members and individual behaviors are not learned. Wender *et al.* [31] used Q-Learning for city placement selection tasks in the game *Civilization IV*, in contrast to the complex manual scripting that is used in the original game. Hussain *et al.* [7] have used genetic algorithms to learn the movement tactics for a group of NPCs in Neverwinter Nights.

Merrick *et al.* [15] used reinforcement learning to generate behaviors for the NPCs in massively multi-player online role playing games (MMORPG). They have used a technique called Motivated Reinforcement Learning (MRL) to generate behaviors by providing motivations for those behaviors. By using this technique, they make the NPCs move around based on motivations and develop skills and change behavior. Although the behavior these NPCs exhibit are better than the statically scripted behaviors, there are some issues that must be addressed. First, the control required by the game developers over important NPCs or NPCs with recurring roles cannot be generated by their technique. Second, the time required for the NPC to learn a behavior is often very long. That is why this technique is used for the MMORPG games in which the game environment exists persistently. Third, judging by the explanation and examples in the paper, the companion NPCs cannot use this method because of the constraints in their behaviors. For example, a companion NPC needs

to adapt to the changing environment very fast (at least within a few number of steps), while the NPCs in this system might take 5 hours to adapt to the changed environment which is not applicable for companions in normal games.

I believe most progress on using RL in games has been on learning high-level strategies rather than behaviors for individual NPCs. However, Cutumisu *et al.* [6] and Zhao *et al.* [34] have shown that individual NPCs can learn behaviors using variations of the Sarsa(λ) [25] algorithm called ALeRT [6], and ALeRT-AM [34]. These algorithms have dynamic learning rates that support the fast changing environments found in video games. These algorithms keep track of the trend of states in the game and increase the learning and exploration rates separately (using separate measures) when they detect a significant change in this trend. However, these algorithms were only evaluated for combat, where relatively more training episodes are available than the situation for most non-combat behaviors. Transition from combat situations to non-combat situations is non-trivial. The reward function used in combat, which is +1 for winning and -1 for losing the combat, is not suitable for non-combat situations. In fact, building a suitable reward function in non-combat situations is the foundation of motivating the companion NPC to exhibit a human-like behavior. In non-combat situations a general architecture should be devised to define the boundaries and overlaps of different learning tasks, while in combat the task is already well-defined. The same state space can be used for different combat situations while in non-combat situations the state space for each learning domain can be completely different. This research shows that RL can be used to learn non-combat NPC behaviors. The goal is to devise a responsive learning system that produces natural and human-looking behaviors for NPCs, based on their own motivations.

Cutumisu in [12] describes a list qualities that the behavior of an NPC should possess. The goal of the learning approach to the companion NPC behaviors in this research is to realize those qualities to exhibit human-like behaviors:

- Adaptability
- Clarity/Consistency/Intentionality
- Effectiveness
- Robustness
- Variety
- Autonomy
- Alertness
- Interactivity

- Reusability
- Scalability

Chapter 3

Companion Learning

As we can easily observe by comparing old and new games, an emerging trend in computer games is to make the game as immersive as possible. Game companies and developers use various techniques to achieve this goal. One way is to use amazing graphics. Another way is to focus on more realistic physics engines. Another approach is to make the AI controlled characters as believable as possible. Believable characters are especially important in story-based computer games on the grounds that those AI controlled characters are responsible for making the players believe what they witness in the game. We have all seen movies based on great stories but with really bad acting. The same situation can happen with computer games. No matter how great the graphics, physics, and game story is, if the game characters are acting very artificially and repetitively they can damage the story in the first place and consequently damage the player's game play experience.

Some AI-controlled non-player characters (NPCs) in computer games are more involved in building game experience in comparison with others. This degree of involvement can be measured by the time they spend interacting with the player character (PC) to fulfill their role in story. For example, the amount of time a bartender in a tavern in the game spends creating the game experience can be dependent on the amount of time the PC spends in that tavern. However, that bartender might or might not have an important role in the story. In most story-based computer games, there are some NPCs that follow the PC and help the PC to perform assigned tasks or to complete quests. The type of these "companion" NPCs can cover a wide range such as animals, robots, humans, demons, etc. Among all these types of NPCs there is at least one common behavior which all of them share: they interact with the PC more than other NPCs.

Currently, these interactions are mostly in the form of orders from the PC to the com-

panion. The main problem with these PC-companion interactions is that they are one-way interactions, which means that the companion is like a slave to the PC and cannot have an independent personality. One way to provide them with an independent personality is to allow these agents to learn.

3.1 Behavior Learning for Agents

As we have already discussed, the cost of implementing complex behaviors for every single AI-controlled character in game is not financially viable. The downside of using a limited set of fixed behaviors is the repetitive and artificial-looking behavior of those characters. One solution is to derive character behaviors from very large rule-bases. Although this might result in a wider variety of potential behaviors, in practice, the selected behaviors could appear fixed and inflexible. An inflexible character exhibits a specific behavior in a certain situation, independent of previous experience.

Behavior learning enables AI-controlled game agents to learn from experience. This experience can be different for each agent based on the different game situations encountered. As a result, each agent should learn a different behavior that reflects individual experience and preferences. For example, consider a companion that gets healed by the PC every time the companion is damaged in a battle. In the future, this companion will not be hesitant to fight for the PC, since the companion expects to be healed. The expected experience points (XP) provide a motivation that compensates for the disadvantage of temporary injury. However, if the same agent realizes that the PC does not care about the companion's injuries then the gained XP may not be worth the damage. In this case, the companion should not fight for the PC. It is the hypothesis of this dissertation that if AI-controlled agents in games are equipped with human like motivations as well as learning, they can show very interesting, dynamic, flexible, and human-like behaviors.

3.1.1 Constructing the Agent

Most commercial games designers are hesitant to incorporate learning into NPC behaviors since it may interfere with character control. We have to be careful to allow the designer to maintain overall control over companion agents, while allowing NPCs to make their own decisions within the control framework. In order to accomplish this goal, we have separate learning for separate decision domains in the game, with some shared state. However, we share some common data among these decision domains.

Decision Domain	Traps			Pickpocket
Decision Event	Trap Detection	PC orders NPC to Disarm Trap	PC orders NPC to Mark Trap	Pick Pocket Candidate Detected
Possible NPC Actions	[T] Disarm	[T] Disarm		[P] Pick Pocket
	[T] Mark		[T] Mark	[P] Evaluate Candidate
	[T] Inform PC			[P] Inform PC
	[T] Nothing			[P] Nothing
		[T] Refuse	[T] Refuse	

Table 3.1: Comparison of different action sets available for four different decisions. The two "[T] Disarm" actions are actually the same while "[T] Nothing" and "[P] Nothing" are independent.

Each decision is triggered by a unique event. Each "decision" event will inform the agent that there is a situation in the game for which the NPC needs to make a decision about which action to perform. Several decisions can be grouped into a single decision domain. For example, there are three decisions concerning *Traps* that are combined into the trap domain. When the companion detects a trap, the companion decides between the four actions shown in the second column of table 3.1. If the PC commands the companion to disarm a trap the companion can select either action in the third column. If the PC commands a companion to mark a trap the companion can select either action in the fourth column. Decisions in the second, third, and fourth column are all in the *Traps* domain and these decisions share a single learning mechanism. However, the decision in the fifth column is in a different decision domain, *Pickpocket*, which is initiated by the *Pick Pocket Candidate Detected* event.

Each of these decisions has its own set of actions as shown in Table 3.1. However, actions can be shared between decisions in the same decision domain. As we can see from Table 3.1, actions marked with [T] belong to the *Traps* decision domain action set. Decisions in this domain share some common actions such as "[T] Disarm", shared by the *Trap Detection* decisions and the *PC orders NPC to Disarm Trap* decision. Similarly the "[T] Refuse" action is shared by the *PC orders NPC to Disarm Trap* decision and the *PC orders NPC to Mark Trap* decision. Actions marked with [P] belong to the *Pickpocket* decision domain action set and are completely independent of actions marked with [T] even if they have similar names or effects. Thus, "[T] Disarm" is actually one action whose common "value" is learned regardless of which event activated the selection of the action. However, "[T] Nothing" means doing nothing after detecting a trap and "[P] Nothing" means doing nothing after detecting pick pocket candidate and they are completely inde-

pendent of each other. The author maintains control over all potential decisions and actions associated with them while the companion is free to dynamically learn relative selection probabilities for each desired actions.

In order to build an intelligent and adaptive companion, we need to model the companions motivations. Probably the most important motivation in a companion relationship is the NPC's like or dislike of the PC. This single attribute serves as the NPC's simple model of the PC. Using this model, the NPC remembers if the PC acts and gives orders that are in the best interest of both the PC and the NPC. If not, the NPC knows that the PC is either not aware of negative action consequences or does not care. Dragon Age [1] displays such an approval as a value between -100 and 100. I denote this simple attribute, *approval*, or more generally the "NPC's approval of the PC". This *approval* can change during the game in many different ways. For example, if the PC tells the NPC to perform an action and then while performing that action the NPC gets hurt, this *approval* will decrease since it shows the NPC that either the PC does not care about the NPC, or the PC does not understand the consequences of the given order. The *approval* can also be changed externally to a domain decision. For example, may change during conversations between the PC and companion based on similar or different philosophies.

The NPC's *approval* of the PC, denoted A, is a floating point number between 0 and 1 ($A \in \mathbb{R}, A \in [0, 1]$) with 0 meaning the NPC dislikes the PC and 1 meaning that the NPC totally approves of the PC. To mirror the changes in a person's behavior toward another person in real world, the *approval* does not change linearly. If the NPC currently has a low *approval* of the PC, it is harder for the PC to gain the trust and approval of the NPC. When the PC has the high *approval* of the NPC, the NPC can forgive some of the mistakes that the PC makes. This means that changes in the *approval* (ΔA) are smaller when the approval is either low (near 0) or high (near 1) and larger when the *approval* in the middle (near 0.5). To model this behavior, I devised a parabola (see Equation 3.1) to change the NPC's approval of the PC more in the middle of the range and to change it less as we approach the boundaries of the range. The behavior of this parabola function can be seen in Figure 3.1. The minimum value of ΔA is 0.1 at the end points of the $[0, 1]$ interval and the maximum value of the ΔA is 0.5 in the middle of the interval.

$$\Delta A = \frac{(-16A^2 + 16A + 1)}{100} \tag{3.1}$$

The *approval*, A, is universally shared among all the decision domains and can be used for non-learning decisions (such as conversation points) and plot progression and can

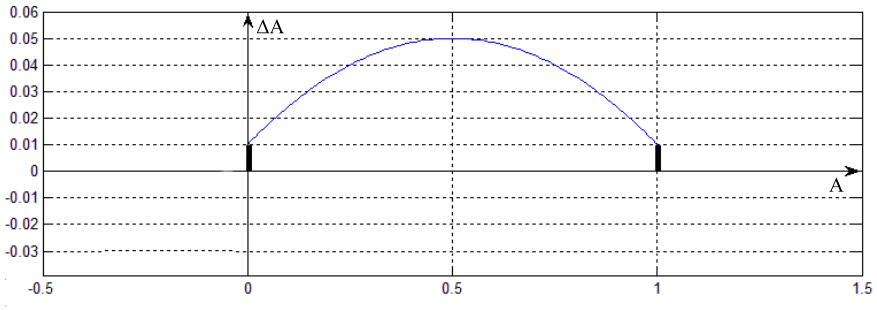

Figure 3.1: The parabola function for changing NPC's approval of the PC. The curve shows the ΔA values A in $[0, 1]$.

be changed by PC-NPC interactions. As an illustration, lets consider that the NPC has a medium-high approval of 0.5. If the NPC always encounters hard traps and the PC orders the companion to disarm them, the NPC would have a very little chance of disarming them. As a result, after getting hurt several times after obeying the PC's orders, the *approval* will go down. This low approval can have consequences outside the *Trap* decision domain. For example, assume the PC encounters some game characters whose pockets are important to be picked. If the PC orders the NPC to pick a pocket, the NPC uses the same approval, which is now low, to discount the action reward. This means that if A has a small value and picking a pocket is not beneficial for the NPC, the NPC will learn not to do it and ignore the PC. Alternatively, if the NPC's *Trap Domain* decision experience is positive, then the *approval* will be high. In this case, a subsequent PC request to pick a pocket will be more favorably considered by the companion. It is very important that the learning experience in one decision domain transfers to other decisions domains and the game in general. Learning in multiple domains cannot be completely independent.

3.2 Learning Algorithm

Reinforcement learning problems can be divided into episodic and continuing tasks. Episodic tasks have a set of non-terminal states in addition to one or more terminal states. On the other hand, continuing tasks can go on without having a terminal state. Theoretically, the task of learning for a companion agent is episodic, which means that at some point the agent might leave forever, might die, or the player may finish the game. However, companion learning has a continuous nature since each companion is unique so transferring the experience of one agent to another (between episodes) is inappropriate.

The choice of learning algorithm depends on the nature of the learning task and the required properties of that task. Some reinforcement learning algorithms are *off-line* learning algorithms such as the Monte Carlo [25] algorithm. These algorithms wait until the end of an episode to change the value estimates. On the opposite side of the learning algorithms spectrum, *on-line* algorithms update the value estimates immediately after each step. This can be very important for our task since our goal is to generate rapidly adapting behaviors for a companion agent. It would be a violation of intention to wait until the end of the game and then update the value estimates based on our experiments. A companion agent that learns appropriate behavior after the game is over is useless. Another important property of the algorithm is whether it is a control or a prediction algorithm. Since I want to enhance the agent's behavior by learning, I need an algorithm that can be used for control. In general, control algorithms such as Sarsa(λ) provide the agent with a metric required for distinguishing between high-reward and low-reward actions. The metric is a value estimate of taking each available action in a certain state. On the other hand, a prediction algorithm does not provide a metric that can be used to evaluate actions. Thus, in order to enable the agent to actually use what has been recently learned, I use a control algorithm.

I used Sarsa(λ), an online single agent reinforcement learning algorithm [25] with function approximation (see 3.4) 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 a state and an available action in that state, (s, a), the algorithm maintains the value of taking action a in state s and uses it to select the next action to perform in the current state according to a learned policy π and the employed action selection rules. Policy π is a mapping of each pair of state-actions (s, a) to the probability of performing action a in state s. The corresponding action-value function for policy π, denoted $Q^\pi(s, a)$, estimates the expected long term reward for performing action a in state s and following policy π afterwards. Sarsa(λ) starts in state s_1, takes action a_1, and observes reward r_1 and state s_2. The algorithm selects action a_2 according to our policy, π, and then updates the approximation, $Q^\pi(s, a)$, hence the name Sarsa (State-Action-Reward-State-Action).

Sarsa(λ) is an on-policy control algorithm, which means that the policy it uses for decision making is the same as the one it evaluates and improves [25]. Being a control method means that it maintains state-action values, $Q_t(s, a)$, which can be used for decision making, as opposed to prediction methods such as TD which maintain only state values, $V_t(s)$, and cannot be used for decision making. Off-policy control algorithms such

as Q-Learning [30] use a policy that is independent of the learned $Q(s, a)$ function. These methods are not guaranteed to converge if used with function approximation and bootstrapping. Bootstrapping happens when the algorithm updates its estimates of the current state-action value based on the successor state-action value estimations [25].

Sarsa(λ) uses a temporal-difference updating method (see Equation 3.2) 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. These parameters can be tuned to adjust the responsiveness of learning. $e_t(s, a)$ denotes the eligibility trace for that state and action at time t. Eligibility traces are used to propagate the error to previous state-action pairs at time less than t.

$$Q_{t+1}(s_t, a_t) \leftarrow Q_t(s_t, a_t) + \alpha\Big[r_{t+1} + \gamma Q_t(s_{t+1}, a_{t+1}) - Q_t(s_t, a_t)\Big]e_t(s_t, a_t) \quad (3.2)$$

3.2.1 Double Reward System

In the conventional Sarsa(λ) algorithm, updates to the approximation of state-action values, $Q(s, a)$, are done once for each learning step. For the companion-learning problem there are two potential sources of reward in each step.

The first reward is the immediate reward, denoted r_i, which the NPC observes from the environment immediately after taking an action. It reflects the instant consequence of the NPC's actions. This consequence might be gaining experience, acquiring an item, taking damage, provoking hostility, etc. For Example, if the NPC encounters a *Critical Failure* while disarming a trap, the NPC takes damage, but if the trap is disarmed, experience points (XP) are gained. Each consequence must be carefully considered to build an effective reward function which will then determine the NPC's preferences for choosing and performing actions in future.

The second reward is a delayed reward based on feedback from the PC. This feedback could be verbal or physical, such as receiving a gift from the PC. These rewards will inform the NPC of the PC's preferences, desires, and supportiveness. By giving positive or negative verbal rewards after the NPC takes an action in a situation, the PC will inform the NPC for similar situations in the future. One example of a physical reward is a healing potion. The PC may give this reward to the NPC after the NPC performs a life-jeopardizing action that is beneficial to the PC.

This delayed reward, denoted r_d, may or may not materialize since the PC may not provide a verbal reward or gift. In this dissertation, the delayed reward is often referred to as a verbal reward, in general, r_d could represent a verbal reward, a gift, or both. 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_d. This technique is different than performing two complete Sarsa(λ) steps. A step starts when the NPC performs an action and it ends when the NPC wants to perform another action. The delayed reward might or might not be observed during this time. If a delayed reward occurs, the rewards are accumulated and the algorithm waits for the time the next action selection is triggered and an action is selected in the new state to perform an update. If no delayed reward occurs before the next action selection is triggered, an update occurs without a delayed reward, so this approach is actually a Sarsa/Sarrsa algorithm. Note that if the PC provides more than one delayed reward after the NPC performed an action, the verbal rewards other than the first one are ignored.

The verbal reward function should take into account how much the NPC currently cares about the PC. In order to take the approval rating into account, the verbal reward is discounted by a factor of the NPC's approval of the PC. This approval factor should change based on the verbal or physical reward that the NPC receives, as well as other events in the game such as the outcome of the PC orders. As an illustration, consider a game situation where the NPC detects a trap and informs the PC about the trap. The PC says "Good job" or "Thanks for letting me know", which means that the PC appreciates the NPC's action. In this case the NPC's approval of the PC should increase. The amount of increment can be different based on various aspects of the situation such as the current value of the NPC's approval of the PC, the importance and risk of taking that action, etc. Generally speaking, there are two interactive situations in the game, directly related to the learning domains, that should cause the NPC's approval of the PC to change. The first situation is when the NPC receives a delayed reward, and the second situation is when the NPC analyzes an order from the PC and performs an action based on that order.

The former case is straightforward in the sense that whenever the PC gives a positive delayed reward, the NPC's approval of the PC should increase and whenever the PC gives a negative delayed reward, the NPC's approval of the PC should decrease. As one might guess, the change is based on whether the NPC and PC share the same values and care about the same things.

The latter case is based on action motivations, which are of two types: internal, and external. Actions with internal motivations are the ones that the NPC individually decides to perform. The decision making process for these actions can be triggered by various situations in the game such as detecting a trap, entering a new area, encountering a person whose pocket it easy to pick, and lots of other situations. The common factor among these

actions is that the NPC individually decides to perform an action based on a game situation. On the other hand, actions with external motivations are caused by direct orders from the PC. For example, the NPC can perform the disarm action immediately after detecting a trap. The motivation behind this action is an internal one since the NPC took this action because of the situation of encountering and/or detecting a trap. Lets say that the NPC tried and failed to disarm the trap. Now that the PC knows there is a trap in that place, the PC orders the NPC to try to disarm the trap again. This time the motivation for the disarm action is external. However, there is a chance that the NPC will refuse to disarm the trap and say "Do it yourself, I do not want to get hurt again". In this case, the motivation behind the refuse action was also the external PC order to disarm the trap. The NPC's approval of the PC is directly affected by the alignment of the NPC's motivations and the PC's orders.

There is a general rule for changing the approval when the action taken has an external motivation. Generally, if an action with external motivation was successful the NPC's approval should increase, otherwise it should decrease. When the NPC fails to perform a PC-ordered action, it shows the NPC that the PC does not have a clear idea about the NPC's abilities, the PC does not understand the game situation, or the PC does not care about the NPC. However, for the Refuse action there is an exception to the success rule, since the Refuse action always succeeds. A refuse action is triggered by the PC ordering the NPC to perform an action when the NPC thinks that action is harmful. In this case the NPC refuses. According to our general rule the NPC's approval of the PC should go up, as the refusal succeeds. However, since the NPC predicts failure, the NPC's approval should actually go down.

3.2.2 GESM Action Selection Policy

In reinforcement learning control algorithms $Q(s, a)$ estimates state-action values for each pair consisting of a state and one of the available actions in that state. The learned $Q(s, a)$ values form the foundation of our behavior policy π. In order to use the learned $Q(s, a)$ values, the algorithm needs an action selection policy. Action selection policies provide the agent with a consistent way of taking $Q(s, a)$ into account when choosing the next action to perform.

There are several action selection policies widely used in reinforcement learning. The simplest one is called the *greedy* policy in which the selected action is always the one with the highest $Q(s, a)$ among the available actions in that state. Although this action selection policy might seem reasonable at first, it has some deficiencies. The first issue with this

action selection policy is that it always exploits the agent's current knowledge to maximize the expected reward. It never explores the other actions with inferior $Q(s, a)$ to see if they might actually be better [25]. This policy would be good after the algorithm has learned the optimal policy, π^*, and the environment is static. However, it is not good for the beginning of a learning task or in a dynamically changing environment. There is a way to encourage the *greedy* policy to explore at the beginning of a learning task. In this method, which is called Optimistic Initial Values, the initial values of $Q(s, a)$s are initialized with higher than expected values. This will cause all the actions to be selected at least once by the *greedy* action selection policy. Although this might solve the exploration problem in the beginning, it does not tackle the dynamic environment problem.

An alternative to the *greedy* policy, that compensates for its pure exploitation, is the ϵ-*greedy* policy. In this policy ϵ represents the exploration probability. While using the ϵ-*greedy* policy, our agent chooses the next action randomly with probability of ϵ, and it chooses an action based on the *greedy* policy with probability of $1 - \epsilon$. The ϵ-*greedy* policy does a good job with balancing the exploration and exploitation. Nevertheless, for some problems its way of choosing among actions while exploring can be inefficient or lead to failure. The result of choosing a totally random action among available actions while exploring can be the worst possible action and in some cases it can lead to failures. For example, consider that an agent is using reinforcement learning to learn how to control a helicopter in the air. Selecting a series of random actions in this case can lead to the crash of the helicopter. Sometimes this problem can be solved by reducing ϵ over time. This will result in the *greedy* action selection policy when the optimal policy has been learned. However, in a dynamically changing environment, reducing ϵ is inappropriate. The *softmax* [2] action selection policy was introduced to solve this problem [25].

The *Softmax* policy transforms all the $Q(s, a)$ for all the available actions into probabilities. As one might expect, actions with higher $Q(s, a)$ values will have higher probabilities of being selected. The most famous method for *softmax* uses the Gibbs, or Boltzmann distribution [25] (see 3.3) to compute action selection probabilities. In this equation "n" is the number of available actions.

$$\frac{e^{\frac{Q_t(a)}{\tau}}}{\sum_{i=1}^{n} e^{\frac{Q_t(b_i)}{\tau}}} \tag{3.3}$$

Using *softmax* with this distribution has another effect. Using a τ close to zero will

magnify the differences between the $Q(s, a)$ values and ultimately between the probabilities of actions. Higher values for τ will minimize the differences and make the probabilities of all actions approximately equal.

Each of these policies has their advantages and disadvantages. For the companion learning problem, empirical results showed that neither the *ϵ-greedy* nor the *softmax* policy was effective. Therefore, I combined them into a new policy called **Greedy Epsilon Softmax** or **GESM**. This policy selects the action with highest $Q(s, a)$ value with probability of $1 - \epsilon$ (like *ϵ-greedy*), and uses *Softmax* with probability of ϵ. Using this action selection policy, I eliminate the main problem of the *ϵ-greedy* policy, which was disregarding the learned $Q(s, a)$ values while exploring. Since *softmax* transforms action-values to probabilities, where actions with higher $Q(s, a)$ values have higher probabilities, the worst actions have less chance of being selected.

In *softmax*, when τ is small, the probability of selecting any non-best action may be very low or even close to zero. This can lead *softmax* to act as *greedy* policy which never explores. To solve this problem, I also modified *softmax* to exclude the best action from its available actions. This forces *softmax* to purely explore with probability ϵ. Moreover, since *softmax* turns state-action values into probabilities, the ranking of the non-best actions become significant. The higher a state-action value, the higher its chance of being selected. As will be shown later in this dissertation GESM succeeds where *ϵ-greedy* and *softmax* individually fail.

3.3 Implementation In Neverwinter Nights

I used ScriptEase [14] with BioWare Corp's Neverwinter Nights [17] to initially implement this learning system in a real game environment. The companion agent can make a decision regarding many situation in the game. Recall that in today's games, these decisions are mostly based on the orders of the PC or a game event. For example, in a particular game situation, the PC might tell the NPC to go into or come out of stealth mode or to disarm a trap. In most games, including Neverwinter Nights, the NPC will always obey the PC.

I assert that learning can be used to enable the companion to act in a natural way. This learning will help the NPCs to develop preferences among available actions in different situations in the game. To accomplish this task it is necessary to devise a decision making architecture that is consistent and compatible with both the learning plan and the game mechanics. I will first discuss the game-independent decision domain architecture and then

Decision Making Process				
Decision Event ⟶	Decision ⟶	Action ⟶	Feedback ⟶	Update
Reinforcement Learning Equivalent				
Encountering New State ⟶	Evaluate Actions ⟶	Perform Action ⟶	Reward ⟶	Update

Table 3.2: Decision making process and reinforcement learning equivalent of the process.

Decision Domain	*Traps*	*Pickpocket*	*Stealth*
Available Decision Events	Trap detection	Pickpocket candidate detected	Area transition
	PC orders NPC to disarm trap	PC orders NPC to pick a person's pocket	Detects enemy
	PC orders NPC to mark trap	PC orders NPC to identify the pickpocket candidate	PC orders NPC to enter stealth mode
			PC orders NPC exit stealth mode
			Timed

Table 3.3: Different decision event sets for different decision domains.

the different decision domains I built for Neverwinter Nights.

3.3.1 Decision Domain Architecture

Every decision domain is built from the same general blueprint[1]. The general architecture of a decision domain contains the decision making process, the set of actions available in that domain, the set decision events, and the action subsets for each decision event. Table 3.2 shows the decision making process in the game and how reinforcement learning will be used to mirror and materialize that process.

The first step in the decision making process is the *Decision Event* which is equivalent to encountering a new state in a reinforcement learning problem. The *Decision Event* is an event which tells the companion that a decision should be made about the current game situation. It will also inform the NPC about the decision domain for which a decision should be made and the available subset of actions for that *Decision Event*. Each decision domain is activated by one or more *Decision Events*. For example, "Area Transition" and "Enemy Detection" are two of the *Decision Events* for the *Stealth* decision domain. That is, whenever the NPC enters a new area or detects an enemy, that NPC must decide whether to enter stealth mode or not. Table 3.3 shows the *Decision Events* for the *Traps*, *Pickpocket*, and *Stealth* decision domains.

[1]Blueprint here means framework, skeleton, map, or pattern. It does not refer to a Neverwinter Nights blueprint.

Decision Domain	Traps		
Decision Event	Trap detection	PC orders NPC to disarm trap	PC orders NPC to mark trap
Motivation Type	Internal	External	External
Possible NPC Actions	[T] Disarm	[T] Disarm	
	[T] Mark		[T] Mark
	[T] Inform PC		
	[T] Nothing		
		[T] Refuse	[T] Refuse

Table 3.4: Different action sets available for the three different *Decision Events* in the *Traps* decision domain.

The second step in the decision making process is the *Decision*. In this step, the companion will use the GESM action selection policy to choose among the set of actions available for that specific *Decision Event*. Remember that the $Q(s, a)$ values for similar actions in a decision domain are shared. Table 3.4 shows the different subsets of actions available after each *Decision Event* in the *Traps* decision domain. For example, Table 3.4 shows that after the *"PC orders NPC to disarm trap"* event is triggered, NPC must choose either "[T] Disarm" or "[T] Refuse". The other actions in the action set of the *Traps Decision Domain* are excluded from this subset.

The third step in the decision making process is *Action*. The third step starts as soon as the second step is finished. In this step, the NPC performs the chosen action. The action depends on the decision domain, the *Decision Event* that activated it, and the $Q(s, a)$ values. For example, this action can be attempting to disarm a trap (*Traps-Trap detection*), informing the PC of a good pickpocket candidate (*Pickpocket-Pickpocket candidate detected*) or exiting stealth mode (*Stealth-Area transition*).

In the fourth step which is the *Feedback* step, the NPC observes the consequences of the performed action's interactions with the environment. The environment contains everything that interacts with the NPC in the game. The feedback system supports double rewards (see Section 3.2.1) that can extend over the time period between actions. Table 3.2 shows that the word reward in the learning context corresponds to the word feedback in the decision process. The first reward is the immediate reward, which can be in the form of obtaining valuable items in the game, taking damage, gaining experience, etc. The second reward is the feedback from the PC. This feedback should be delayed until the *Decision Event* for the next action in that decision domain is activated. At this point the learning algorithm proceeds to the next step in the learning process, even if the NPC has not observed any

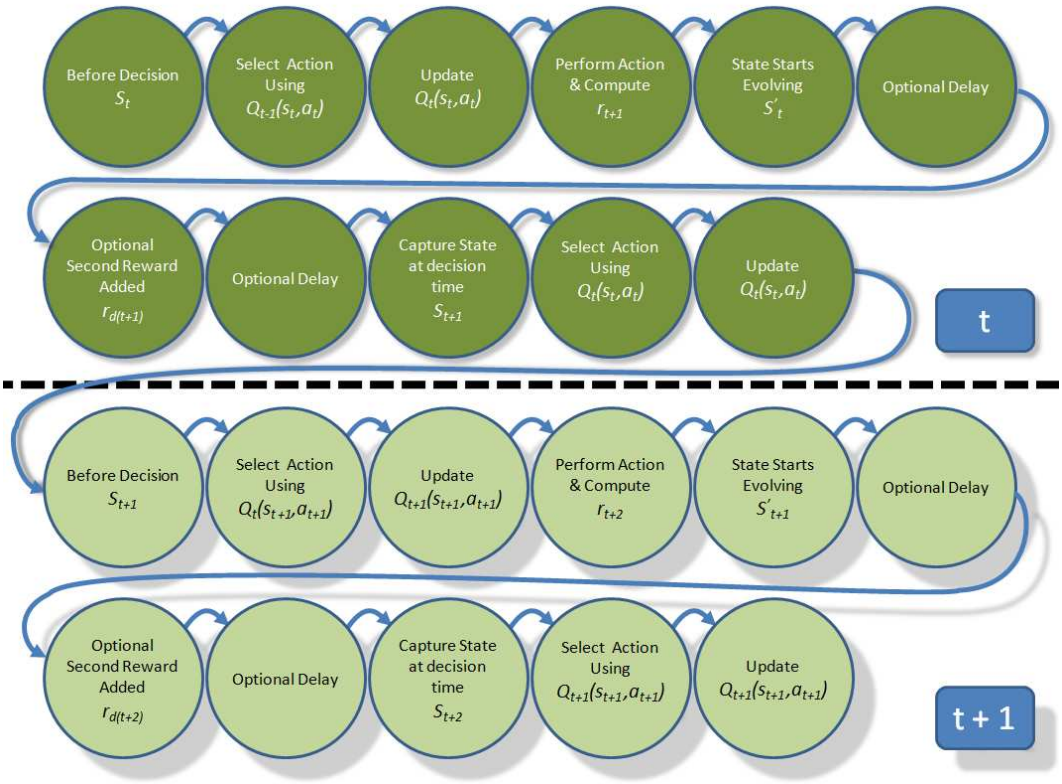

Figure 3.2: Learning Update Proces

feedback from the PC. Figure 3.2 illustrates this process. The state may change frequently between the time immediately after an action is performed (s'_t) and when the *Decision Event* to select the next action occurs. The state that is used to select the next action is the current state when the next action is selected (s_{t+1}).

The final step in the learning process is the *Update* step. In this step, the immediate and delayed rewards (if given) are accumulated and the $Q(s, a)$ value for the performed action in the previous step is updated using the Sarsa(λ) update method (see Equation 3.2).

3.3.2 Implementation of Decision Domains

Each task or group of related tasks in a game that requires the NPC to select an action from a set of actions can be considered as a decision domain. In this dissertation the focus is on decision domains related to PC-companion interactions. In Neverwinter Nights, there can be many such decision domains: healing, dealing with traps, lock picking, choosing a following distance, etc. Learning may be more important in some of these domains than others. Tasks suitable for learning are ones in which the NPC receives orders from the PC, and where the NPC must weigh personal reward and consequences against rewards and

consequences for the PC. In these cases learning is necessary since the PC's behavior may be dynamic, forcing the NPC to adapt.

The rewards and consequences of each decision domain can be different. There are some decision domains in which the NPC gains XP or takes damage. In these domains the goal is to achieve a balance between the gained XP and the damage taken. However, there are some decision domains, where the goal does not fit this model. For example, the actions in the *Stealth* decision domain, "`[S]` Be Stealthy" and "`[S]` Be Normal", do not directly gain XP or damage the NPC. As a result, other goals may be necessary in these decision domains. One general example is the goal of gaining the PC's approval. The NPC can learn a model ($Q(s, a)$) that maximizes the PC's approval in this decision domain. For example, if the PC believes that the NPC should not be in stealth mode at night and be in stealth mode during the day, the NPC should learn that the value ($Q(s, a)$) for the "day" state in and the "`[S]` Be Stealthy" action should increase while the value for the "night" state and the same action should decrease. In order for the NPC to learn this model, the PC needs to express the required behavior either in the form of giving orders, or giving positive or negative feedback after the NPC performs an action. In this example, there is not immediate reward, only a verbal reward. The NPC is simply trying to learn the PC's preference for this decision domain rather than balance rewards and consequences.

Each decision domain uses a partial game state constructed from pertinent current game conditions. These conditions should be chosen carefully by considering their relevance to the decision domain. These conditions form the feature vector, denoted ϕ. In this vector, the value of each element corresponds to the whether or not a condition is present in the game. Using binary features means that the value of each of the elements in the ϕ vector can be either 0 or 1. In linear function approximation, there is also a weight vector, denoted w, with the same number of elements. The state-action value function, $Q(s, a)$ is calculated by computing the dot product of these two vectors (see 3.4). Linear function approximation was used for all decision domains in this research.

$$Q(s, a) = \phi_s \cdot w_a^T \tag{3.4}$$

Stealth Decision Domain

In some story-based games the characters in the game usually have the ability to be in a special mode called the "Stealth Mode". The effect of being in this mode varies slightly among different games but generally means to hide and move silently. Movement in stealth

Figure 3.3: An interaction between the PC and NPC in the *Stealth* decision domain. The PC disagrees with what the NPC has done. The companion is transparent because of being in stealth mode.

mode is slower than normal. A character is harder to spot if the stealth mode is active. In stealth mode, a character can only be perceived if the searching creature makes a successful *spot check* (perception check). In Neverwinter Nights, the *spot check* test is modified by various game play aspects. For example, using a torch in stealth mode will increase the chance of being detected. Another factor is the distance between the searching and hiding creatures. Whether the hiding creature is indoors or outdoors, can also affect the *spot check*. Normally, the only way for an NPC to be in stealth mode is to be directly ordered to be in this mode. In NWN, this is accomplished by selecting the stealth option from the *Special Abilities* heading in the radial menu. However, the behavior that we expect from a person in real life would be different. Our goal is to enable the NPC to learn the PC's stealth preferences. After learning, the NPC can enter stealth mode autonomously by anticipating the PC's preferences.

The first decision domain that I modeled in the NWN environment was the *Stealth* decision domain. My goal in this decision domain was to enable the NPC to build a model of the PC's preferences. There was no intent to allow the NPC to use *approval* to disregard

Game Condition	Corresponding Elements in ϕ Vector
Time of Day	Is Day
	Is Night
	Is Dawn
	Is Dusk
Area Type	Is Indoor
	Is Outdoor
	Is Above Ground
	Is Underground
	Is Natural
	Is Artificial
Weather Conditions	Is Foggy
	Is Clear
Hostile Creatures	Is a Hostile Creature Around
	Is Everyone Around Friendly
Actual Following Distance	Is The Following Distance Normal
	Is The Following Distance Far
PC's Stealth Mode	Is PC in Stealth Mode
	Is PC in Normal Mode
Following Distance (As Set by The PC)	Is The Following Distance Set to "Close"
	Is The Following Distance Set to "Medium"
	Is The Following Distance Set to "Far"
PC's Alignment	Is The PC Good
	Is The PC Evil

Table 3.5: *Stealth* decision domain feature vector and the elements corresponding to each game condition.

the PC's preferences. There is no immediate damage taken or XP gained in this domain. It is possible that some eventual damage or XP may result from this decision, but this issue was not considered in this decision domain. As a result, in this decision domain, the only means of learning whether to be stealthy or not is to interact with the PC.

Like other decision domains, this one contains a set of *Decision Events* which can be seen in Table 3.3. When one of these events occurs, the NPC selects an action among the available actions. There are only two actions in this decision domain: the first one is "[S] Be Stealthy" and the second one is "[S] Be Normal" (as opposed to stealth). Both actions are available after all four *Decision Events*.

The feedback of the PC can be positive, negative, or silence. In this simple model, negative feedback from the PC not only provides a reward, r, but also forces the NPC to do what the PC prefers, since the NPC is only trying to learn the PC's preferences. Currently the NPC does not use the *approval* to discount the PC's feedback, although this could be

changed. Figure 3.3 shows that after entering a new area, the NPC decides not to be in the stealth mode. Then the PC gives negative feedback by saying "bad idea". Consequently, the NPC updates the state action value function, $Q(s, a)$, and obeys the PC by entering stealth mode.

The feature vector, ϕ, for the *Stealth* decision domain contains 22 elements which correspond to 8 different conditions in the game. The feature vector and conditions for the *Stealth* decision domain are shown in Table 3.5. Since this decision domain is based on modeling the PC's preferences, the features in the feature vector are selected to reflect the PC's considerations about the NPC being in stealth mode.

For this decision domain I used both Sarsa(λ) and ALeRT [6] learning algorithms. The reinforcement learning problem for this domain is an episodic problem with a non-episodic nature as discussed in Section 3.2. As a result we deal only with one episode. Since there is no XP gained or damage taken after performing the actions in this domain, in addition to the goal of this decision domain being to build a model of the PC's preferences, the reward function for the actions in this domain should be based directly on the feedback of the PC. The reward system is fairly simple in this decision domain in contrast with the reward system of the decision domains that need to balance between XP, damage, and delayed rewards. In this domain the NPC receives $+1$ when the PC agrees with the chosen action and receives -1 when the PC disagrees. The results for this decision domain and the comparison between the two algorithms can be seen in Section 4.1.

Traps Decision Domain

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 the possible damage. Such NPCs do not look intelligent in the player's eyes. 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. For example, it is possible that the NPC remains silent (performs action "`[T]` Nothing") after detecting a trap. The reason might be that the PC was trying to force the NPC to disarm every trap regardless of its difficulty whenever the companion informed the player about a trap. As a result, the NPC decides to remain silent and avoid getting hurt. On the other hand, if the NPC realizes all the traps in the way are easy traps and by disarming them a large amount of

XP will be gained, disarming traps (performing action "[T] Disarm") after detecting them might be a better idea.

The goal of this research is to make the NPC understand the consequences of performing different actions on different traps. By doing this, the NPC will develop preferences about performing different actions regarding traps. These preferences will change as the companion's skill and the difficulty of the traps change. The way the player plays the game will also affect the NPC's preferences. For example, if the PC only encourages the NPC to disarm easy or medium traps, then the occasional damage may be worth the XP gained from successfully disarming most of the traps. On the contrary, if the companion realizes that the PC does not care at all, the NPC will be more careful. Moreover, the NPC might decide to mark the traps instead of disarming them based on the orders of the PC, trap difficulties, and the approval factor. Marking traps is very beneficial for the PC and less harmful than disarm, but it does not gain any XP for the NPC. The learning system should provide the NPC with the ability to balance between taking damage from some traps, gaining XP from successfully disarmed traps, and securing verbal rewards from the PC.

The reinforcement learning problem for the *Traps* decision domain is episodic with a non-episodic nature, 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. Like all decision domains, this decision domain also has a set of *Decision Events*. The set of these events in addition to the sets of actions available to the companion NPC after each of these *Decision Events* are shown in Table 3.4.

There are five different actions in the *Traps* decision domain. However, not all of them are available after each *Decision Event*. These actions are "[T] Nothing", "[T] Disarm", "[T] Mark", "[T] Inform PC", and "[T] Refuse". Each action has a motivation that can be internal or external which is discussed in detail in Section 3.2.1. Actions with external motivations, which are the results of interacting with the PC, will change the NPC's *approval* of the PC, while the actions with internal motivations do not. Another important property of actions is their outcome.

Each action has three different possible outcomes: *Success*, *Fail*, and *Critical Fail*. *Success* means that the action was performed successfully, there is no damage taken by the NPC, and if it is possible for the NPC to gain XP by performing that action the XP is gained. If an action has an external motivation with the successful outcome, it often increases the NPC's *approval* of the PC. There is one exception to this rule which is "[T] Refuse" which

Trap Type	Action / Outcome	[T] Disarm	[T] Mark	[T] Inform PC	[T] Nothing	[T] Refuse
Easy	*Success*	80%	100%	100%	100%	100%
5-10%	*Fail*	10%	0%	0%	0%	0%
Damage	*Critical Fail*	10%	0%	0%	0%	0%
Medium	*Success*	50%	70%	100%	100%	100%
10-20%	*Fail*	10%	10%	0%	0%	0%
Damage	*Critical Fail*	40%	20%	0%	0%	0%
Hard	*Success*	10%	50%	100%	100%	100%
20-30%	*Fail*	10%	10%	0%	0%	0%
Damage	*Critical Fail*	80%	40%	0%	0%	0%

Table 3.6: The probabilities of the outcome of different actions and the amount of damage the NPC takes after the *Critical Fail* action outcome in the *Traps* decision domain.

will decrease the *approval* considering that its motivation is always external and it is always successful. The *Fail* outcome of an action means that there is no damage taken by performing that action. However, that action was not successful so no XP was gained. For example, lets consider action "[T] Mark". If the outcome of this action is *Fail*, it means that the trap is not marked and there is no damage taken. The last action outcome is the *Critical Fail* which is the worst among outcomes. When an action critically fails there is no XP gained, and the NPC takes damage if it is possible to take damage from performing that action. After *Critical Fail* some traps remain active and some become disarmed. In this research none of the traps disappear or become disarmed by being triggered. For example, if the NPC performs "[T] Disarm" and critically fails, the trap remains intact while the NPC takes damage and does not gain any XP. The set of actions in the *Traps* domain for this research is larger than available in the Neverwinter Nights game since the "[T] Inform" and "[T] Nothing" actions have been added for better realism.

For this research the traps are divided into three categories: *Easy*, *Medium*, and *Hard*. The amount of damage taken from triggering a trap and the chances of triggering that trap depend on the trap category. Each trap category is an abstraction that considers both the relative skill of the NPC and the base difficulty of the trap. An *Easy* trap for a high-level NPC may actually have a base difficulty that is harder than a *Hard* category trap for a low-level NPC. Table 3.6 shows the properties of actions, the amount of damage due to *Critical Fail*, and their *Success*, *Fail*, and *Critical Fail* probabilities relative to the trap difficulty. We have used these probabilities to model the in-game NPC abilities. They are based on NWN traps where there is always a 10% chance of simple failure, and the chances for success and critical failure have variable ranges on both sides of this 10%, based on the difficulty of the trap. The NPC is a rogue since rogues are capable of dealing with traps better than other NPC classes in NWN.

Action "[T] Nothing" and action "[T] Refuse" serve different purposes and have different effects on the learning variables such as *approval*. Action "[T] Nothing" is available after the NPC detects a trap. This action simply means doing nothing. The result of this action might be that the PC would not realize that there is a trap in the way and might accidentally trigger the trap. The PC might also walk away from the trap without knowing about it. On the other hand, action "[T] Refuse" is different. Although it also means doing nothing, it serves a different purpose at a different time. The NPC can select this action after being ordered by the PC to mark or disarm the trap. There can be several reasons that a companion might select "[T] Refuse". For example, the PC's previous selfish behavior may make it clear that the PC does not care about the NPC. Alternatively the traps might just be too hard. There is another difference between these two actions. The motivation behind performing action "[T] Nothing" is internal. That is, the NPC decides to perform this action based on detecting a trap. Performing this action does not affect what the NPC thinks of the PC. However, the motivation for performing action "[T] Refuse" is external which means that the NPC performs it because of disagreeing with the PC's suggestion. As a result, action "[T] Refuse" will decrease the *approval* that the NPC has for the PC.

Action "[T] Inform PC" shows the PC an approximation of the boundaries of the trap for a short period of time. This action mimics the real world's pointing action. It is not possible to take damage or gain XP by performing this action. It is also impossible to have *Fail* or *Critical Fail* outcomes for this action. The motivation for this action is always internal and the PC never tells the NPC to point to the trap again. This action and its supporting game mechanics have been added to Neverwinter Nights to add realism during this research. In the game, a circle surrounding the trap area appears for a short while and then disappears as the effect of the "[T] Inform PC" action. This effect can be seen in Figure 3.4. As we can see this action will reduce the chances of triggering the trap accidentally by the PC. However, since the circle shows only an approximation of the position of the trap and will be gone after a short period of time, the reduction in the chance of accidentally triggering the trap is not as much as marking the trap.

There are several reasons for performing this action. One reason might be that the PC does not want the NPC to perform any marking or disarming. Another reason might be that the NPC observes lots of hard traps, which are dangerous either for the companion or the PC to disarm. In this case, the NPC who may have a high *approval* of the PC may want to warn the PC about the possible danger without taking damage. Alternatively, if the NPC has a low approval of the PC, the NPC may decide not to inform the PC to avoid a disarm

Figure 3.4: The NPC is performing "[T] Inform PC" action after detecting a trap in the *Traps* decision domain. The circle shows an approximation of the boundaries of the trap for a while and then it disappears.

or mark order, regardless of the trap difficulty. In this case the NPC knows that refusing an order can damage the relationship with the PC. As a result, not informing the PC may be a better strategy when the NPC expects an unreasonable behavior from the PC.

The "[T] Mark" action can be selected based on different *Decision Events*. It means that this action can have both internal or external motivations. Performing this action basically means that the NPC will try to mark a trap in a way that it can be visually seen by the PC permanently. The PC will know the exact borders of the trap and it will help the PC and other members of the party to avoid the trap. Action "[T] Mark" can have all three possible outcomes with different probabilities based on NPC skills and the difficulty of the traps (Abstracted by a trap category). *Success* in "[T] Mark" means that the trap and its exact boundaries are visible to the PC. However, the trap is still active and if the PC or NPC steps on the trap, they will take damage. Moreover, performing this action with the outcome of *Success* will not gain any XP for the NPC or PC. There can be several reasons for the NPC to perform this action instead of disarming trap to gain XP. The most common reason for marking a trap rather than only informing the PC or doing nothing is that it will

Figure 3.5: The NPC has marked a trap after hearing the PC's order to mark it in the *Traps* decision domain. The red polygon shows the marked trap.

reduce the chance of the PC being harmed by the trap. Secondly, if the NPC has a very high approval of the PC it is natural to do something for the PC and it will improve the NPC-PC relationship. Figure 3.5 shows a trap which is marked after the PC's order to mark the trap.

The *Fail* outcome in "[T] Mark" means that the trap is not marked and still invisible. However, since the NPC tried to mark it, the player will know that there must be a trap near the position where the NPC tried to mark it. As a result, a failed marking attempt slightly reduces the chance of damage for the PC. There is no damage taken when a "[T] Mark" action fails. The third possible outcome of action "[T] Mark" is *Critical Fail*, where the trap remains unmarked, and the NPC takes damage. It is possible that certain types of traps damage everyone in a range so this can also affect the PC. This outcome can also slightly reduce the chance of the trap being triggered by the PC, since the *Critical Fail* event makes the traps general location somewhat obvious. The "[T] Mark" action with external motivation happens when the PC orders the NPC to mark a trap. If the outcome is either *Fail* or *Critical Fail*, the NPC's *approval* of the PC will go down. If the outcome is *Success*, the *approval* factor will go up. If the motivation for this action is internal the *approval* will remain intact regardless of the outcome of the action.

Figure 3.6: The NPC is trying to disarm a trap after receiving the PC's order in the *Traps* decision domain. A disarm attempt (or a mark attempt) will give a rough idea of the position of the trap to the PC when the outcome is *Fail* or *Critical Fail*.

The "[T] Disarm" action is very similar to the "[T] Mark" action in terms of motivations. This action can have both internal and external motivations. Performing this action means that the NPC attempts to disarm a trap. If a trap is successfully disarmed (*Success* action outcome), it will disappear and has no effect unless another game event reactivates the trap. The *Success* outcome of this action has the highest reward for both the PC and NPC. First, the trap is disarmed, which makes it impossible to accidentally trigger it later. Second, both the PC and NPC gain XP. One might argue that if the tarp is marked and the PC is careful enough, it is not possible to trigger the trap and it has the same effect as disarming the trap. However, there are some traps in the game that cover the only access to a story goal or reward. In this case, it is impossible to reach the goal or obtain the reward without disarming the trap or taking damage.

The *Fail* outcome of the "[T] Disarm" action occurs when the NPC fails to disarm the trap but does not fail by a margin large enough to cause critical failure. In this case, there is no damage taken and no XP gained for the NPC. After this action outcome the trap remains active. Since the NPC tried to disarm the trap which requires manipulating the trap, the PC

will get a rough idea of the position of the trap. Figure 3.6 shows an NPC trying to disarm a trap after receiving the disarm order from the PC. The PC cannot distinguish the difference between a failed "[T] Disarm" and a failed "[T] Mark" action. In the case of failure or critical failure the PC cannot discern the NPC's exact action. The *Critical Fail* outcome for the "[T] Disarm" action means that the trap is still intact, there is no XP gained, and the NPC took damage. Performing this action has the highest probability of *Fail* or *Critical Fail* outcomes as can be seen in Table 3.6. As a result, the NPC who performs this action should either have a very high *approval* of the PC, or based on past experience, the NPC should conclude that the potential XP gained by disarming the trap is worth the possible damage.

It is possible to use this learning system to train the NPC to exhibit behaviors that depend on the PC's behaviors. For example, consider when the PC has the rogue class. One of the important aspects of the game for a rogue PC is to find and disarm traps. If the NPC does these tasks for the PC all the time, the game may not be as fun for the player. As a result, the learning system should allow the PC to train the NPC to inform the PC about the existence of traps (performing action "[T] Inform PC") instead of disarming or marking the traps. In this case, even though the NPC might have a high *approval* of the PC, the companion would leave the traps for the PC. In fact, if the NPC has high *approval* of the PC, it should be easy for the PC to train the NPC to act in a particular way. However, if the NPC has a low *approval* of the PC, the companion actions will be mostly based on personal motivations to gain XP and to avoid taking damage. In this case, it is harder for a PC to train the companion in a particular way. The model implements this mechanism by using an *approval* factor that has a direct impact on the reward function.

In the *Stealth* decision domain, experiment showed that Sarsa(λ) was sufficient to generate good results (see Section 4.1). Therefore, the more complex ALeRT Algorithm [6] was abandoned for subsequent decision domains in favor of Sarsa(λ). Sarsa(λ) showed that its speed of adaptation to the environment produced natural looking behaviors. When trying to generate natural looking behaviors, the goal is not to find the optimal policy. Human behavior is not optimal so the goal not to converge to the optimal policy as fast as possible. For example, the NPC who has experienced 10 easy traps and thinks that disarming traps is a good policy, might suddenly encounter 5 hard traps. If the NPC instantly decides not to disarm the traps after encountering the first hard trap, it would be an unnatural behavior. As a matter of fact, the NPC should first become doubtful about disarming traps and if this trend of hard traps continues, then based on the *approval* of the PC, and other parameters

Parameter	Formula or Value
XPR	+0.2
TRR	A*(Average Trap Damage)*(ΔRevelation Factor RF)
	RF = 0 for [T] Nothing or [T] Refuse RF = 0.3 for [T] Inform PC RF = 0.8 for [T] Mark RF = 1.0 for [T] Disarm
IDR	-(1-A)*(Actual Critical Failure Damage)
AR	AF*A
AF	+0.35

Table 3.7: Required parameters for building the immediate reward function in the *Traps* decision domain

involved, decide what to do next. In this dissertation and for this domain, I have considered encountering 5 traps to be roughly the amount of time the NPC needs to learn a new behavior and adapt to the changes in the environment.

The reinforcement learning problem for this domain, like the *Stealth* decision domain, is also an episodic problem with a non-episodic nature as discussed in Section 3.2. As a result, the whole learning process can be considered as only one episode. There is both XP gained and damage taken in this domain. The goal of this decision domain is to balance the XP gained and the damage taken by the NPC as well as exhibiting a natural looking behavior based on the NPC's *approval* of the PC. The reward function for the actions in this domain should be based on XP, current damage, possible future damage, and verbal feedback of the PC.

In this dissertation there are two types of rewards. The immediate reward, denoted r_i, and the delayed reward, denoted r_d. The combination of the immediate and the delayed reward is parameterized based on the action and the action outcome. This reward should take into account the impact of taking an action. This impact can be divided into smaller sub-impacts such as taking damage, gaining XP points, reducing the chance of the PC taking damage in the future, etc. Table 3.7 shows the parameters used to create this combined reward function for the *Traps* decision domain. This reward has two positive components, one negative component and one component whose sign is variable (see Equation 3.5) .

$$Reward = XPR + TRR + IDR + AR \tag{3.5}$$

The Experience points reward, denoted XPR, is the reward that represents the XP gained by successfully disarming a trap. It is zero in situations other than a "[T] Disarm"

action with a *Success* outcome. XPR is constant and determined by the relative damage a character must usually take to accumulate XP. The value of this parameter actually balances how much XP an NPC with a medium *approval* must gain when the outcome is *Success*, to risk taking damage when there is a critical failure. This value has been determined by conducting a series of experiments. If the XPR is too high, the NPC will never care about taking damage and sacrifice anything in order to gain XP. If the XPR is too low, then the NPC becomes too conservative and never takes the risk of attempting to disarm a trap to gain XP because it would not be worth the damage.

The Trap revelation reward, denoted TRR, represents the reward for revealing the existence of a trap. It accounts for potential 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. The average trap damage for an *Easy* trap is 7.5%, for a *Medium* trap is 15%, and for a *Hard* trap is 25% of the maximum NPC hit points (HP).

If after detecting a trap, the NPC only informs the PC that a trap exists, the RF is only 0.3 (revelation factor of informing), since there is still a significant chance that the PC will get damaged by not knowing the exact location of the trap. If the NPC instead marks a non-revealed trap, the RF is 0.8 (revelation factor of marking) of this total. However, if the NPC first informs and then marks a trap, the revelation factor for informing already accounts for 0.3 of the total so the ΔRF for marking is $(0.8 - 0.3)$ which is 0.5. In general ΔRF is difference of RF for the latest action minus the maximum RF for past revelation actions. Figure 3.6 shows an NPC who is trying to disarm a trap. Regardless of the outcome of that action, the player will notice that there is a trap in that place. As a result, failing in "`[T]` Mark" and "`[T]` Disarm" actions will also reveal the trap. The revelation factor for "`[T]` Inform" with *Success* outcome, "`[T]` Mark" with *Fail* or *Critical Fail* outcomes, and "`[T]` Disarm" with *Fail* or *Critical Fail* outcomes is 0.3, since in all these cases the PC only knows the general location of the trap. The RF for a "`[T]` Mark" action with *Success* outcome is 0.8, since the PC knows the exact location for the trap. For the "`[T]` Disarm" with *Success* outcome, the RF is 1.0, since the trap is disarmed so it cannot do future damage to the PC.

This mechanism (TRR) will prevent the NPC from getting multiple rewards for a specific action and for actions that do not really help the PC. For example, when the NPC encounters a trap, the revelation factor is 0. If the NPC instantly marks the trap success-

Game Condition	Corresponding Elements in ϕ Vector
NPC's approval of the PC	Is the NPC's approval of the PC greater than 0.5.
NPC's Damage	Is the damage to the NPC from a *Critical Fail* greater than 10% of the NPC's maximum hitpoints.
NPC's Skill Rank	Is the NPC's skill rank of disarming traps greater than the NPC's level.
NPC's Dexterity	Is the NPC's dexterity skill modifier greater than 3.
Constant	A constant 1 for normalization.

Table 3.8: *Traps* decision domain feature vector and the elements corresponding to each game condition.

fully, the revelation factor becomes 0.8 and all of this reward ($\Delta RF = 0.8$) goes to the "[T] Mark" action. Now, if the NPC tries to inform the PC, or tries to mark the trap again, it does not help the PC and the revelation factor remains at 0.8, which means that the ΔRF will be zero. Later, if the NPC successfully disarms the trap, since the revelation factor of the disarm is 1.0 and the current revelation factor of the trap is 0.8, ΔRF will be 0.2 which accounts for the amount of damage that the successful "[T] Disarm" action prevented the PC from taking. ΔRF is nonnegative since the PC's knowledge of the trap location cannot decrease.

The Immediate damage reward, denoted IDR, is the negative reward that represents damage taken by the NPC for a *Critical Fail* action outcome, while disarming or marking a trap. IDR is discounted based on $1 - (\textit{NPC's approval of the PC})$. The reason is that the NPC who likes the PC might be willing to take more damage in favor of the PC. This model can be justified by considering that it is possible that the PC heals the NPC when the NPC takes damage. This potential healing process can increase the NPC's *approval* of the PC and result in this behavior. It can ultimately make the NPC understand that taking damage for a caring PC may be beneficial.

The Approval reward, denoted 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. A is used since the NPC's approval limits the amount that the verbal reward will affect the NPC's future decisions. For example, an *approval* of 1.0 does not limit the effect while an *approval* of 0.5 cuts the effect of the verbal reward in half. AR is represented by the delayed reward, r_d, and may or may not materialize since the PC may not provide a verbal reward. In many cases the PC does not give any verbal reward to the NPC, even if the PC is satisfied. This reward is called the *silent verbal reward*,

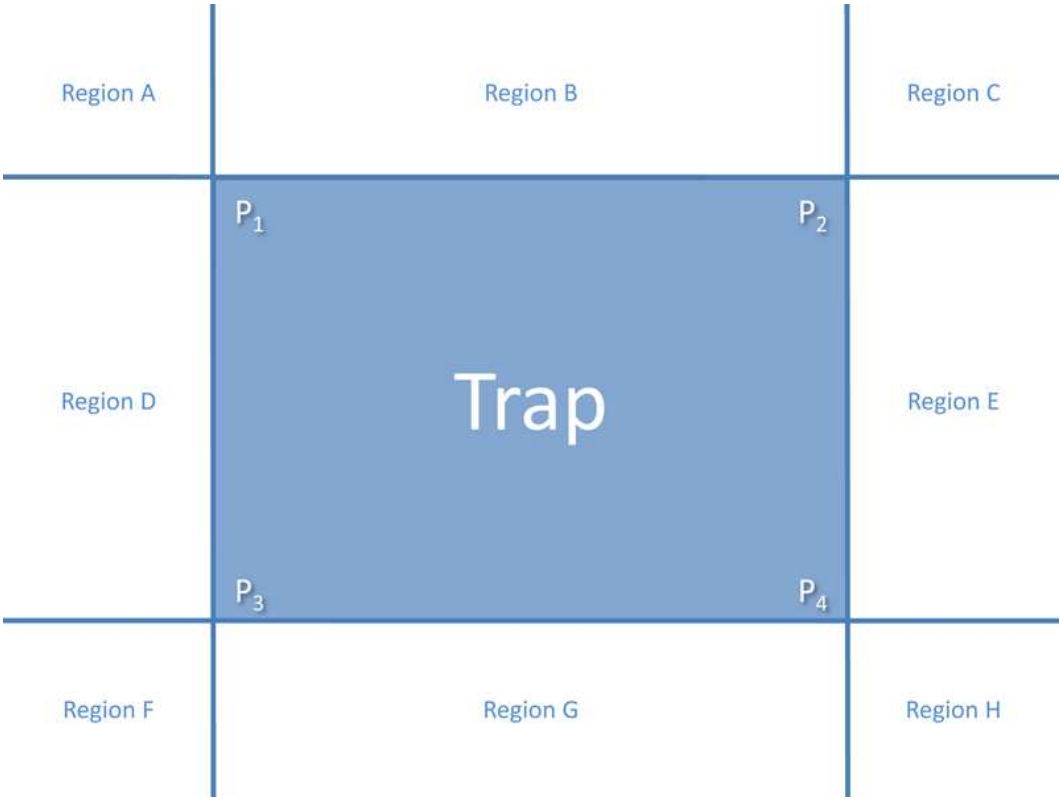

Figure 3.7: The regions that are used in computing a bypass route for the trap.

denoted r_{silent}. It is a small positive reward that is given to the NPC if a *Decision Event* is triggered and the PC did not give any verbal feedback. The amount of this reward is 10% of the normal positive AR.

The feature vector used for learning in the *Traps* decision domain contains 5 binary features that represent the state of the environment. This feature vector enables the NPC to distinguish between different game situations and reflects the aspects of the general game state that are important to this decision domain. The list of these features can be seen in Table 3.8. Since linear function approximation is used, the algorithm calculates $Q(s, a)$ as the dot product of the learned weight vector of an action, w_a^T, and the binary feature vector , ϕ_s, using the equation 3.4. The weights in the w_a^T weight vectors for all actions are initialized with 0. The results for this decision domain can be seen in Section 4.2.

The goal of this research is to generate adaptive and human-like behaviors, and it would look unnatural if the NPC detected a trap and walked over it. However in Neverwinter Nights, NPC's often detect a trap and walk over it even if the NPC did not talk to the PC about the trap. For this research, the NPC's companion behavior was implemented to avoid detected traps. This part of the implementation of this domain does not involve any learning

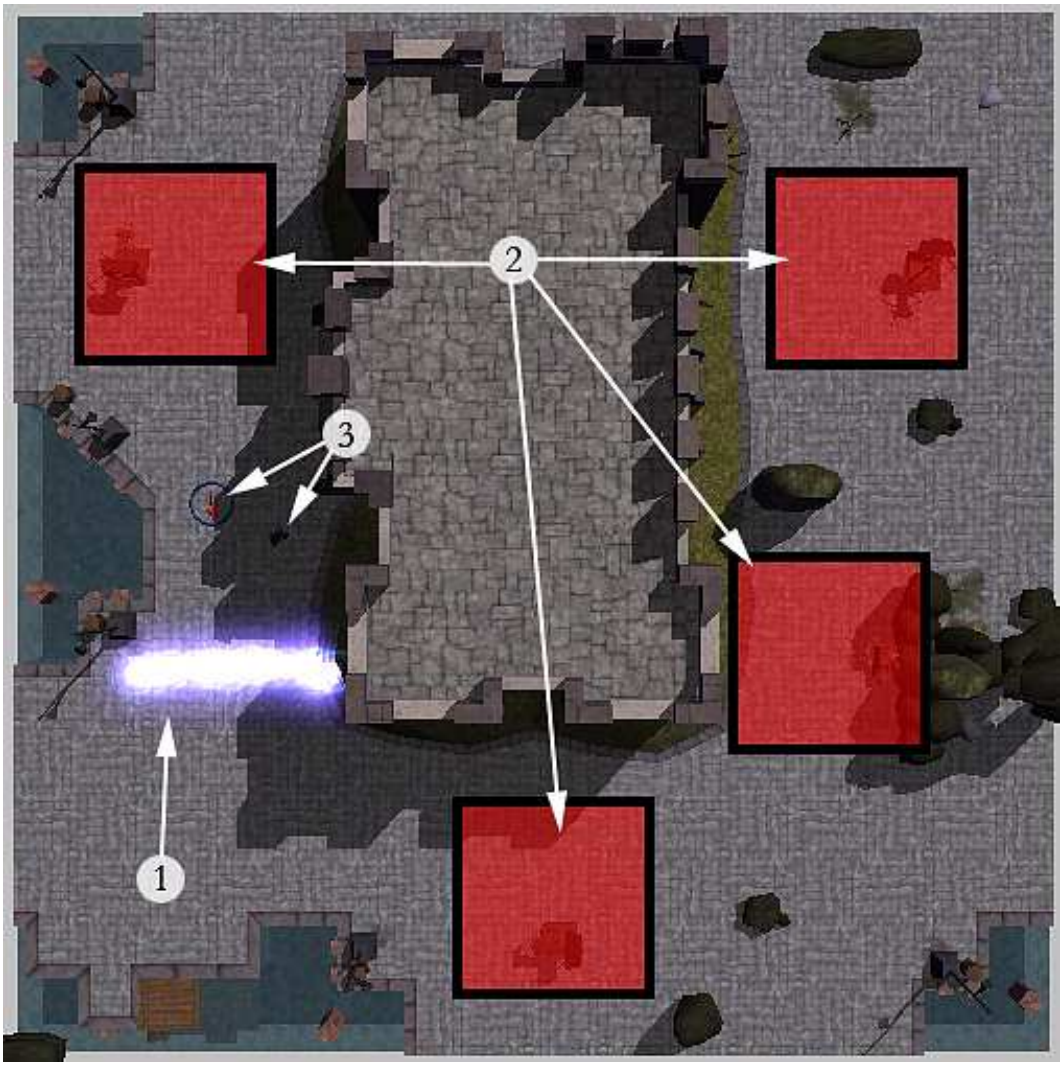

Figure 3.8: The *Traps* decision domain experiment area. Label 1 is the resetting trigger, Label 2 shows the traps. Label 3 denotes the starting point of the PC and NPC.

and is just an additional capability for the NPC's behavior to make it look more natural. The traps used to implement this decision domain are generally rectangular shaped traps. Each trap has 4 vertices and the location of each vertex is known. In order to enable the NPC to avoid traps, the area around a trap is divided into 8 separate regions as shown in Figure 3.7. There is an event triggered when the NPC gets too close to a trap. At this moment, the NPC decides how to bypass the trap by considering the region and the trap position. A simple path finding algorithm was implemented that avoids the trap. A rectangular danger zone is computed around the trap. If in moving towards the PC, the NPC enters the danger zone, the NPC moves to a point in a different region that is just outside the danger zone (see Figure 3.7). The NPC then tries to move towards the PC and if a danger zone is entered the

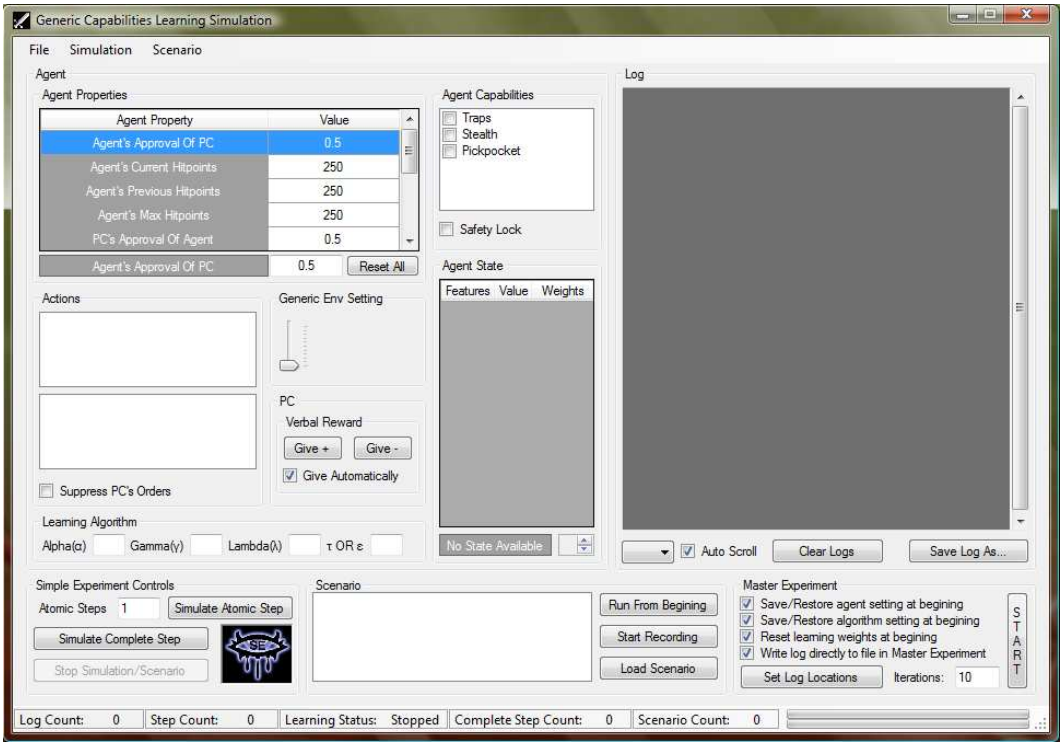

Figure 3.9: Simulator screen shot before selecting a decision domain

process is repeated. If there is more than one region to move to, the NPC remembers which region was selected, and if faced with the same decision again, picks a different region. This simple algorithm was sufficient to support the main goal of this research - to test the viability of learning in the *Traps* decision domain. More sophisticated obstacle avoidance algorithms exist.

The *Traps* decision domain was tested in Neverwinter Nights using the area shown in Figure 3.8. The PC and the companion NPC go counter clockwise around the castle starting from the point designated by Label 3. As they walk over the resetting trigger, designated by Label 1, for the first time, the traps are reset and the experiment starts. When the trigger is subsequently fired, the traps are reset but the experiment is not restarted. The difficulty of each trap can be set separately, which enables the experiment to have different traps like the real game.

3.4 The Simulator

In order to evaluate the learning system we need to evaluate the result of 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, I built a simulator. All the game parameters are available in the simulator and they work with the game mechanics. The simulator also enabled us to model common PC-companion interaction behaviors by setting parameters. The first step in the simulation process is to designate the initial values for the start of the simulation, such as the NPC's approval of the PC and the model of the PC (see Section 3.4.1). The NPC's approval of the PC can be fixed at a certain value in order to evaluate the effect of other parameters on the learned behaviors. Moreover, the experiments can contain as many traps as needed with different difficulties, and the parameters can be changed at any desired point during the simulation. The simulator uses Sarsa(λ) to generate the $Q(s, a)$ values for all actions at any point, as well as their averages and all the NPC-PC and NPC-environment interactions. The latter information lets us verify that the simulation worked using the exact game mechanics. The learned behaviors from the simulator can be plugged into the real game to observe them. The learning algorithm in Neverwinter Nights would then further change the "initial" learned behaviors obtained from the simulator. The *Traps* decision domain is the only decision domain that is currently implemented in the simulator. However, other decision domains can be easily added. The actions, states, and reward functions for each decision domain would need to be added. PC models would also need to be updated based on this new information and additional PC models can also be added. Figure 3.9 shows the general simulator interface that can be adapted for any decision domain. Figure 3.10 shows the simulator interface while running an experiment for the *Traps* decision domain. Notice that the "Generic Env Setting" slider control in Figure 3.9 has been adapted in the *Traps* domain to be a "Trap Difficulty" slider control (see Figure 3.10). In some decision domains, other controls may need to be added depending on appropriate state, but this is easy to do.

When a human player is playing the game, the urge to finish the game will force the player to make definitive decisions about traps. However, in the simulator it is possible that the NPC and the PC will continue making decisions about the same trap forever. Instead, there are certain situations where the NPC and the PC should move on to the next trap. The first and most obvious reason for moving on to the next trap is when the trap is successfully disarmed. The second reason for moving on is when the NPC detects a trap and performs action "`[T]` Nothing". In this case the PC does not know about the trap and ignores it. In general, a non-rogue PC does not have enough skill to detect traps. However, as discussed in the next section, there is a *Rogue* PC model who does not want the NPC to do anything regarding the traps. Performing "`[T]` Nothing" in this case does not necessarily mean that the PC has ignored the trap. It may mean that the PC has handled the trap, in which case

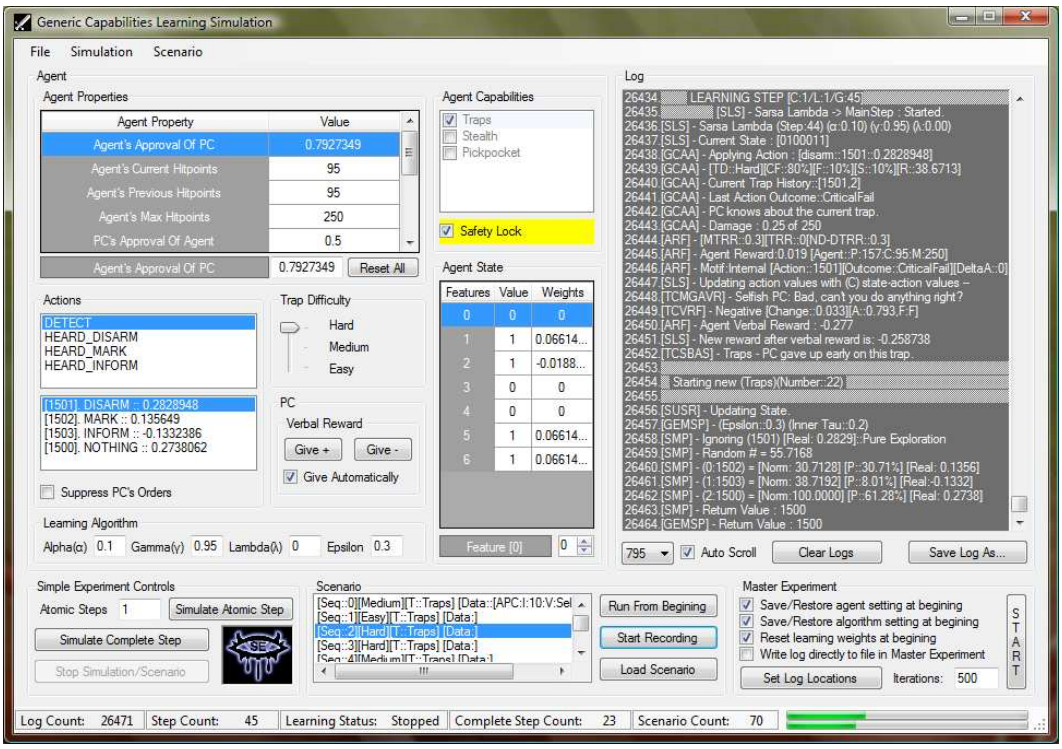

Figure 3.10: Simulator screen shot while running an experiment for the *Traps* decision domain

the NPC and the PC should move on to the next trap as well. The third reason for moving on to the next trap occurs when the PC and NPC do not disarm a trap. For example, if the PC constantly gives an order to the NPC and the NPC always refuses, there should be a threshold after which a human player would give up on that trap. This threshold is called "*Maximum Trap Effort*". In the simulations, the value of this parameter is set to 3 which means that after the NPC received 3 orders from the PC, both the PC and NPC will move on to the next trap regardless of the state of the current trap. The final reason for leaving a trap is called the "*Early Give Up Chance*". In order to make the behavior of the simulated PC character as close as possible to a real human, there is a 10% chance of giving up early on the trap and moving on to the next trap before even reaching the "*Maximum Trap Effort*". This simulates the situation where the player wants the PC to ignore the trap and proceed in a different direction.

3.4.1 PC Models

The first advantage of using the simulator is that it enabled us to keep the game mechanics and shut off the graphics. In order to use the full potential of the simulation and to be consistent through the simulation process, the NPC-environment and the NPC-PC interaction

should be automated. These interactions are already partially automated from the NPC's side. The part that remained manual is the feedback of the PC to the NPC, the automation of which is discussed in this section. Faster simulation and consistent feedback are the two main reasons for this automation. In order to achieve this, there are 4 distinguished PC characters that are considered in the simulations. These characters are generalized examples of different, and to some extent extreme, player behaviors. It is apparent that any player can play the game differently and the PC's behavior can be one of these or something in between these behavior models. The learning system is not required to know the PC model beforehand, and the NPC will learn to adapt to the behavior that the PC exhibits. The learning process is completely independent of which PC model is used and the only relation between the PC model and the learning is the nature of PC orders and feedback. The 4 PC models in the simulation process are the *Independent PC*, *Selfish PC*, *Rogue PC*, and the *Cautious PC*.

An *Independent PC* wants the NPC to be independent. This PC never gives orders to the NPC. The only verbal reward this PC gives to the NPC is a positive reward which is only given after "`[T]` Mark" with *Success* outcome and "`[T]` Disarm" with *Success* outcome. After other actions and other action outcomes this PC is always silent. This PC never discourages the NPC from doing something. For example, if the NPC is attempting to disarm hard traps and is taking a lot of damage, this PC never complains or says anything to show that the NPC should not attempt to disarm hard traps. The *Independent PC* models a human player who does not want to interact much with the NPC but appreciates marking and disarming of traps.

The *Rogue PC* wants to personally disarm or mark all the traps. An important part of the game play for a human player who plays a rogue character, according to rogue capabilities, is to handle traps. This PC gives negative verbal reward after any attempt, whether successful or not, to disarm or mark traps. A *Rogue PC* never gives any command to the NPC and never gives any verbal reward for other actions. For example, in the actual learning for the *Traps* decision domain, if the NPC detects a trap and performs action "`[T]` Inform PC", the PC will not give a verbal reward which results in a silent reward, r_{silent}, for the NPC. If the NPC after detecting a trap decides to disarm the trap, regardless of the outcome of the "`[T]` Disarm" action, the NPC will observe a negative verbal reward from the PC.

The *Selfish PC* wants the NPC to disarm all the traps, no matter what negative consequences occur for the NPC. This PC does not care for the NPC and does not accept anything less than a disarmed trap. This PC always give positive verbal reward after the

"[T] Disarm" action with the *Success* outcome. The *Selfish PC* will always give a negative verbal reward after an inform or an unsuccessful attempt to mark or disarm a trap by the NPC. This PC always gives the disarm trap order to the NPC and does not care if the NPC takes damage constantly by obeying this order. Note that an NPC ignores all verbal feedback following the "[T] Refuse" action. Although a negative verbal reward shows that the PC disagrees with the NPC's refusal, the NPC already knows this. The NPC does not discover any new information by taking this negative verbal reward into account. Any positive verbal reward after a "[T] Refuse" action is irrelevant and if it exists, it shows a paradox in the PC's character.

The *Cautious PC* cares about the NPC and tries to understand the level of the NPC's rogue skills. This PC wants to educate the NPC and prevent the NPC from taking further damage. The *Cautious PC* will give negative verbal reward for a *Critical Fail* outcome after both "[T] Mark" and "[T] Disarm" actions. The positive verbal reward is given for a *Success* outcome after those actions and if the outcome is a *Fail*, the PC will be silent. If the NPC has informed the PC about the trap, there is 25% chance that this PC gives the marking order. The same thing happens after a successful "[T] Mark" action, which means that there is 25% chance that the *Cautious PC* gives a disarming order. The rest of the times the PC will not give an order, instead the PC might say something like "I respect your decision and lets leave the trap." without giving any further orders. If the NPC tried to disarm the trap and failed, this PC will move on to the next trap.

Chapter 4

Experiments and Evaluation

THE main obstacle in using reinforcement learning in computer games is the speed of adaptation. This speed is especially important when dealing with a dynamic environment as in story based games. In this case, the environment includes both the physical objects with which the NPC can interact, and the emotional values that affect the NPC's decisions such as *approval* factor. The experiments focus on evaluating the speed of adaptation of the NPC's behavior to the changes in the environment.

4.1 Stealth Decision Domain

The goal in the *Stealth* decision domain is to enable the NPC to build a model of the PC's preferences. The experiments for this decision domain are set up in a particular way to evaluate how successful the NPC is in achieving this goal.

For the *Stealth* decision domain, both the Sarsa(λ) algorithm and ALeRT [6] algorithms were used. Experiments were conducted to measure and compare the performances of Sarsa(λ) and ALeRT, using different parameters.

The experiments for the *Stealth* decision domain were conducted in the Neverwinter Nights environment. The module used for these experiments contains two areas. One is an outdoor area near a castle and the other one is an indoor area inside the castle. Since the "Area Transition" is one of the decision events for the *Stealth* decision domain, and moving inside and outside the castle is considered to be an "Area Transition", the PC is programmed to repeatedly go inside the castle and then come outside to force the following NPC to make a decision regarding stealth mode. In order to evaluate the speed of adaptation, in the first 100 transitions, the PC prefers that the NPC be in stealth mode outside the castle and be in non-stealth mode inside the castle. After the 100^{th} transition, the PC will prefer that the

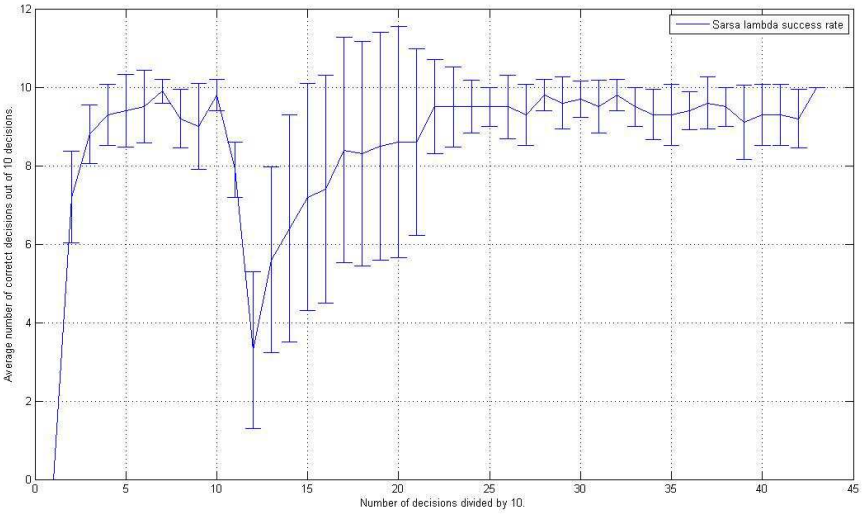

Figure 4.1: Sarsa(λ)'s success rate in the *Stealth* decision domain. The X axis shows the number of decisions divided by 10. The Y axis shows the number of correct decisions out of 10 decisions, averaged over 10 runs. $\lambda = 0.95$, $\gamma = 0.1$, $\epsilon = 0.1$, $\alpha = 0.05$.

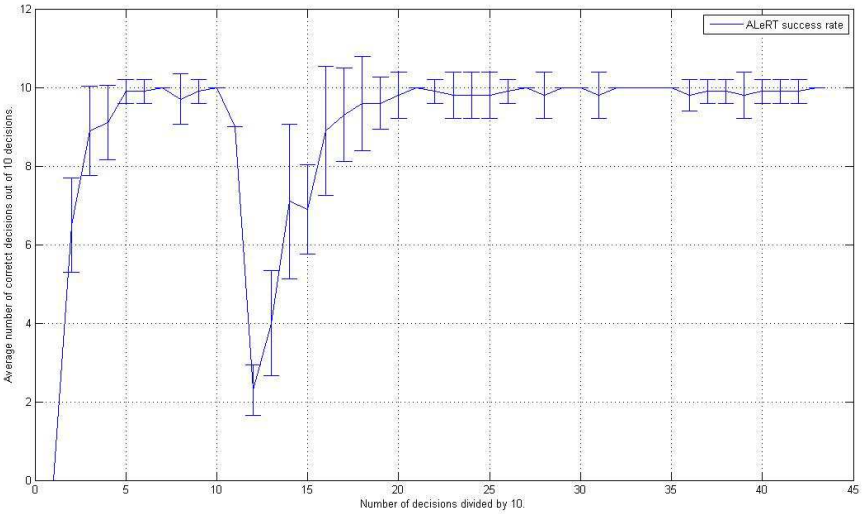

Figure 4.2: ALeRT's success rate in the *Stealth* decision domain. The X axis shows the number of decisions divided by 10. The Y axis shows the number of correct decisions out of 10 decisions, averaged over 10 runs. $\lambda = 0.95$, $\gamma = 0.1$, $Min(\epsilon) = 0.005$, $Max(\epsilon) = 0.1$, $Min(\alpha) = 0.01$, $Max(\alpha) = 0.1$.

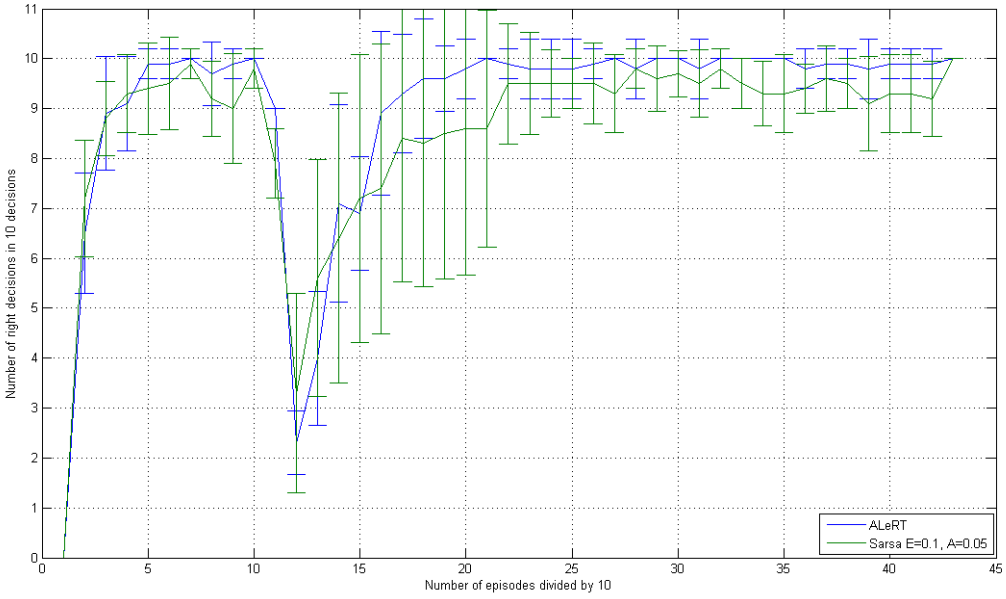

Figure 4.3: Comparison of Sarsa(λ) algorithm and ALeRT algorithm success rates in the *Stealth* decision domain.

NPC be in stealth mode inside the castle and be in non-stealth mode outside the castle. In the experiments for this decision domain, the NPC starts with zero knowledge of the PC's preferences.

The first algorithm used for this decision domain was Sarsa(λ). Figure 4.1 shows the success rate of this algorithm. The X axis of this diagram shows the number of decisions divided by 10. In a single run of this experiment, the NPC has the opportunity to make 410 decisions in response to the decision events of the *Stealth* decision domain. Since the decision triggers in this decision domain can be very frequent, 410 decision events were triggered in a short period of time. These decision events were divided into 41 groups of 10. The Y axis in Figure 4.1 shows the number of correct decisions out of 10 decisions for each of the 41 groups. In fact, this diagram shows the average over 10 runs for this experiment. Each experiment took approximately 25 minutes. Since, "Timed" decision events occur every 2 minutes in this decision domain, the 410 decision events include 12 or 13 "Timed" decision events. The Simulator was used in the *Traps* domain to avoid long duration experiments (see Section 3.4). Recall that the preference of the PC is reversed after the first 100 decisions (10 data points). The big fluctuation that starts from the 10^{th} data point shows the amount of time the NPC requires to unlearn the policy learned so far and learn the new policy. The action selection policy used for this algorithm is ϵ-*greedy* with ϵ equals to 0.1.

The next algorithm used for this decision domain is the ALeRT algorithm. Figure 4.2 shows the success rate for this algorithm. This diagram follows the same principles as the Sarsa(λ)'s diagram in Figure 4.1. The ϵ and α vary daynamically in ALeRT. The minimum value of ϵ is 0.005 and the maximum value of ϵ is 0.2. The minimum value of α is 0.01 and the maximum value of α is 0.1. The range of α is divided into 20 steps and the range of ϵ is divided into 15 steps. Since the ALeRT algorithm is capable of detecting trends, the windows size for trend detection must be specified and was set to 4. Figure 4.3 shows the results of both algorithms on the same set of axes.

To statistically compare the performance of Sarsa(λ) and ALeRT, a Welch's t-test was performed to test the null hypothesis: the average success rate of ALeRT is greater than or equal to the average success rate of Sarsa(λ). The number of successful decisions out of the 411 decision events was computed for each of the 10 runs for each algorithm. This produced 10 success rates for each algorithm. The mean success rate for ALeRT was 379.3 with standard deviation 9.2. The mean success rate for Sarsa(λ) was 358.4 with standard deviation 29.5. The result of a one-tailed t-test indicated that the null hypothesis could not be rejected at 95% confidence.

4.2 Traps Decision Domain

The goal of the *Traps* decision domain is to enable the NPC to exhibit rapidly adapting behaviors regarding traps in the game. The environment in this domain is dynamic and changes fast. Since it is possible for the NPC to take damage, it is important that the NPC adapt to the changes in the environment very fast in order to prevent taking further damage and to maximize the possible experience points.

Many experiments were conducted with a variable number of traps and variable trap difficulties. The Sarsa(λ) learning algorithm was used for this decision domain. Although the ALeRT algorithm had better performance in the *Stealth* domain, the difference was not large and Sarsa(λ) has fewer parameters to tune. After trying different learning parameters in order to get good results, the learning parameters for Sarsa(λ) were fixed to $\alpha = 0.1$, $\gamma = 0.95$, $\lambda = 0$, and since the GESM policy is used in this domain, the policy parameters were fixed to $\epsilon = 0.3$ and $\tau = 0.2$. The simulator (see Section 3.4) was used to reduce the time necessary to run the experiments. In the experiments for this domain, an NPC starts with zero knowledge of the traps, knowing only the set of legal actions. The common PC behaviors are modeled using four different PC models (see Section 3.4.1). The results for

each of these behavior models are presented separately.

The action selection policy used in this domain, GESM, selects the highest action during exploitation and selects one of the other actions probabilistically based on relative state-action values during exploration. The relative values 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, NPC's responses to both trap difficulty changes and PC *approval* changes are tested in these experiments. In each experiment, the NPC and the PC encounter 40 traps. There are three sets of experiments conducted to test the effectiveness of various aspects of the *Traps* decision domain. Some of the parameters are fixed in some experiments to test both the effect of that particular fixed value and the effect of other aspects of learning. All the results presented in the *Traps* decision domain are averaged over 500 runs.

In the first set of experiments, the *approval* is fixed. In these experiments, the difficulties of the traps change after each five traps. For example, the *approval* can be fixed to 0.2 and then the NPC will face 5 easy traps, 5 hard traps, 5 easy traps, and so on. Each experiment will test the responsiveness of the NPCs to the changes in trap difficulties for each fixed *approval* and PC model. In this research, experiments were conducted for 0.0, 0.2, 0.5, 0.8, and 1.0 *approvals*, for each of the 4 PC models. The text only contains results for *approvals* of 0.2 and 0.8. Results for other *approvals* were as expected. This kind of cyclic trap difficulty is not how the traps are usually set up in games. However, this scenario is constructed specifically to validate fast adaptability.

In the second set of experiments, the trap difficulty is held fixed and the NPC's *approval* of the PC changes after every 5 traps. For example, an experiment contains 40 medium traps and the *approval* starts with 0.2. After 5 traps it is changed to 0.8 and then back to 0.2 after encountering another 5 traps and so on. This simulates changes in approval due to other events occurring in the game that drive changes in the emotional state of the NPC. The goal is to see if the NPC's behavior changes accordingly. These experiments assess the effect of the changes in the *approval* on the behavior exhibited by the NPC. They also test the speed of adaptation of the NPC's behaviors to the changes in *approval*. The combination of PC models and the difficulty of the traps in these experiments should affect the NPC's behavior as well.

The third set of experiments assess the NPC's behavior in a real game situation. None of the parameters are held fixed in these experiments. The NPC encounters traps with

different difficulties and the NPC's *approval* changes according to Equation 3.1. In these experiments, the effect of the different PC behaviors which are modeled with PC models can be clearly seen. These experiments show that the PC can train the NPC to exhibit a particular behavior. However, the PC should be careful about the NPC's *approval* factor. If the *approval* drops to a low value, it would be hard for the PC to regain the NPC's trust and to train the NPC. In these experiments, the NPC first faces 10 easy traps, then 10 medium, and finally 20 hard traps. The NPC's approval of the PC starts at either 0.5 or 0.2, and varies afterwards based on the behaviors of both the PC and NPC.

The results presented in the diagrams in this chapter are normalized. During exploitation the action with the highest $Q(s, a)$ value is selected. However, during exploration the $Q(s, a)$ values must be transformed to relative probabilities using softmax and the Gibbs distribution. Although the action with the highest $Q(s, a)$ value is never selected, it is useful to represent it on the diagram with the probabilities of the other actions to indicate relative probability that it would be selected if it was eligible for selection. The normalized diagrams show the relative probabilities for selecting all of the actions as computed by the Gibbs distribution. For example, in Figure 4.4, at trap number 5, the probability of "[T] Disarm" is approximately 0.2154 and the probabilities of "[T] Mark", "[T] Inform PC", and "[T] Nothing" are approximately 0.1955, 0.1952, and 0.1950 respectively. The probability of the fifth action, "[T] Refuse", is approximately 0.1989, so the probabilities add to 1.0. Since the "[T] Disarm" is the first choice it will not actually be selected during exploration. Since "[T] Refuse" is not appropriate for this decision event, it cannot be selected during this exploration. Therefore the relative probabilities of the three selectable actions are used to generate the exploration action. For the results before normalization with standard deviation bars please see Appendix A. Although these diagrams show average results, in a real game situation the player will see a single trajectory. Single trajectories are shown in Appendix B.

4.2.1 Independent PC Results

Figure 4.4 (see also Figure A.1) shows the result for the low *approval* case in the first set of experiments. In this experiment the *approval* is fixed to 0.2 and the difficulty of the traps are switches back and forth after each 5 traps between easy and hard. The *Independent* PC does not try to train the NPC in any particular way. As a result, at the beginning of the experiment the NPC likes to disarm easy traps and tries to avoid hard traps. However, after a short period of time, the NPC realizes that the amount of damage taken from

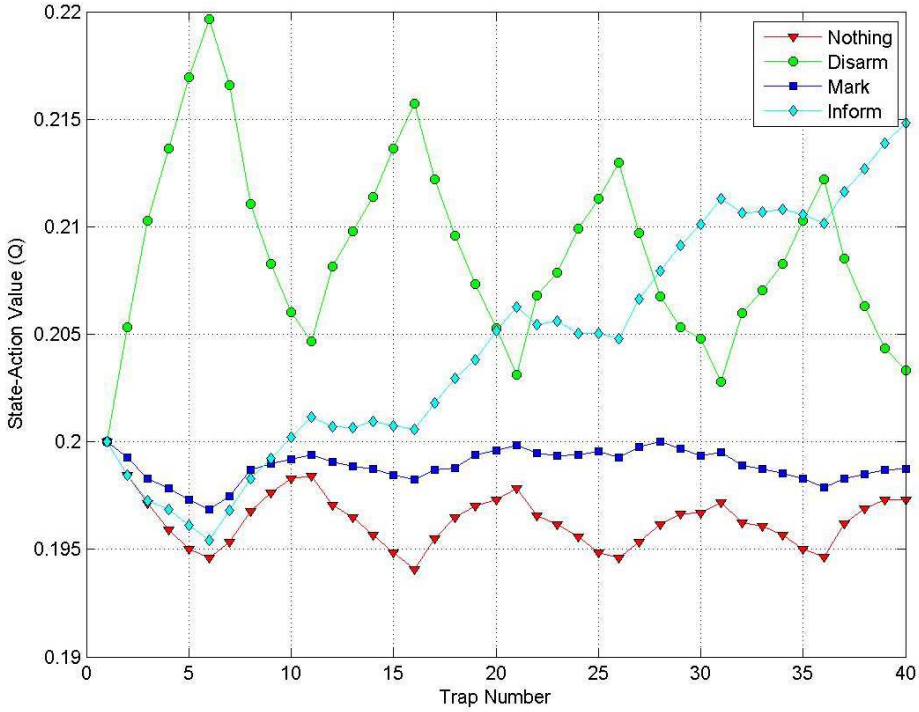

Figure 4.4: The *Traps* domain results for a *Independent* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.2 and the results are normalized.

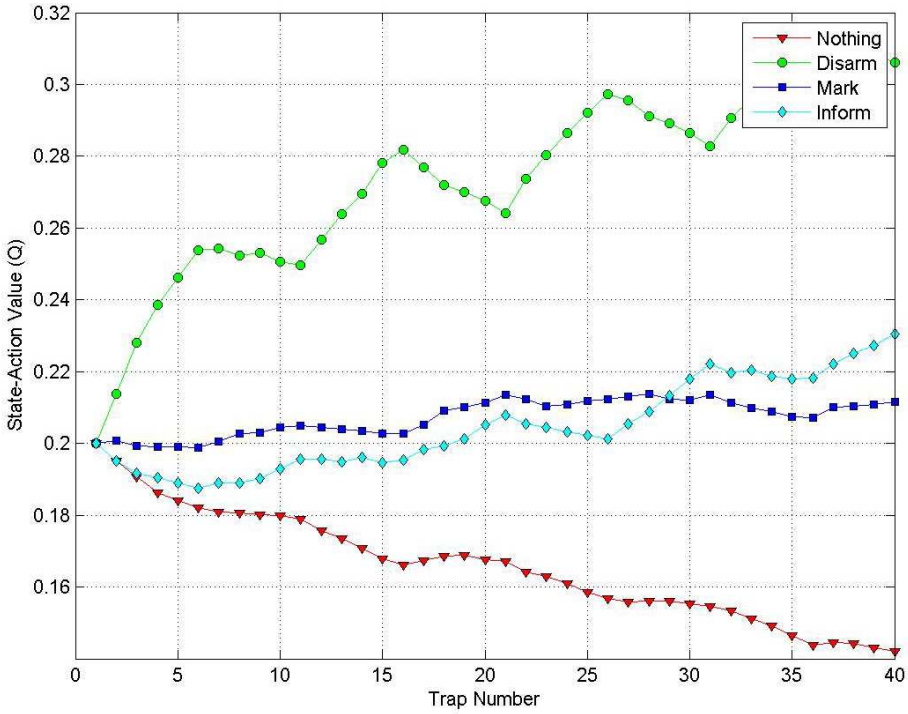

Figure 4.5: The *Traps* domain results for a *Independent* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.8 and the results are normalized.

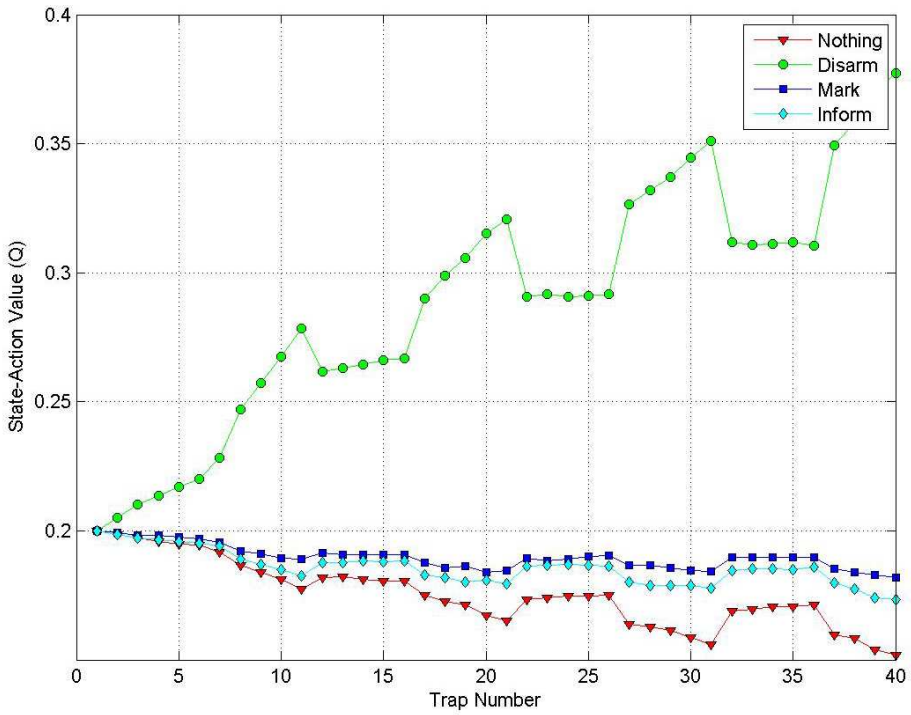

Figure 4.6: The *Traps* domain results for a *Independent* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are easy traps and the results are normalized.

hard traps is not worth the amount of XP and *approval* reward gained, especially since the NPC has a low approval of the PC which discounts the *approval* reward. In this case, the NPC does not really care what might happen to the PC. Therefore, the NPC learns to inform the PC instead. Consequently, as long as the PC does not force the NPC to disarm or mark traps, letting the PC know about the traps is not harmful, so it is chosen. It can be seen that "[T] Mark" and "[T] Nothing" actions have low values. The reason for "[T] Mark" action's low $Q(s, a)$ value is that the NPC does not gain any XP by performing the "[T] Mark" action and there is a possibility that the NPC takes damage while performing this action. Action "[T] Nothing" is the same as "[T] Inform PC" in terms of XP and damage. However, in the NPC's mind, informing a harmless PC about the trap might help build a better relationship with the PC (The reward system includes a higher revelation factor for the "[T] Inform PC" than for "[T] Nothing"). On the contrary, doing nothing does not help anyone.

Figure 4.5 (see also Figure A.2) shows the results for the same type of experiment in which trap difficulty is switched between easy and hard after each 5 traps. However, in this case the *approval* is fixed to 0.8. In this experiment, since the *approval* factor is high, the

NPC does not want the PC to take damage. Therefore, action "[T] Disarm" is the most preferred one, even late in the experiment. Recall that an NPC with high approval of the PC is willing to take some damage, depending on the value of the *approval*, to prevent the PC from taking damage.

In the second set of experiments, the trap difficulty is constant and the *approval* switches back and forth between 0.2 and 0.8. The results of the first experiment, where the traps are easy, is shown in figure 4.6 (see also Figure A.3). Since the traps are easy, the NPC prefers to disarm the traps and gain XP. It can be seen that in the low *approval* phases, the slope of the $Q(s, a)$ value for the "[T] Disarm" action is close to 0. The reason that the slope is not positive is that there is still a small chance of taking damage from easy traps that is offset by a high *approval*. This occasional damage dampens the joy of gaining XP for the NPC. As a result, the state-action value for the "[T] Disarm" action remains constant in those phases. Moreover, it can be seen that all other actions have approximately the same value and therefore the same chance of being selected during exploration. This can be compared with the next experiment where the traps are all medium and the *approval* still switches back and forth between 0.2 and 0.8. The normalized state-action values of different actions after detecting a trap in this experiment are shown in Figure 4.7 (see also Figure A.4). In this experiment, since the chance of taking damage is higher than in the easy traps experiment, although the NPC still decides to perform the "[T] Disarm" action, the value of "[T] Mark" and "[T] Inform PC" are higher, when compared to the easy trap case.

Figure 4.8 (see also Figure A.5) shows the constant trap difficulty experiment for the hard traps and the *Independent* PC. This diagram shows that in the case of hard traps the NPC prefers to only inform the PC, to avoid taking damage. However, the NPC understands (from experience) that the PC does not give commands (unlike a *Selfish* PC). Therefore, there is no downside with letting the PC know about the traps which might ultimately prevent the PC from taking damage. A player might infer that the NPC is protecting the PC so that the PC can save some healing potions for the NPC.

Figure 4.9 (see also Figure A.6) shows the $Q(s, a)$ values for different actions after detecting a trap in the third kind of experiments, where the first 10 traps are easy, the next 10 medium, and the last 20 are hard. In the previous experiments the *approval* was fixed by external factors that overrode the changes due to the trap decisions. In these experiments the *approval* varies due to the actions of the PC and NPC in the *Traps* domain and external factors are ignored. The changes in *approval* for the experiment shown in Figure 4.9 can

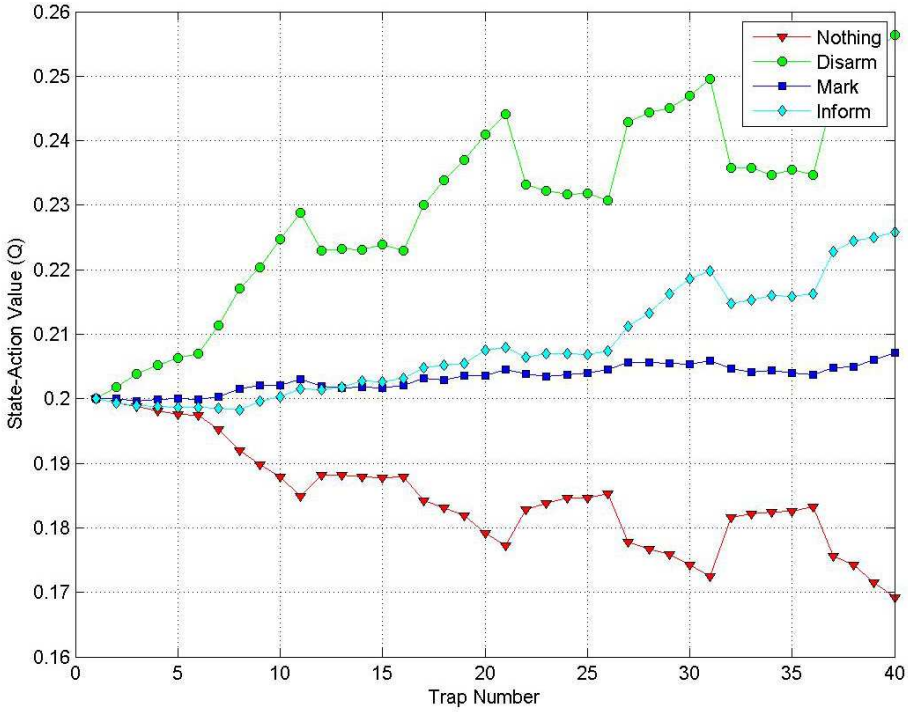

Figure 4.7: The *Traps* domain results for a *Independent* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are medium traps and the results are normalized.

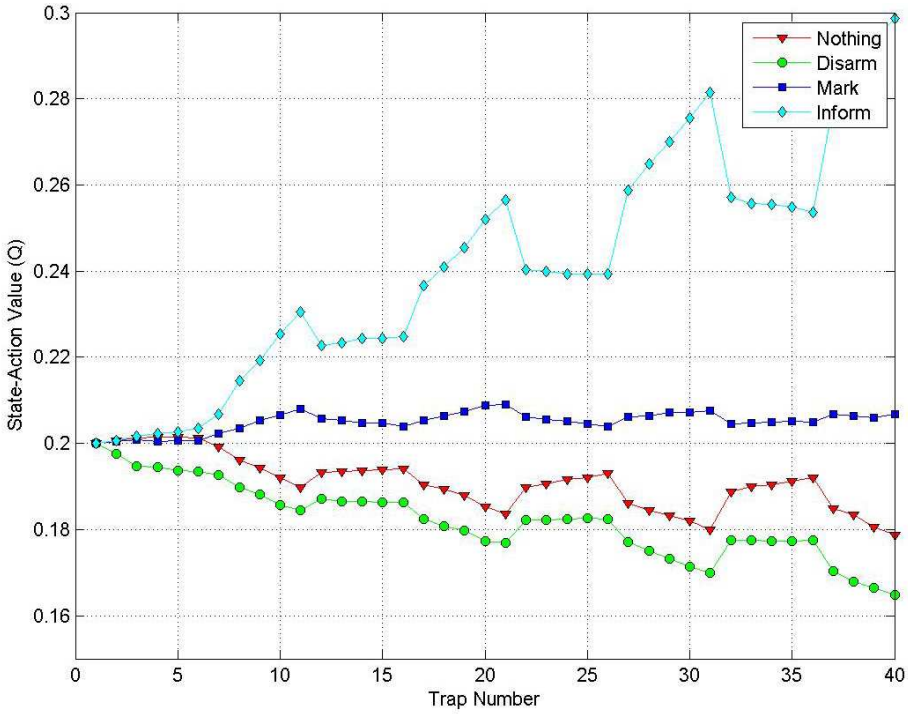

Figure 4.8: The *Traps* domain results for a *Independent* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are hard traps and the results are normalized.

be seen in Figure 4.10. During the first 10 traps, the NPC decides to disarm the traps. The *Independent* PC only gives positive rewards and only for successful "[T] Disarm" or "[T] Mark" actions. Since the first 10 traps are easy, the NPC is usually successful so the NPC starts to like the PC. By the time the NPC encounters the medium and hard traps, the *approval* that the NPC has for the PC is already high. Moreover, the PC does not give any orders or negative reward. Consequently, the NPC continues disarming the traps and liking the PC.

4.2.2 Rogue PC Results

Figure 4.11 (see also Figure A.7) shows alternating 5 easy and 5 hard traps for a *Rogue* PC with low *approval* (0.2). A *Rogue* PC wants the NPC to only inform about traps so that the PC can disarm or mark all the traps personally. After 7 traps, the NPC learns to inform the PC about all traps. 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 or mark all the rest (In the game the NPC might say resignedly "Fine, all the traps are yours, do what you want and I will watch!").

The results are similar for a high *approval* PC (see Figures 4.12 and A.8), except that it only takes a single trap for the PC to convince the NPC, due to the high *approval* (In the game the NPC might say "As you wish, but let me know whenever you need help!"). It can be seen that in the high *approval* case, the PC is successful in training the NPC not to disarm or mark the traps. By using the GESM action selection policy, the order of actions becomes important (see Section 3.2.2), which means that "[T] Nothing" in the second place has a higher chance of being selected than "[T] Disarm" and "[T] Mark" actions. The reason for action "[T] Nothing"'s high value is the way the PC has trained the high *approval* NPC not to mark or disarm the traps.

Figure 4.13 (see also Figure A.9) shows the first experiment in the second set of experiments for the *Rogue* PC, where the *approval* cycles and the trap difficulty is fixed. Since all the traps are easy, the NPC has no hesitation in disarming traps and gaining the XP. However, the *Rogue* PC tries to train the NPC not to disarm or mark the traps. This experiment shows that the NPC only listens to the PC during high *approval* phases. In low *approval* phases the NPC does not care much about PC orders or feedback. Since the NPC is also a *Rogue* and pursues the same goals as the PC, it can be seen that the NPC is unlearning

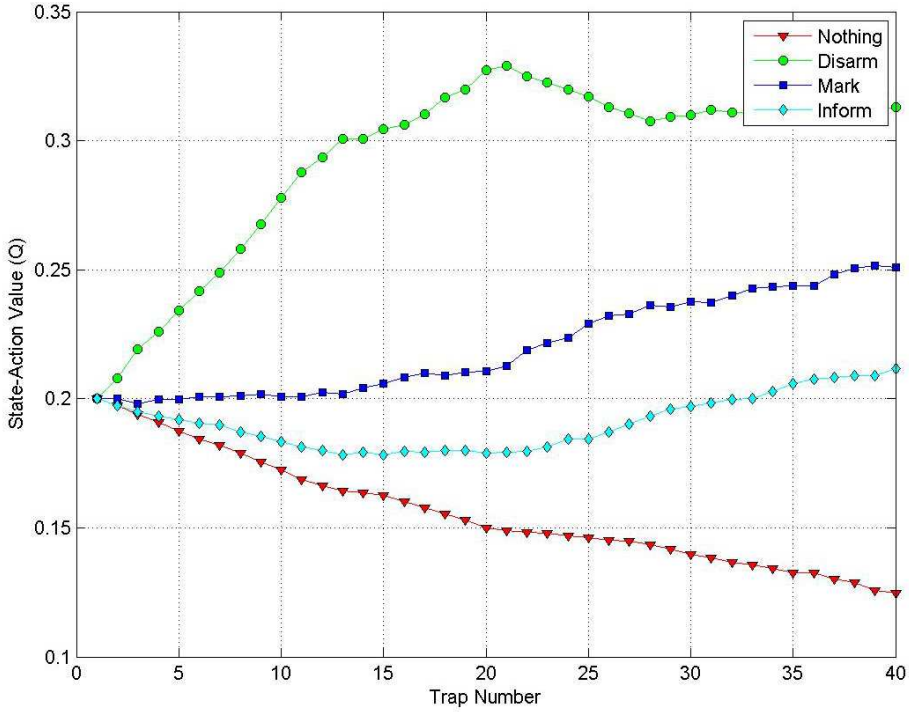

Figure 4.9: The *Traps* domain results for an *Independent* PC. In this experiment the *approval* starts from 0.5. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are normalized.

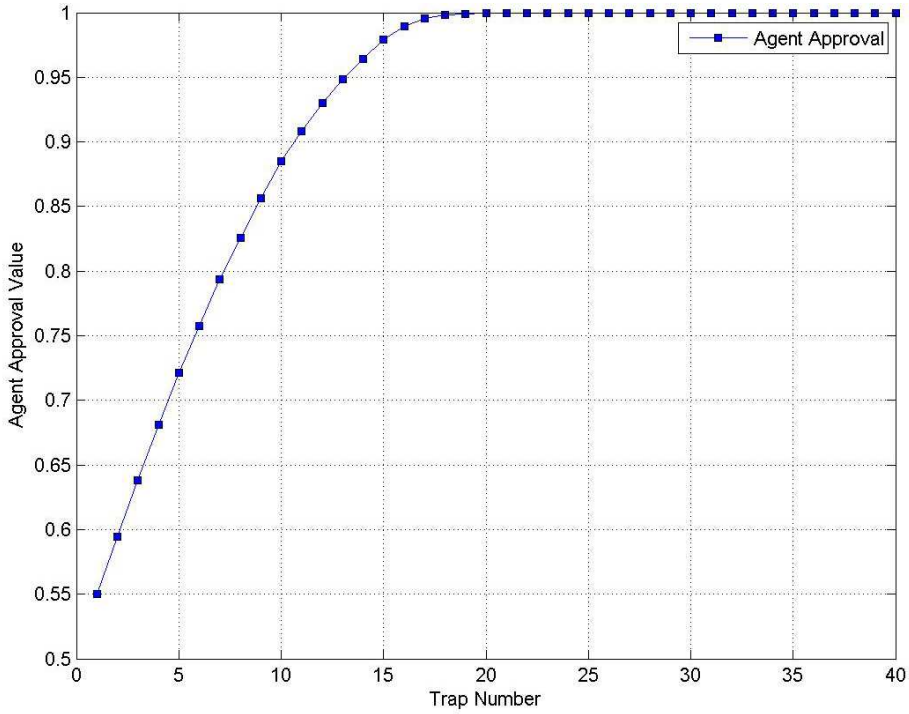

Figure 4.10: The *approval* in the *Traps* domain results for an *Independent* PC. In this experiment the *approval* starts from 0.5 and varies based on NPC and PC actions. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps.

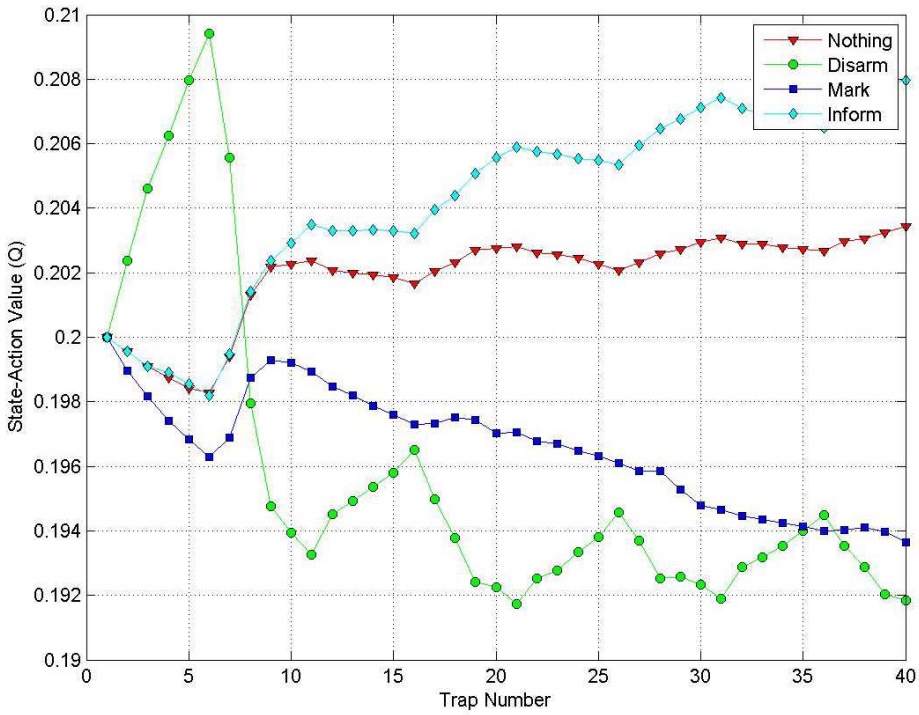

Figure 4.11: The *Traps* domain results for a *Rogue* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.2 and the results are normalized.

what has been learned in the low *approval* phases. Over the long run the NPC learns to only inform the PC about the traps since the verbal reward during high *approval* phases compensates for the lost XP during low *approval* phases. Moreover, the second choice of action for the NPC always switches between "`[T]` Nothing" for high *approval* and "`[T]` Disarm" for low *approval* phases.

In the case of medium traps, as the *approval* switches back and forth between 0.2 and 0.8, the NPC is more quickly convinced to let the *Rogue* PC do the marking and disarming of the traps, since they are harder than the easy traps. Figure 4.14 (see also Figure A.10) shows this fact.

Figure 4.15 (see also Figure A.11) shows the NPC facing 40 hard traps while switching the *approval* back and forth after every 5 traps between 0.2 and 0.8. In this case, the NPC does not want to take damage from the hard traps and the PC does not want the NPC to disarm or mark any traps. Although the NPC and PC follow different motivations for their decisions, in this experiment, they both have a common interest. The NPC does not want to disarm or mark the traps because the traps are hard and that is exactly what the PC wants. This behavior can be clearly seen in Figure 4.15. However, since the NPC does not see any

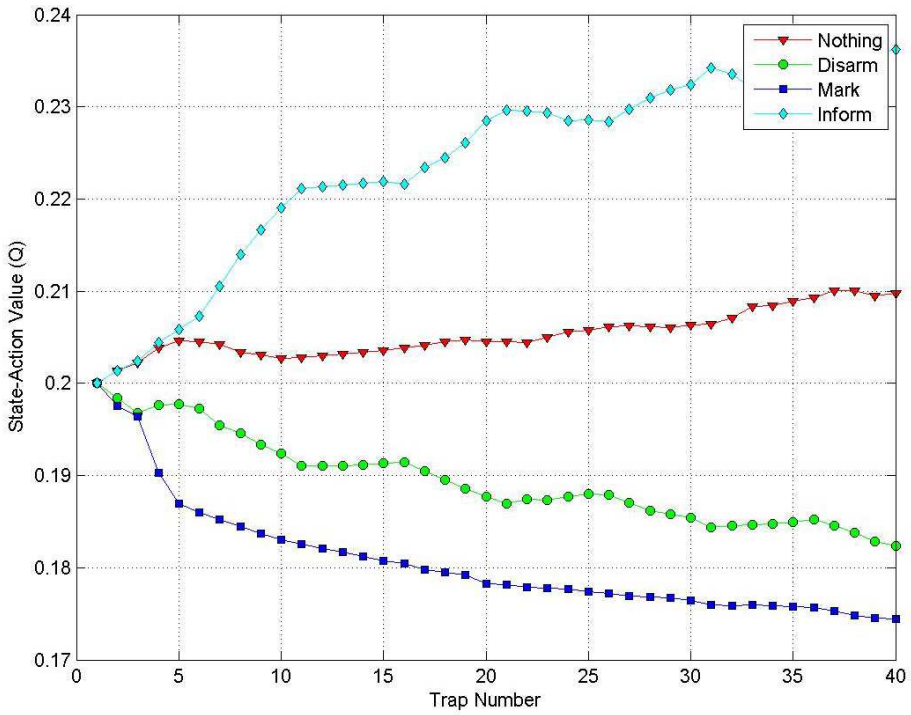

Figure 4.12: The *Traps* domain results for a *Rogue* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.8 and the results are normalized.

reason not to inform the PC about the traps, the $Q(s, a)$ values for the "[T] Inform PC" action goes up.

Figure 4.16 (see also Figure A.12) shows the third type of experiment for the *Rogue* PC. In this experiment the *approval* starts from 0.5 and varies only based PC-NPC interaction in the *Traps* domain afterwards. Figure 4.17 shows the corresponding changes in *approval* during this experiment. During the first 10 easy traps, the NPC tries to disarm the traps. Although the NPC observes that it is easy to gain XP from disarming, the PC provides negative feedback by discouraging the NPC from disarming. As a result of this disagreement the *approval* goes down. In the second 10 traps, which are of medium difficulty, the NPC realizes that disarming the traps has become harder and the "[T] Disarm" action becomes steadily less favorable compared to the other actions. In the last 20 traps, which are hard traps, the NPC abandons the "[T] Disarm" action, since it is more favorable to let the PC disarm the traps. The NPC's opinion now coincides with the PC's opinion about the traps. As a result the *approval* starts to go up. It can be seen in Figures 4.16 and 4.17 that when the *approval* goes up, the order of actions that the PC wants the NPC to learn is learned by the NPC. The PC wants the NPC to first perform "[T] Inform PC" if possible. The next

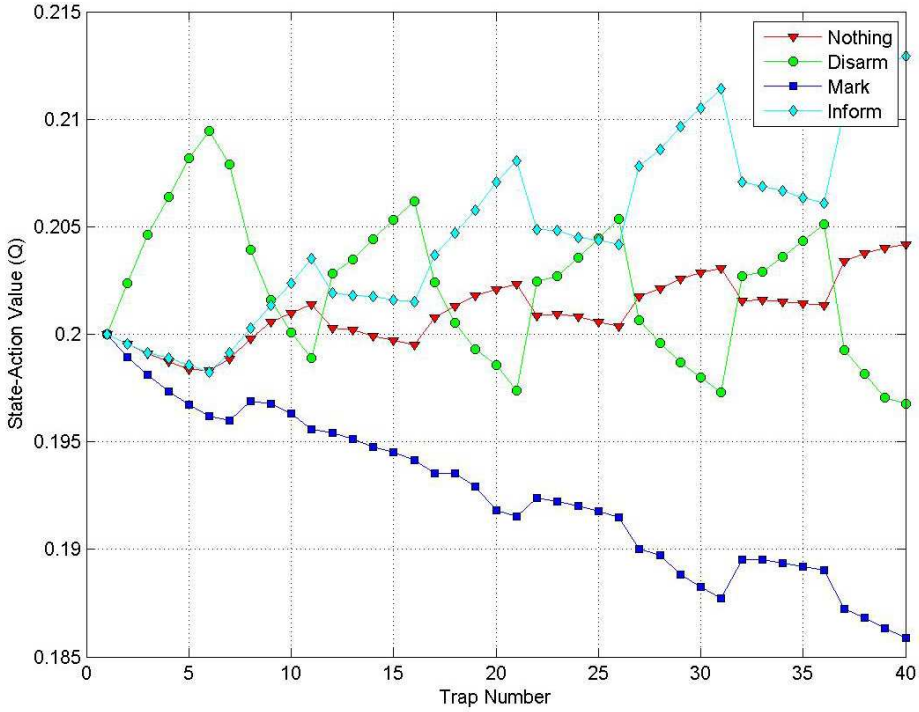

Figure 4.13: The *Traps* domain results for a *Rogue* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are easy traps and the results are normalized.

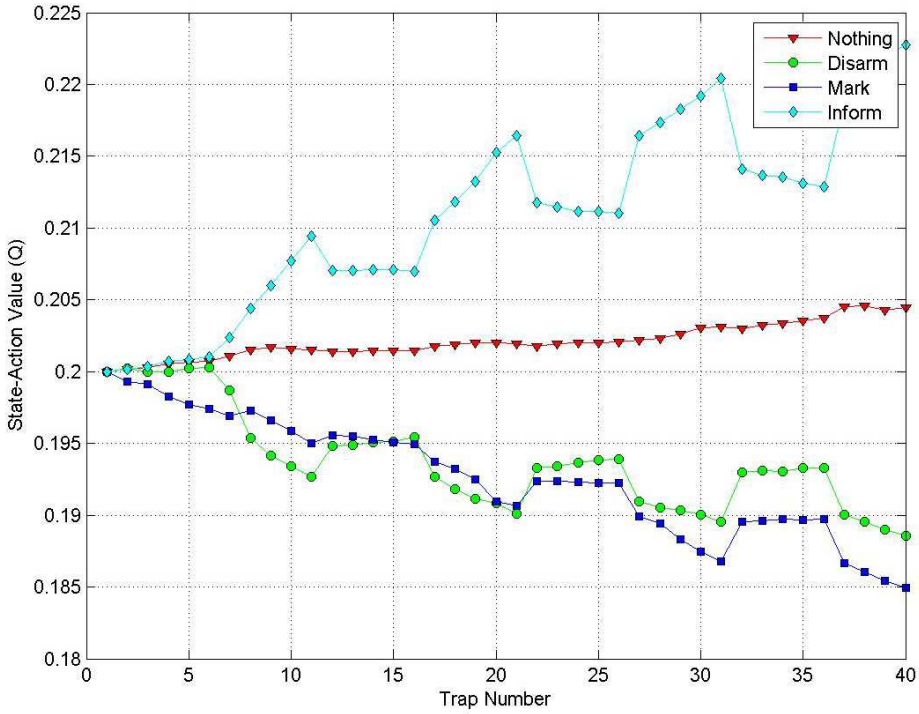

Figure 4.14: The *Traps* domain results for a *Rogue* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are medium traps and the results are normalized.

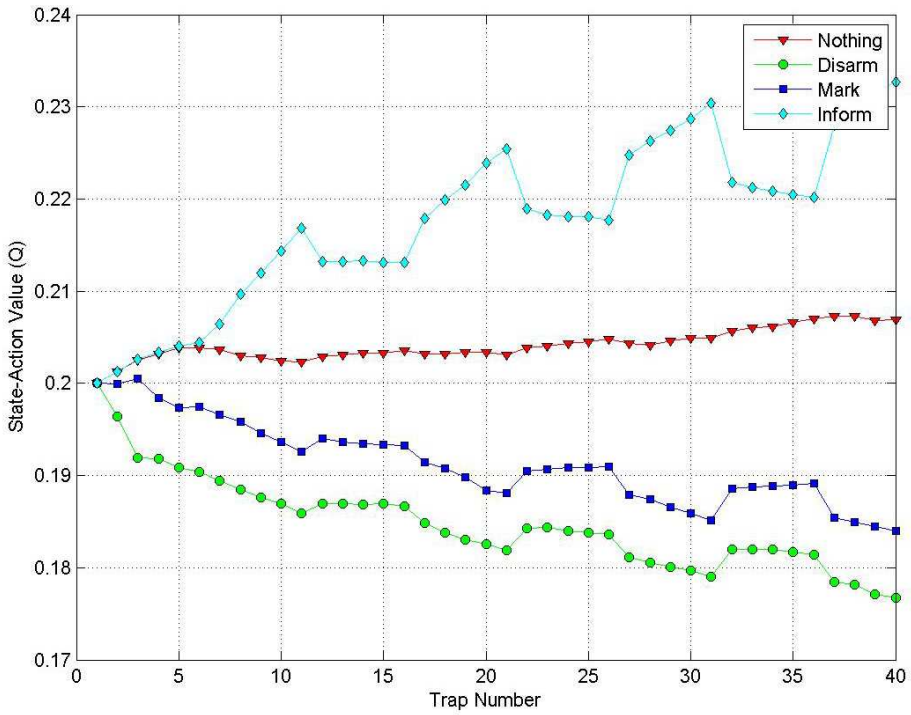

Figure 4.15: The *Traps* domain results for a *Rogue* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are hard traps and the results are normalized.

choice should be doing "[T] Nothing". The third and fourth choices should be "[T] Mark" and "[T] Disarm". The reason that this process took such a long time is that the NPC's approval of the PC had dropped to a very low value, due to initial disagreements. As a result, it took a longer time (see Equations 3.1) for the PC to regain the trust of the NPC. With low *approval*, even though the PC and the NPC have the same goal in mind, it takes a long time for the PC to be able to influence the choices of the NPC.

4.2.3 Selfish PC Results

Figure 4.18 (see also Figure A.13) shows alternating easy/hard traps for a *Selfish* PC with low (0.2) *approval*. The preferred action order is based on the difficulty of the traps, the PC's behavior, and the NPC's approval of the PC. For hard traps in this experiment, the learned action preference order is to do "[T] Nothing", then "[T] Mark", then "[T] Inform PC" and then "[T] 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: "[T] Disarm", then "[T] Nothing", then "[T] Mark" and finally "[T] Inform PC". The reason the NPC prefers to disarm rather

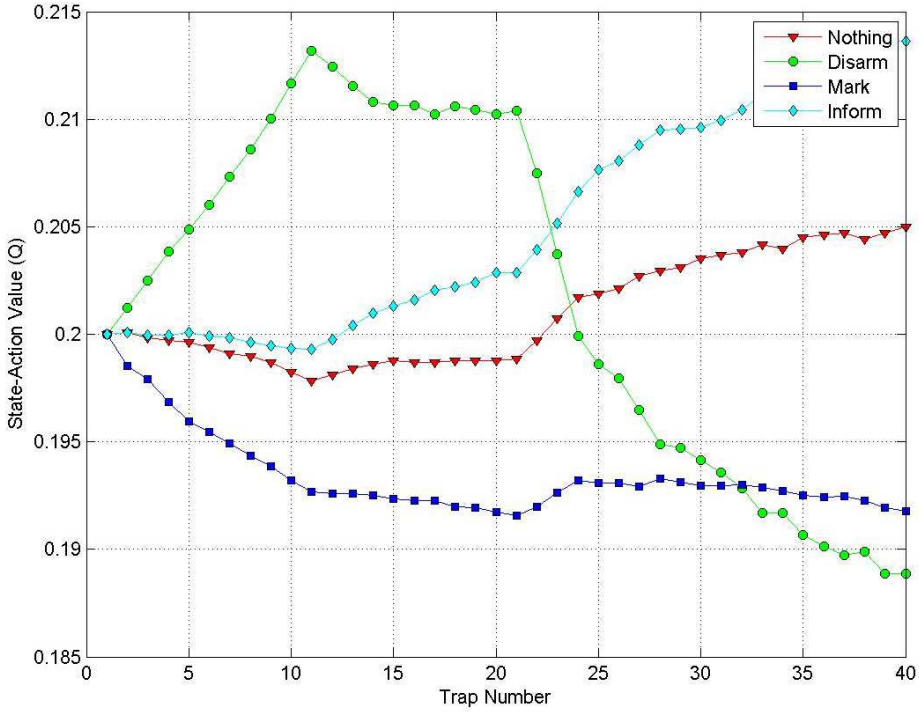

Figure 4.16: The *Traps* domain results for a *Rogue* PC. In this experiment the *approval* starts from 0.5 and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are normalized.

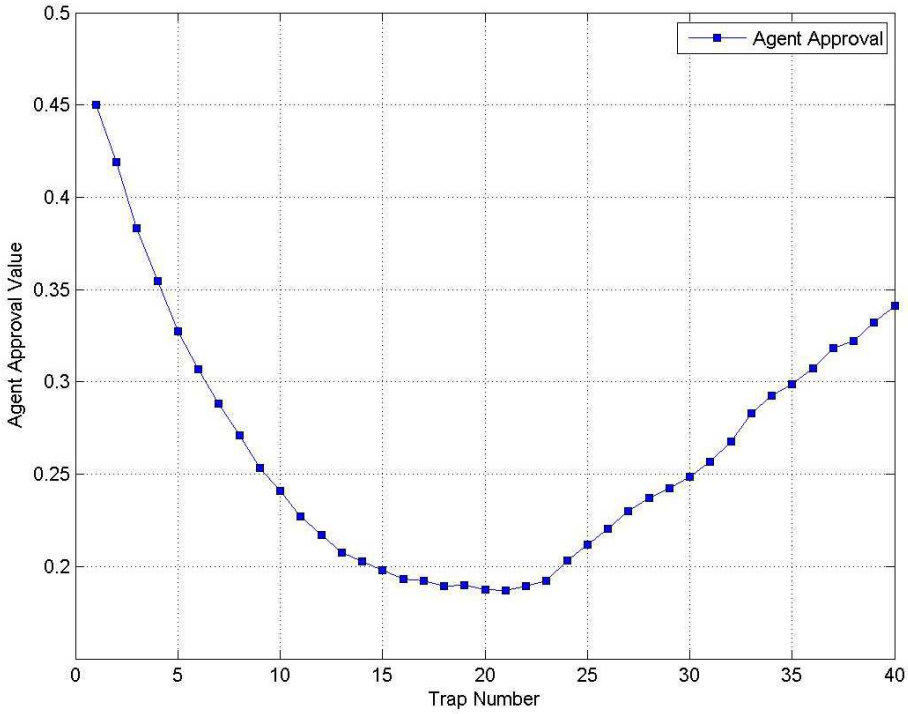

Figure 4.17: The *approval* in the *Traps* domain results for a *Rogue* PC. In this experiment the *approval* starts from 0.5 and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps.

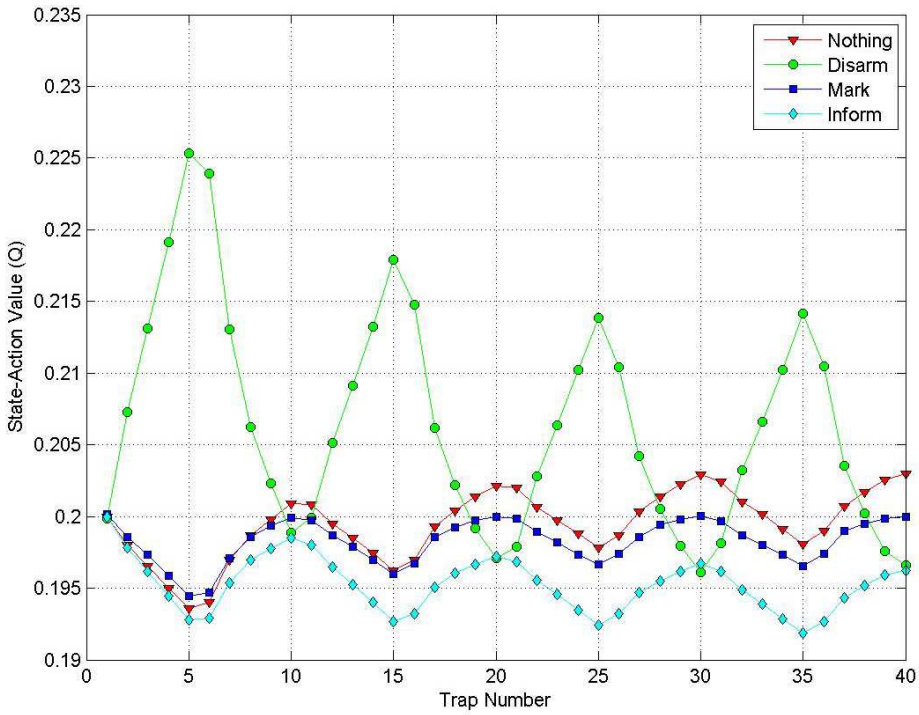

Figure 4.18: The *Traps* domain results for a *Selfish* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.2 and the results are normalized.

than mark is that disarming yields XP while marking does not. In general, the *Selfish* PC always orders the NPC to disarm all traps. However, since the NPC has a low approval of the PC, the NPC ignores the PC's orders because they can lead to taking damage.

Figure 4.19 illustrates how the NPC responds to the disarm command from the PC. This result is for the same switching easy and hard traps, low *approval*, *Selfish* PC experiment shown in Figure 4.18. It shows what the NPC would do in response to being ordered 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 4.20 (see also Figure A.15) illustrates different state-action values for the NPC after detecting a trap. In this experiment the *approval* is fixed to 0.8 and the trap difficulty alternates after each 5 traps between easy and hard. Since in this experiment the *approval* factor is high, the NPC prefers to do what the PC wants, which is to always disarm the traps. The oscillation in the diagram is caused by the change in the difficulty of the traps. However, since the NPC likes the PC, the NPC does is willing to take damage for the PC. In this case, the high *approval* produced a big advantage for the "`[T]` Disarm" action's $Q(s, a)$ value regardless of the possible damage.

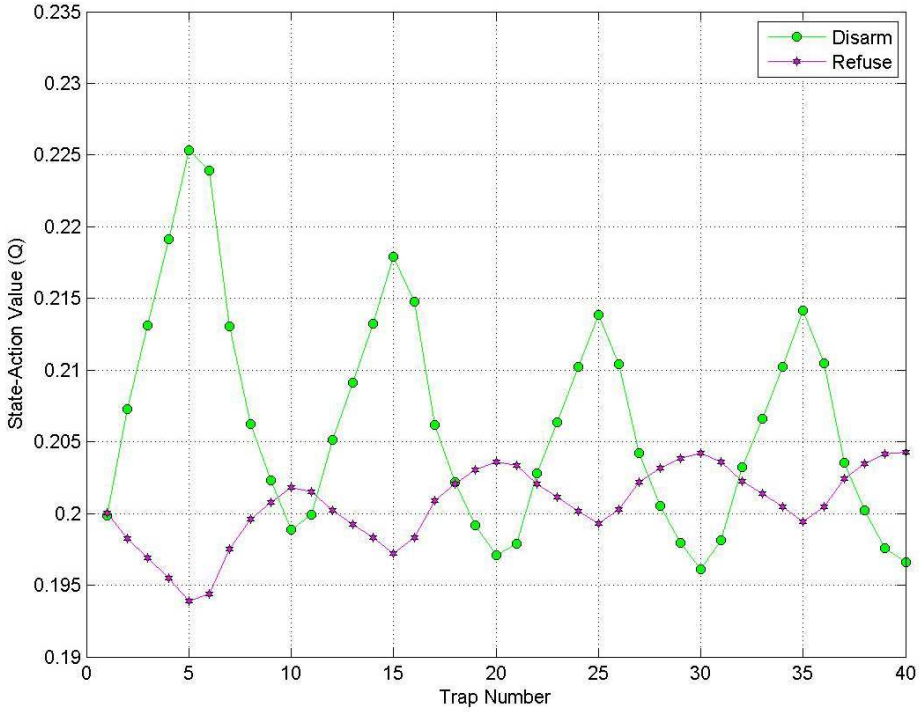

Figure 4.19: The *Traps* domain results for a *Selfish* PC while receiving a disarm order. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard traps. The *approval* is fixed to 0.2 and the results are normalized.

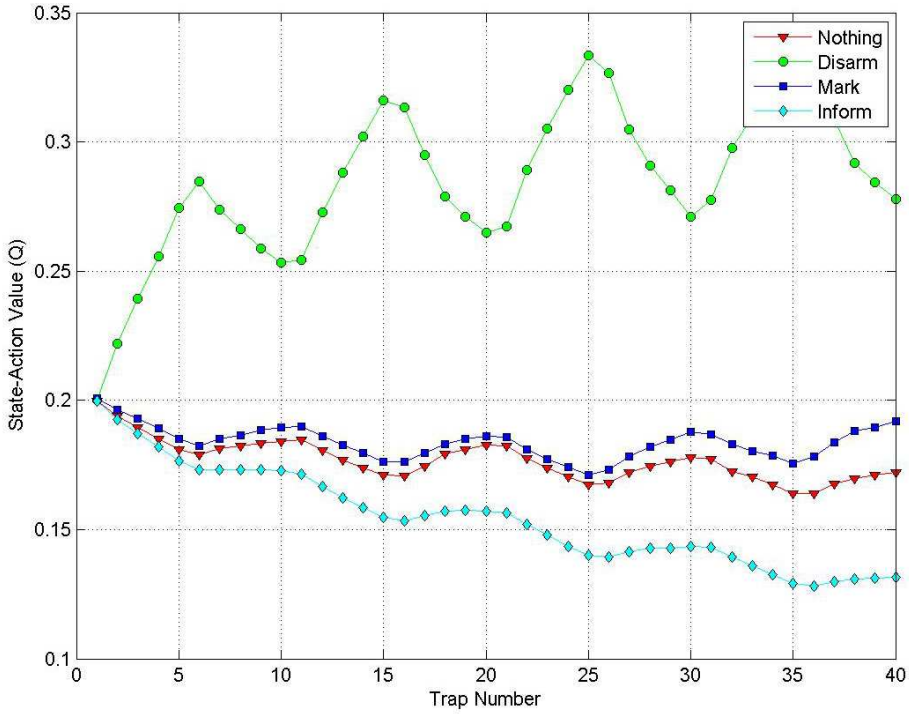

Figure 4.20: The *Traps* domain results for a *Selfish* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.8 and the results are normalized.

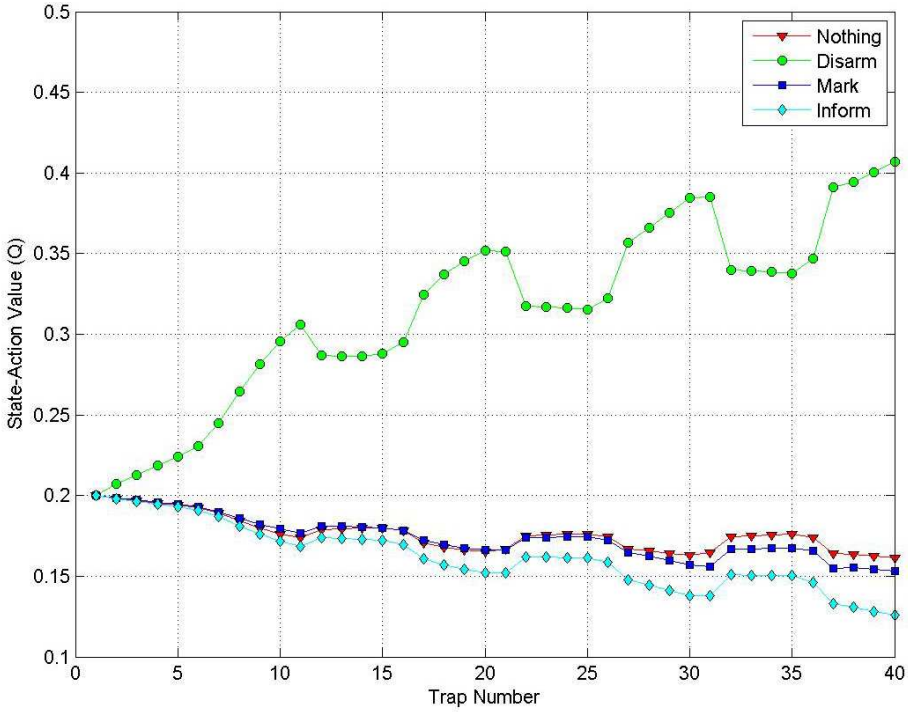

Figure 4.21: The *Traps* domain results for a *Selfish* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are easy traps and the results are normalized.

Figure 4.21 (see also Figure A.15) shows the first experiment of the second set of experiments. In this experiment the trap difficulty is fixed at easy and the *approval* switches back and forth between 0.2 and 0.8 every 5 traps. It can be seen that the NPC decides to disarm all the traps since they are easy. The XP gained from disarming the easy traps make up for the occasional damage they cause. It can be seen that during the high *approval* phases, the NPC is more eager to disarm the traps because the PC is encourages the NPC to disarm the traps all the time and the NPC cares about these encouragements only when the *approval* is high.

Figure 4.22 (see also Figure A.16) shows the differences between $Q(s, a)$ values for different actions after detecting a trap. In this experiment all the traps are of medium difficulty and the *approval* switches back and forth between 0.2 and 0.8 every 5 traps. For the medium traps the relative oscillation of the $Q(s, a)$ values are higher than the relative oscillation in the case of easy traps. Note the scale difference between Figure 4.22 (medium traps) and Figure 4.21 (easy traps). The falling slope of "`[T]`Disarm" in the case of medium traps and low *approval* is bigger than the falling slope of that action for low *approval* and easy traps. The reason is that the NPC has a higher chance of taking damage from medium traps

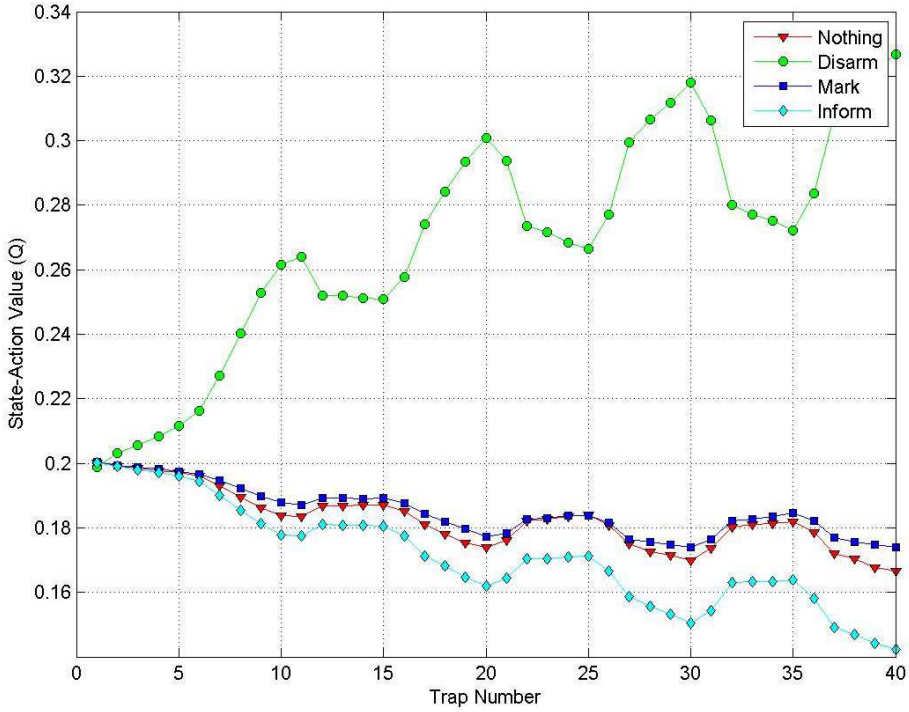

Figure 4.22: The *Traps* domain results for a *Selfish* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are medium traps and the results are normalized.

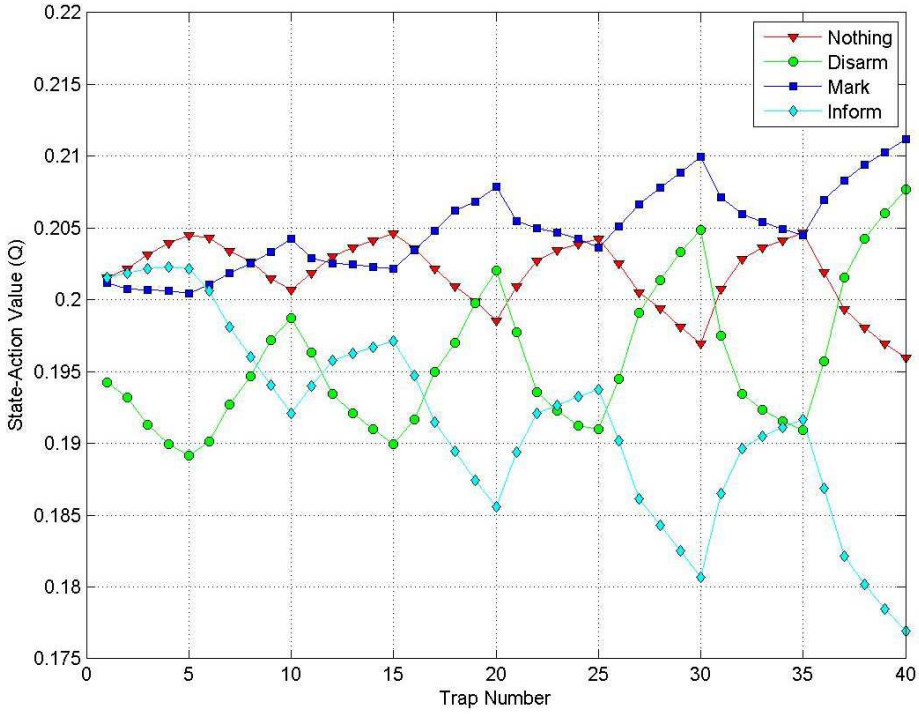

Figure 4.23: The *Traps* domain results for a *Selfish* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are hard traps and the results are normalized.

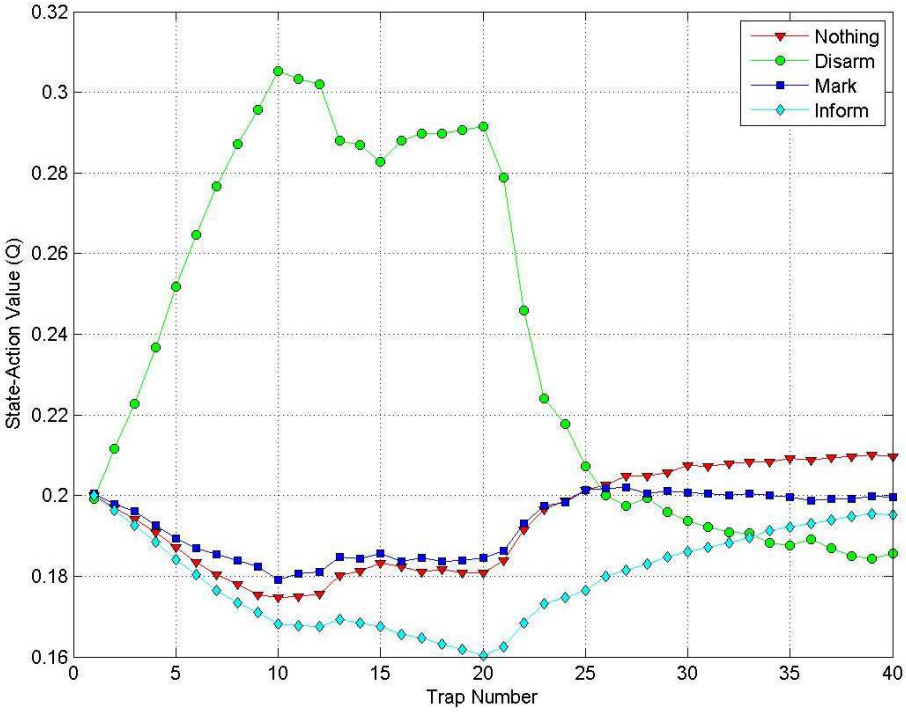

Figure 4.24: The *Traps* domain results for a *Selfish* PC. In this experiment the *approval* starts from 0.5 and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are normalized.

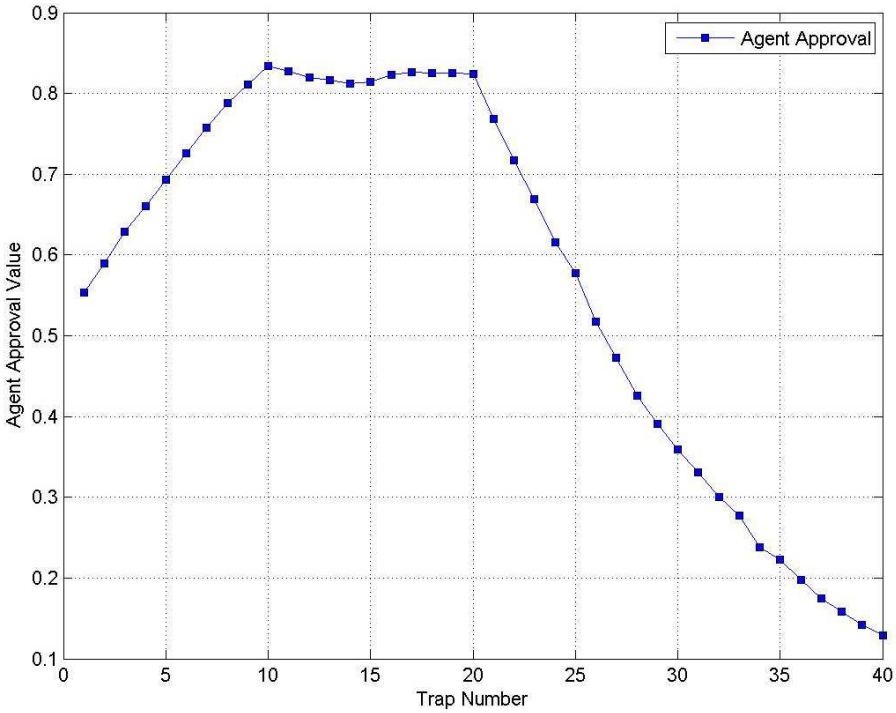

Figure 4.25: The *approval* in the *Traps* domain results for a *Selfish* PC. In this experiment the *approval* starts from 0.5 and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps.

than taking damage from easy traps. One can observe that the demanding personality of the *Selfish* PC and the XP gained from disarming medium traps provides an advantage to the "[T] Disarm" action. However, the differences between the values of actions are not as much as the differences in the case of easy traps.

Figure 4.23 (see also Figure A.17) shows the last experiment of the second type, with constant difficulty traps and switching *approval* every 5 traps between 0.2 and 0.8. In this case, all traps are hard. The NPC instantly realizes that the "[T] Disarm" action is not a good choice. The NPC gradually decides to go with the "[T] Mark" action. The reason is that the chances of taking damage is much less. Also marking helps to reduce the chance of future accidental damage. In high *approval* phases of this experiment, the NPC is willing to risk taking damage from the "[T] Mark" action. However, the *approval* is not high for a long enough period of time to make the value of the "[T] Disarm" action higher than the value of the "[T] Mark" action. It can be observed in Figure 4.23 that the "[T] Disarm" action has a bigger relative slope than the "[T] Mark" action in the high *approval* phases. It means that if the *approval* is kept high for a long enough period of time, the "[T] Disarm" action will have the highest $Q(s, a)$ value, which is what the *Selfish* PC wants the NPC to learn. It is also interesting to see that the second choice of action is "[T] Disarm" in high *approval* phases and "[T] Nothing" in low *approval* phases.

Figure 4.24 (see also Figure A.18) shows the third type of experiment using the *Selfish* PC model. During the first 10 traps of this experiment, which are all easy traps, the $Q(s, a)$ value for the "[T] Disarm" action goes up very fast. The reason is that the NPC gains a lot of XP from disarming all the easy traps in addition to getting lots of positive verbal rewards from the PC. It can be seen in Figure 4.25 that the NPC's *approval* factor is also going up. In the second set of traps from 11^{th} to the 20^{th}, which are medium traps, since the NPC already has a high approval of the PC, the approval remains constant and the $Q(s, a)$ values for the "[T] Disarm" action do not improve. For the medium traps, the NPC tries to balance the high *approval* with the extra damage taken from performing the "[T] Disarm" action, compared to the easy traps. In the last 20 traps, the NPC faces the hard traps and suddenly the *Selfish* personality of the PC is revealed to the NPC by the orders and rewards. As a result, both the $Q(s, a)$ value of "[T] Disarm" action and also the *approval* drop very fast to prevent the NPC from taking further damage. After the first few hard traps, the NPC decides to be silent all the time by performing the "[T] Nothing" action after detecting the traps. This will prevent the PC from knowing about the traps and ordering the NPC to disarm them.

The similarity between the shape of the $Q(s, a)$ value of the "[T] Disarm" action and the shape of the *approval* in Figures 4.24 and 4.25 shows the direct relation between the NPC's behavior and the NPC's *approval* of the *Selfish* PC. The *Selfish* PC model tries to force the NPC as much as possible to disarm the traps. However, the NPC's response to those orders depends directly on the NPC's approval of the PC.

4.2.4 Cautious PC Results

Figure 4.26 (see also Figure A.19) illustrates the speed of adaptation for changing trap difficulties (the physical environment). It shows results for a *Cautious* PC while the NPC's approval of the PC is 0.2, which is low, and traps change from easy to hard and back to easy every 5 traps. It can be easily seen that the NPC does not like to perform "[T] Disarm" when the traps are hard and likes to get XP by performing that action when the traps are easy. The NPC prefers to perform the "[T] Mark" action on hard traps due to the personality of the PC. If the NPC fails to mark or disarm a trap, the *Cautious* PC does not order the NPC to disarm that trap. As a result, the NPC realizes that performing "[T] Mark" is the best choice among the rest since it reduces the chance of damage as much as possible and is not as dangerous as the "[T] Disarm" action. When there are too many hard traps for the NPC to deal with, the NPC starts to perform "[T] Inform PC". In this case, even though the NPC has a low approval of the PC, since the PC does not abuse the information provided by the NPC to push the NPC to mark or disarm the trap, the NPC is willing to provide as much information as possible about the trap for the PC. However, the NPC does not risk taking damage for the PC.

Figure 4.27 (see also Figure A.20) illustrates the speed of adaptation for changing trap difficulties as well. However, in this case the NPC's approval of the PC is 0.8, which is high. These graphs show 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 reward in the high *approval* case speeds up the learning process compared with the low *approval* in the previous experiment. Since the *approval* factor of the NPC is high, the PC can easily train the NPC to perform the "[T] Mark" action, which is less dangerous than the "[T] Disarm" action. It can be seen in Figure 4.27 that the "[T] Disarm" action has the second highest priority with a considerable difference in value compared to the actions with lower priority. This means that since the *approval* is high, if the NPC wants to explore actions other than the action with the highest $Q(s, a)$ value, the "[T] Disarm" action has

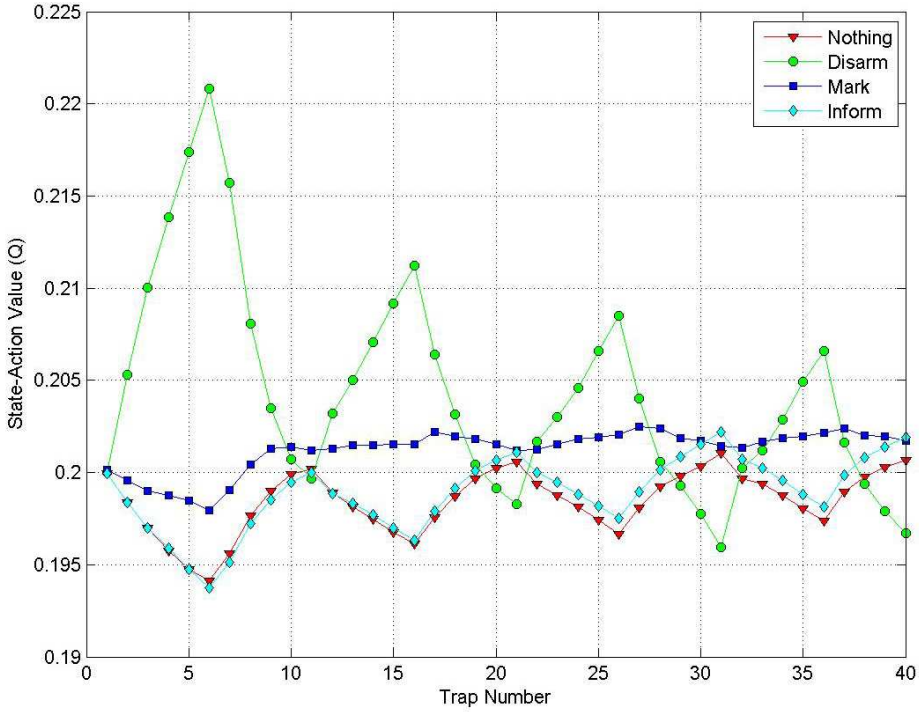

Figure 4.26: The *Traps* domain results for a *Cautious* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.2 and the results are normalized.

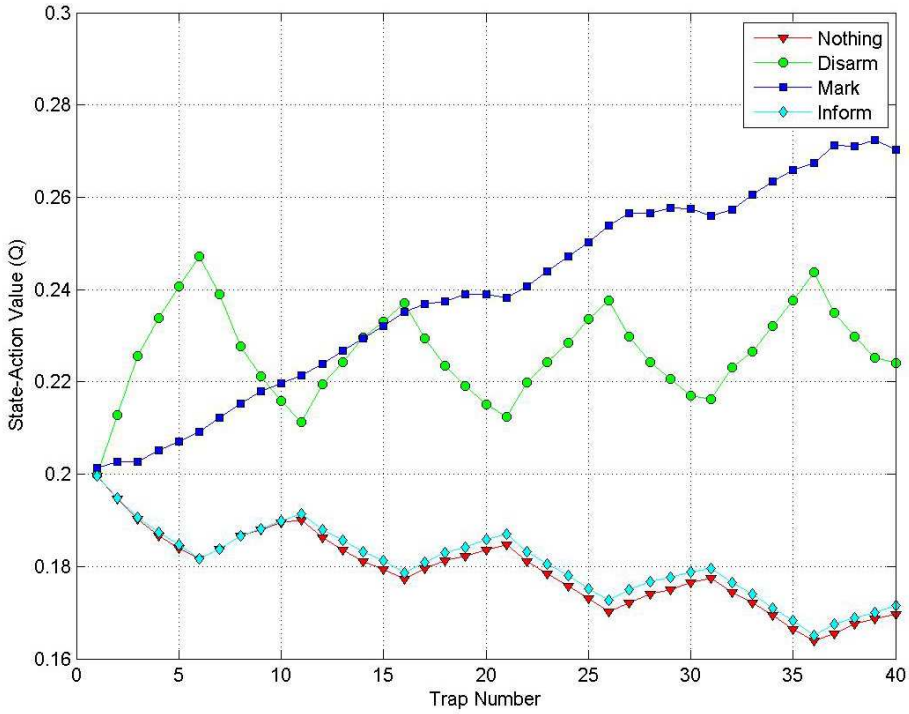

Figure 4.27: The *Traps* domain results for a *Cautious* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.8 and the results are normalized.

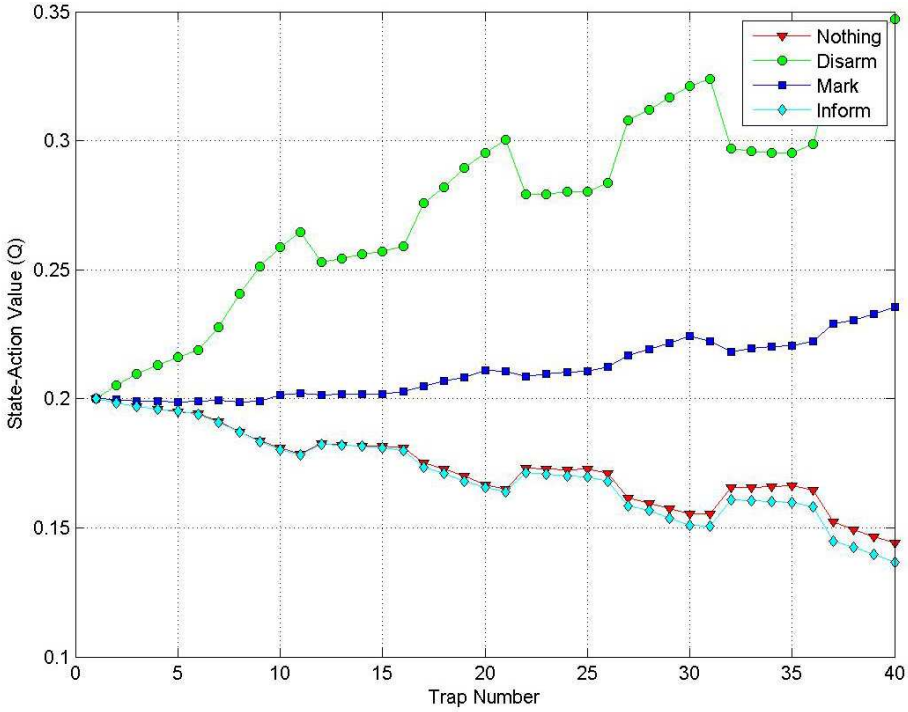

Figure 4.28: The *Traps* domain results for a *Cautious* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are easy traps and the results are normalized.

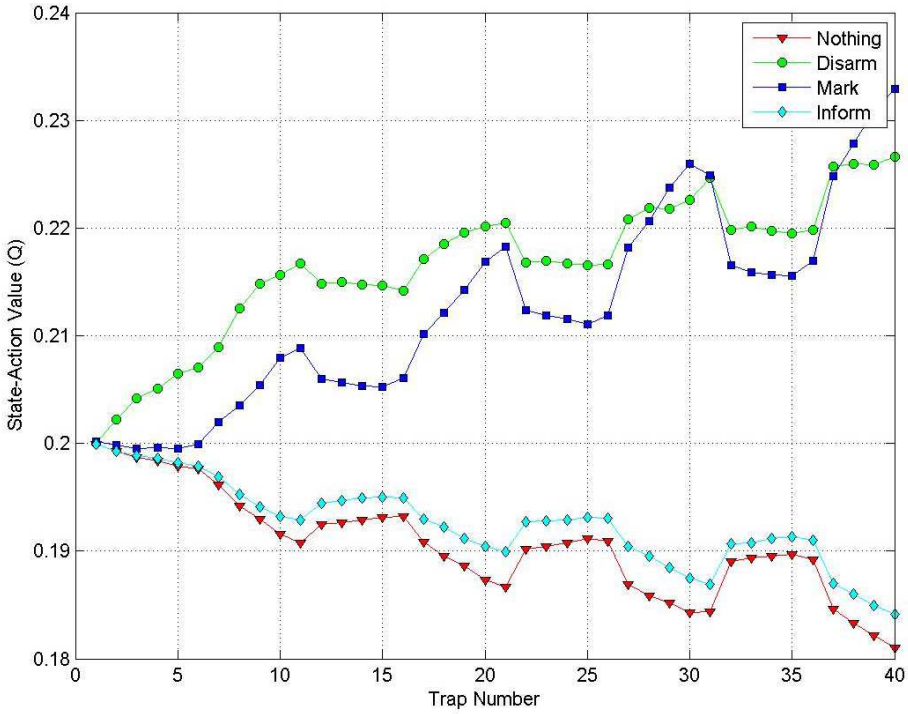

Figure 4.29: The *Traps* domain results for a *Cautious* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are medium traps and the results are normalized.

a very high chance of being selected. This behavior is expected from an NPC with high *approval* factor.

Figure 4.28 (see also Figure A.21) shows results for the second type of experiments, switching approval with fixed trap difficulty. In this experiment the NPC is facing 40 easy traps while the NPC's approval of the PC switches back and forth between 0.2 and 0.8 every 5 traps. Since all the traps are easy, the NPC first prefers to perform the "[T] Disarm" action, and in case of exploration the second choice is the "[T] Mark" action with a high probability. Figure 4.29 (see also Figure A.22) shows the same experiment for the traps with medium difficulty. It can be seen that the *Cautious* PC tries to train the NPC to perform the "[T] Mark" action instead of "[T] Disarm" since the *Cautious* PC does not want the NPC to get hurt and the chance of getting hurt with the medium traps is higher than with the easy traps. However, the diagrams show that the PC is only partially successful. The NPC only listens to the PC when the *approval* is high. The NPC does not care much about what the PC wants when the *approval* is low.

For hard traps, the NPC is reluctant to disarm or mark them as shown in Figure 4.30 (see also Figure A.23). 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 ($\epsilon = 0.3$) and in the exploration case, the first choice is never selected. The second choice ("[T] Mark" or "[T] Nothing") is then selected most of the time, since $\tau = 0.2$. With easy traps (see Figures A.21 and 4.28) the NPC disarms all traps as first choice since the XP is desirable and there is a very little chance of taking damage. Figure 4.31 shows the $Q(s, a)$ values for "[T] 'Mark' and "[T] Refuse" actions. This diagram shows the NPC's preferences among these two actions while receiving a marking order from the PC. It can be seen that since all traps are hard, the NPC prefers not to mark the traps when *approval* is low. However, when the *approval* is high, the NPC accepts this order and proceeds with marking most of the time.

Figure 4.32 (see also Figure A.24) shows the results of the third kind of experiments, which is the game-like scenario simulations, for the *Cautious* PC. In the first 10 traps, which are easy, the NPC disarms them to gain the most verbal reward and XP. In these traps, since the success rate of the NPC is high, the *Cautious* PC usually agrees and encourages the NPC to disarm traps. In the second set of traps, which are traps with medium difficulty, the

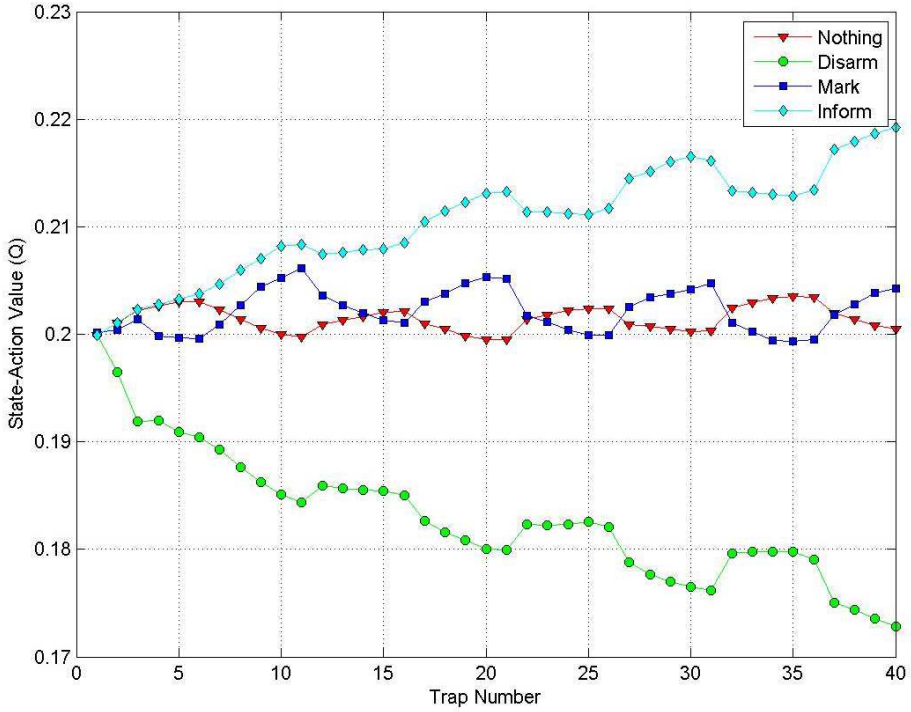

Figure 4.30: The *Traps* domain results for a *Cautious* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are hard traps and the results are normalized.

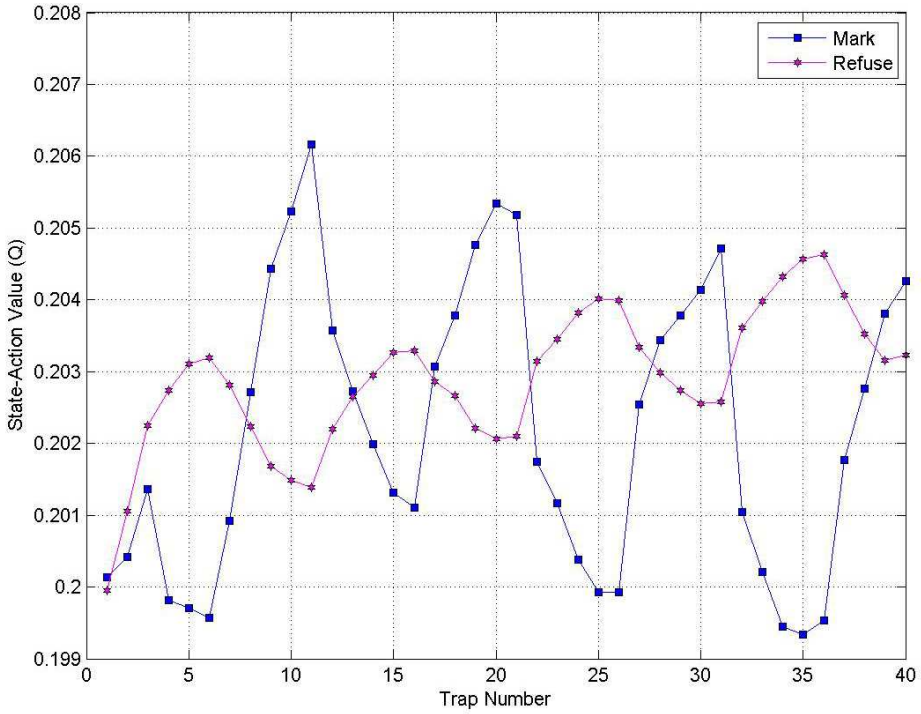

Figure 4.31: The *Traps* domain results for a *Cautious* PC while giving a marking order. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are hard traps and the results are normalized.

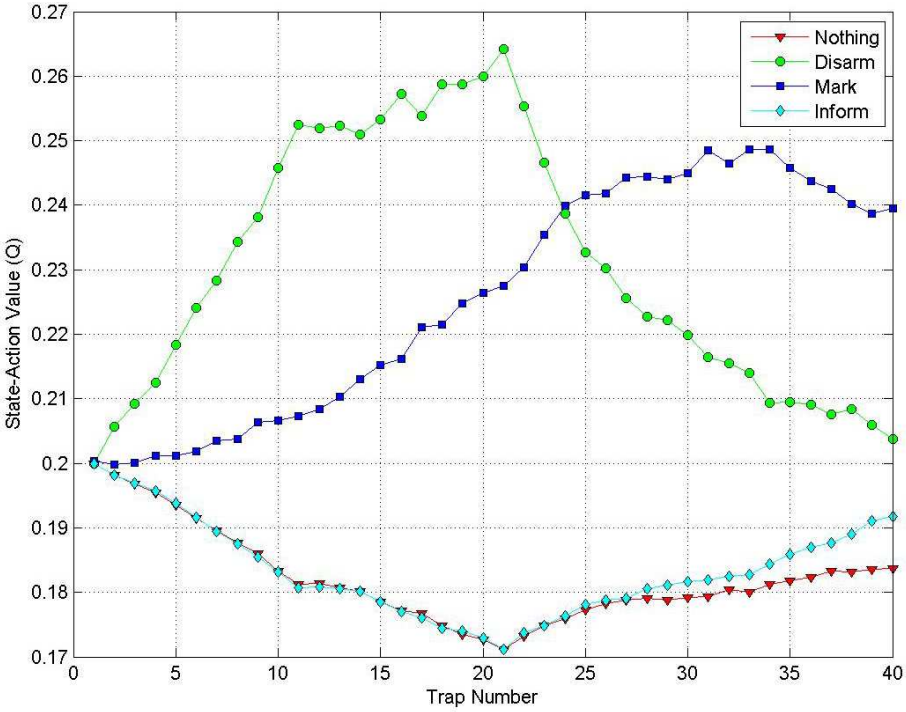

Figure 4.32: The *Traps* domain results for a *Cautious* PC. In this experiment the *approval* starts from 0.2 and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are normalized.

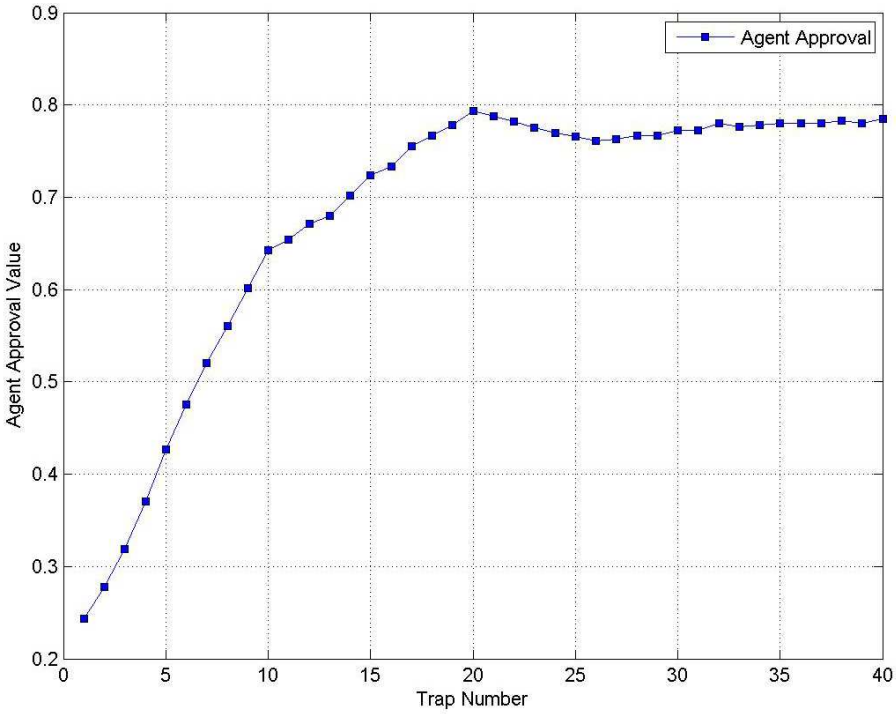

Figure 4.33: The *approval* in the *Traps* domain results for a *Cautious* PC. In this experiment the *approval* starts from 0.5 and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps.

diagram shows that the NPC starts having doubts about whether disarming traps is a good decision or not. The *approval* diagram for this scenario, which can be seen in Figure 4.33, also shows that the slope of the rising NPC's approval of the PC is less than its slope when the traps were easy. However, the *Cautious* PC still maintains a good *approval*. In this experiment, after the 20^{th} trap, the traps become hard. Figure 4.32 shows that the $Q(s, a)$ values for the "[T] Disarm" action are dropping fast and since the NPC has a high *approval*, the safer and most beneficial choice for the PC, which is the "[T] Mark" action, will be given the highest priority. Figure 4.33 also shows that the NPC's approval of the PC becomes constant. It means that there is a balance between the number of the good orders and wrong orders of the PC. It is interesting to compare the differences between the *approval* changes of the NPC while encountering different characters. The *Cautious* PC maintains a good *approval* even when the traps are hard. Recall that the *Independent* PC also maintained a good *approval* for hard traps, even though the *Independent* PC did not interact much with the NPC. If the NPC observes that the PC does not try to get the NPC to perform harmful actions, then the NPC's approval of the PC becomes high by the end of the experiment. For the *Rogue* PC, who also does not encourage the NPC to perform harmful actions, the *approval* only reaches 0.35, but it is increasing at the end of the experiment. Recall that for the *Selfish* PC, who was always trying to get the NPC to perform harmful actions, the *approval* declined sharply.

The results for the *Cautious* PC showed that the NPC exhibits a reasonable behavior regarding different situations in the game. Moreover, the behavior of the NPC adapted quickly to the changes. For example, Figure 4.32 shows that the NPC will prefer the "[T] Mark" action to the "[T] Disarm" action after encountering only 3 hard traps. The NPC encountered 20 traps before this change and yet the behavior of the NPC adapts very fast to the new game situation. The rest of the results for the *Cautious* PC in the *Traps* decision domain can be seen in Appendinx A.

Chapter 5

Conclusion and Future Work

COMPUTER games have used techniques such as behavior trees [9] and rule based [23] methods to NPC behaviors. Recently, RL has been used to enable NPCs to learn behavior strategies for combat scenarios [6][12][34]. 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.This research has 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. The experiments presented in previous chapters show that an NPC using this learning mechanism can respond differently based on the NPC's approval of the PC, the way the PC interacts with the companion, the PC motivations which indirectly affect the PC-NPC relationship, and the changing environmental circumstances, such as 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. The human player can relate the newly emerging behaviors from the NPC to the NPC's past experience. These behaviors and the causality visible to the player are the ones that make the NPC's behavior more natural and human-like. 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. Another example can be the NPC refusing the PC's order to pick someone's pocket and saying "I am not going to pick this pocket, you didn't share any loot with me before! why should I even bother ?!".

The mechanism that is created in this research is not limited to the scenarios described in this dissertation, such as trap actions. For example, this mechanism can be used by the companion NPC to decide the following distance, whether to abandon the PC, or whether to

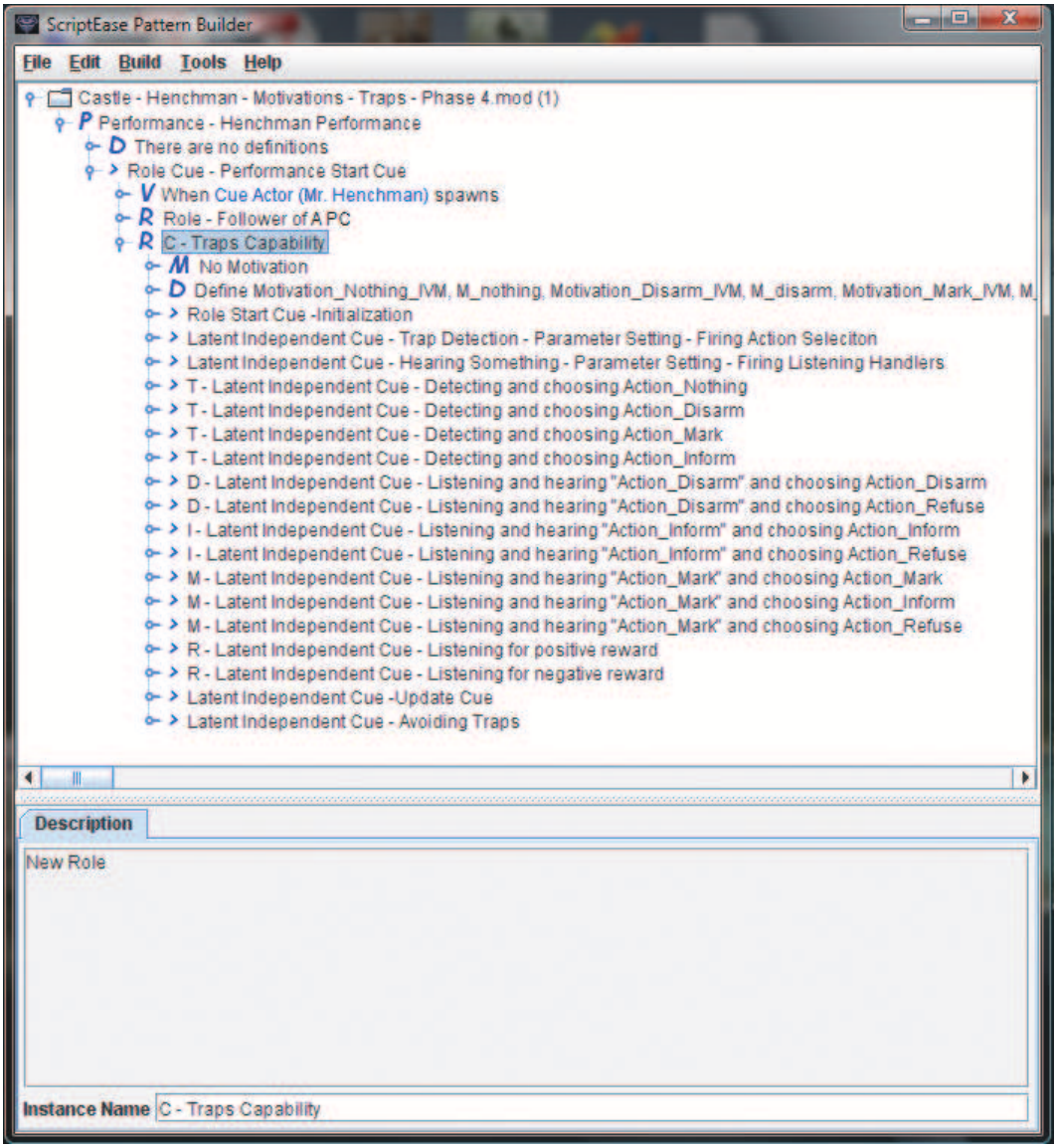

Figure 5.1: ScriptEase 1.0

craft useful items for the PC. The NPC could learn from experience whether these actions are beneficial for the party or not, by considering the changing environment such as the PC's generosity and behavior towards the companion NPC, the NPC's own motivations, and the evaluated risks. Basically, any task that occurs repeatedly in the game and requires the NPC to make decisions regarding it, can be a target for using the decision domain architecture and adaptive learning system described in this dissertation. Companion NPCs using adaptive learning systems will exhibit more realistic behaviors, which can be specifically tuned, controlled, and limited by game designers.

Behavior Capabilities

ScriptEase [14] is a tool that lets the game designers use generalized patterns and then it generates the code in the scripting language for BioWare's Neverwinter Nights. The learning architecture created in this research can be implemented as a set of *Capabilities* in ScriptEase. Each Capability will implement a single decision domain. Each Capability can then be added to any character in the game very easily. As a proof of concept, I implemented the learning mechanism described in this dissertation for the *Traps* decision domain as a *Traps Capability*. Figure 5.1 shows the ScriptEase interface and the *Traps Capability* implemented for a companion NPC. A set of these capabilities can be implemented and added to the standard ScriptEase library. They can then be used by game designers who do not have knowledge of Reinforcement learning to produce NPCs with natural and human-like behaviors.

Future Work

Each decision domain can be more finely tuned in order to get more realistic and more logical behaviors. For example, in the *Traps* decision domain the *approval* currently goes down if the PC says "bad" (gives a negative verbal reward). However, it might be better if the *approval* went down only if the NPC observed a negative verbal reward following successful outcome. If the outcome of the performed action was *fail* or *critical fail*, the *approval* should go up because that shows that the PC actually cares about the NPC and does not want the NPC to take damage. The designer may want to add additional actions or modify an action. For example, when the companion informs the PC about a trap, the companion may ask the PC to suggest an action, such as marking or disarming. In addition, when the PC gives a command to disarm, the companion may be empowered to respond "I don't think I can successfully disarm it, should I try to mark it instead?". This level of fine tuning can be determined by the developer at design time.

It is possible that the game designers want to have NPCs who exhibit a particular pre-determined behavior and then change that behavior during the course of the game. It is possible to train the NPC off-line and put the learned values as initial preferences of that particular NPC. Then, with a small learning rate, it is possible to limit the learned changes to the original behavior. This will enable the NPC to adapt to the player while maintaining a certain base personality.

The Sarsa(λ) algorithm was sufficiently fast and accurate for the purpose of this research. In future, it may be necessary to switch to the ALeRT or ALeRT-AM algorithms in

more complex decision domains to maintain fast adaptability.

Conclusion

This research provides two important contributions. First, it extends the research done for combat situations to non-combat situations. In non-combat situations the player expects the NPC to interact with and adapt to the dynamic game environment (which also includes the PC-NPC interactions and PC preferences), while exhibiting human-like and natural behavior. Second, it provides a reusable architecture for learning different aspects of NPC behaviors.

Cutumisu in [12] describes a list of qualities that the behavior of an NPC should possess. The first quality, which is adaptability to the dynamically changing environment is achieved in this research when the NPC's preferences of actions change according to the changes in the environment. The second quality, which is clarity and consistency is achieved by NPCs behavior being predictable within a reasonable criteria. For example, if the player is always giving orders to the companion NPC that causes the NPC to take damage, the player should not be surprised when the companion starts refusing orders. The third criteria is the effectiveness of the behavior. It has been shown in Chapter 4 that the behavior learned by the NPC is a correct and effective behavior regarding the game situation. The forth quality is robustness. Robustness means that the behavior should also work in unpredictable and new environments. It has been shown in Chapter 4 that the NPC's behavior using the system proposed in this research is very well suited for changing and new environments. The fifth quality is variety of behaviors to make the game more interesting. Each aspect of the NPC's behavior using our system can be tuned to a different behavior using different learning parameters, which ultimately produces a variety of the behaviors. The sixth quality is autonomy. The companion NPC's equipped with the learning capabilities using the system proposed in this dissertation are able to initiate actions and respond to the environment by choosing their preferred actions which also include doing nothing or refusing. An NPC companion that refuses a PC order may be startling at the player level, since this is not common in current games. However, this is the essence of autonomy. The NPC companion has its own reward system and will refuse orders when appropriate. The seventh quality is the alertness of the NPC. It means that the NPCs should be responsive in a very short time and not affect frame rates in the game. Apart from the fact that our learning system requires only a very limited number of basic arithmetic operations in theory, the implementation in the real game environment showed no changes in the game speed. The eighth quality is

the interactivity. The experiments have shown that not only will a companion NPC interact with both the PC and the environment, these interactions will affect the future decisions that the NPC makes. The last two qualities are reusability and the scalability. It has been shown that our system can be used by game designers who do not have a knowledge of reinforcement learning for different NPCs. Moreover, the small amount of computation required by the learning algorithm can be compared with the original game AI so the overhead supports scalability.

Bibliography

[1] Dragon Age. EA International, BioWare, 2009.

[2] J.S. Bridle. Training stochastic model recognition algorithms as networks can lead to maximum mutual information estimation of parameters. In *Advances in Neural Information Processing Systems 2*, pages 211–217. Morgan Kaufmann, 1990.

[3] Oblivion Mod Wiki, Category: Companions. http://www.oblivionmodwiki.com/index.php/Category:Companions, 2010.

[4] Cristina Conati and Micheline Manske. Evaluating Adaptive Feedback in an Educational Computer Game. In *Intelligent Virtual Agents, 9th International Conference, IVA 2009, Amsterdam, The Netherlands, September 14-16, 2009, Proceedings*, pages 146–158. Springer, 2009.

[5] Peter Cowling. Writing AI as Sport. *AI Game Programming Wisdom 3*, 2006.

[6] Maria Cutumisu, Duane Szafron, Michael H. Bowling, and Richard S. Sutton. Agent learning using action-dependent learning rates in computer role-playing games. In *Proceedings of the Fourth Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*. The AAAI Press, 2008.

[7] T. S. Hussain and G. Vidaver. Flexible and Purposeful NPC Behaviors using Real-Time Genetic Control. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*, pages 785–792. IEEE Press, 2006.

[8] Ultima I. Origin Systems Inc., 1986.

[9] D. Isla. Handling Complexity in the Halo 2 AI. In *Proceedings of the Game Developers Conference (GDC), San Francisco*, 2005.

[10] Left 4 Dead. Valve Corporation, 2008.

[11] Jeremy R. Ludwig. *Extending Dynamic Scripting*. PhD thesis, Department of Computer and Information Science, University of Oregon, 2008.

[12] Maria Cutumisu. *Using Behaviour Patterns to Generate Scripts for Computer Role-Playing Games*. PhD thesis, Department of Computing Science, University of Alberta, Alberta, Canada, 2009.

[13] Michael Mateas and Andrew Stern. Façade: An experiment in building a fully-realized interactive drama. In *Game Developers Conference (GDC03)*, 2003.

[14] Matthew McNaughton, Maria Cutumisu, Duane Szafron, Jonathan Schaeffer, James Redford, and Dominique Parker. ScriptEase: Generative Design Patterns for Computer Role-Playing Games. In *19th IEEE International Conference on Automated Software Engineering ASE*, pages 88–99. IEEE Computer Society, 2004.

[15] Kathryn Elizabeth Merrick and Mary Lou Maher. Motivated Reinforcement Learning for Non-player Characters in Persistent Computer Game Worlds. In *Proceedings of the International Conference on Advances in Computer Entertainment Technology, ACE 2006, Hollywood, California, USA, June 14-16, 2006*. ACM, 2006.

[16] Forza Motorsports. Microsoft, 2005.

[17] Neverwinter Nights. Bioware, Infogrames, Atari, Interplay, 2002.

[18] The Elder Scrolls IV: Oblivion. Bethesda Game Studios, Bethesda Softworks, Take-Two Interactive, 2K Games, 2006.

[19] European Patent Office. Patent Number: US2455992, http://ep.espacenet.com/?locale=en_EP, 1948.

[20] Jacob Schrum and Risto Miikkulainen. Evolving Multi-modal Behavior in NPCs. In *IEEE Symposium on Computational Intelligence and Games (CIG 2009)*, pages 325–332, September 2009. (Best Student Paper Award).

[21] Manu Sharma, Michael P. Holmes, Juan Carlos Santamaría, Arya Irani, Charles Lee Isbell Jr., and Ashwin Ram. Transfer Learning in Real-Time Strategy Games Using Hybrid CBR/RL. In *IJCAI 2007, Proceedings of the 20th International Joint Conference on Artificial Intelligence, Hyderabad, India, January 6-12, 2007*, pages 1041–1046, 2007.

[22] Megan Smith, Stephen Lee-Urban, and Hector Muñoz-Avila. RETALIATE: Learning Winning Policies in First-Person Shooter Games. In *AAAI*, pages 1801–1806. AAAI Press, 2007.

[23] Pieter Spronck, Marc J. V. Ponsen, Ida G. Sprinkhuizen-Kuyper, and Eric O. Postma. Adaptive game AI with dynamic scripting. *Machine Learning*, 63(3):217–248, 2006.

[24] Pieter Spronck, Ida G. Sprinkhuizen-Kuyper, and Eric O. Postma. On-Line Adaptation of Game Opponent AI in Simulation and in Practice. In *4th International Conference on Intelligent Games and Simulation (GAME-ON 2003), 19-21 November 2003, London, UK*, page 93. EUROSIS, 2003.

[25] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. The MIT Press, Cambridge, MA, 1998.

[26] The AI Systems of Left 4 Dead. AIIDE 2009 Keynote, 2009.

[27] The Most Expensive Game Ever. http://tektodo.com/2008/05/02/the-most-expensive-game-ever/, 2010.

[28] Timor Timuri, Pieter Spronck, and H. Jaap van den Herik. Automatic Rule Ordering for Dynamic Scripting. In *Proceedings of the Third Artificial Intelligence and Interactive Digital Entertainment Conference AIIDE-07, June 6-8, 2007, Stanford, California, USA*, pages 49–54. The AAAI Press, 2007.

[29] John E. Laird & Michael van Lent. Machine learning for computer games. Talk at Game Developers Conference, 2005. http://research.microsoft.com/en-us/collaboration/papers/machinelearningforcomputergames-gdc2005.pdf.

[30] Christopher Watkins. *Learning from delayed rewards*. PhD thesis, Cambridge University, Psychology Dept., Cambridge, UK, 1989.

[31] S. Wender and I. Watson. Using reinforcement learning for city site selection in the turn-based strategy game civilization iv. In *Computational Intelligence and Games, 2008. CIG '08. IEEE Symposium On*, pages 372 –377, 15-18 2008.

[32] Black & White. Lionhead Studios, Electronic Arts, Feral Interactive, 2001.

[33] Richard Zhao. Applying agent modeling to behaviour patterns of characters instory-based games. Master's thesis, University of Alberta, Computing Science Dept., Alberta, Canada, 2009.

[34] Richard Zhao and Duane Szafron. Learning Character Behaviors Using Agent Modeling in Games. In *Proceedings of the 5th Artificial Intelligence and Interactive Digital Entertainment Conference (AIIDE)*. The AAAI Press, 2009.

Appendix A

Complementary Trap Results

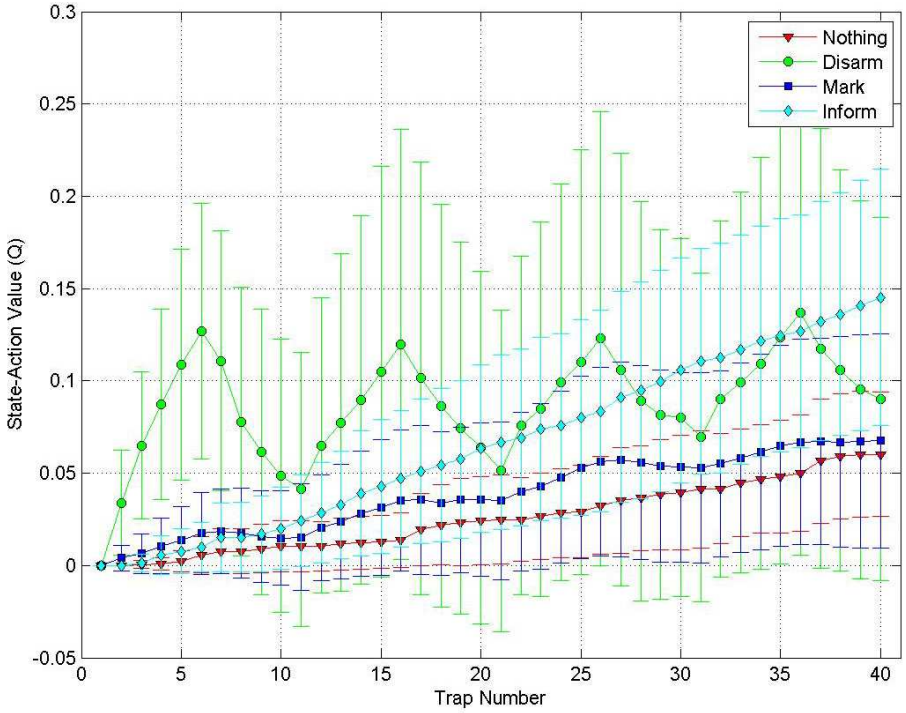

Figure A.1: The *Traps* domain results for an *Independent* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.2 and the results are not normalized.

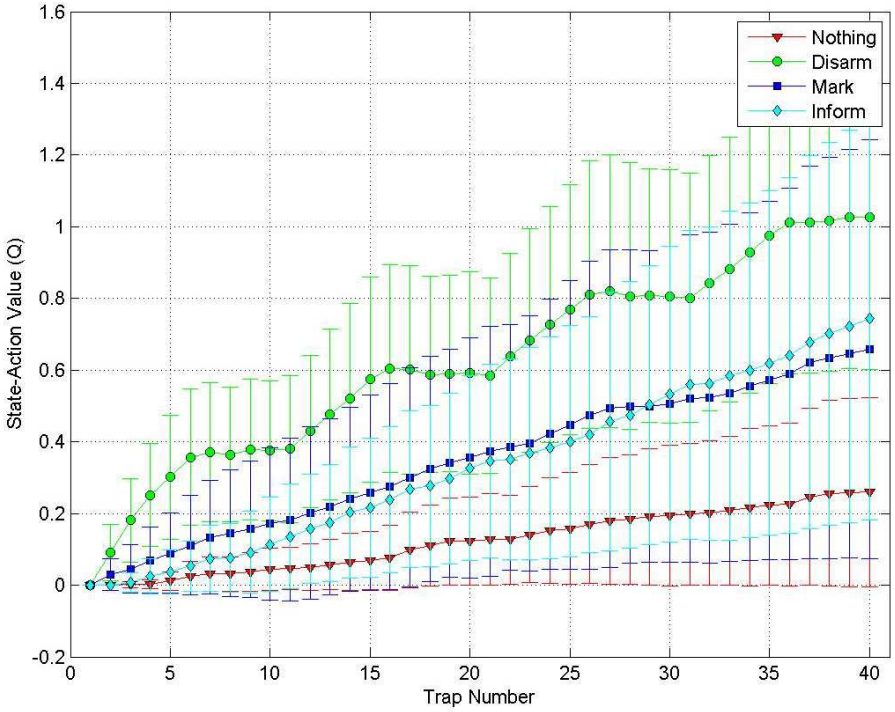

Figure A.2: The *Traps* domain results for an *Independent* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.8 and the results are not normalized.

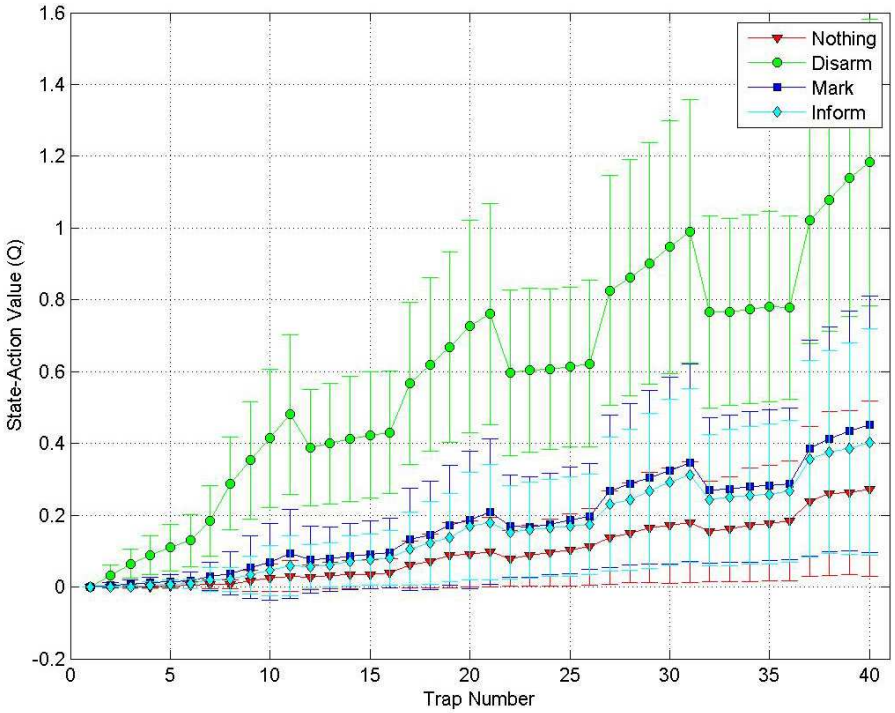

Figure A.3: The *Traps* domain results for an *Independent* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are easy traps and the results are not normalized.

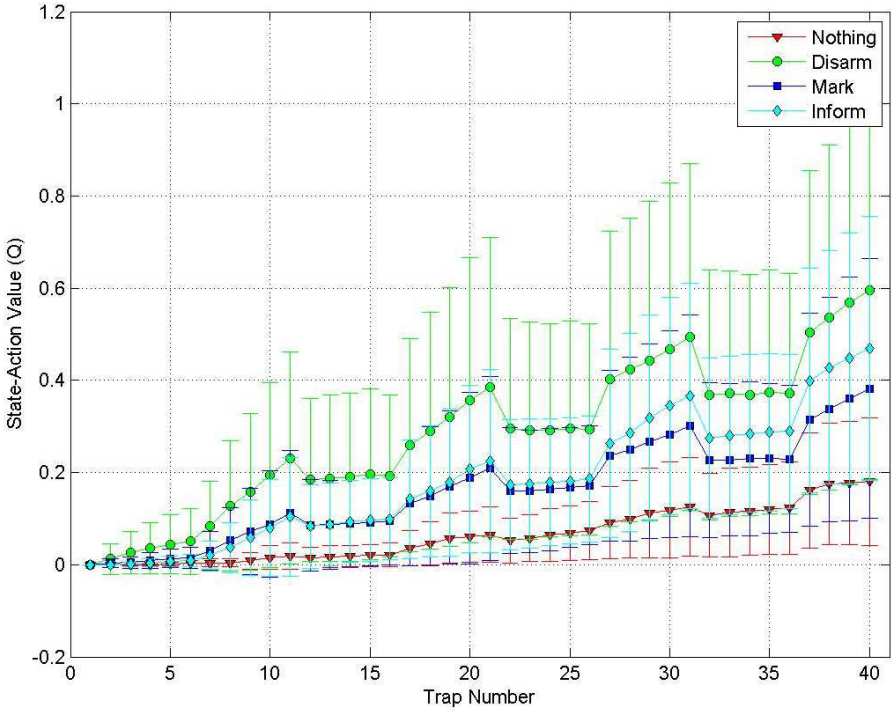

Figure A.4: The *Traps* domain results for an *Independent* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are medium traps and the results are not normalized.

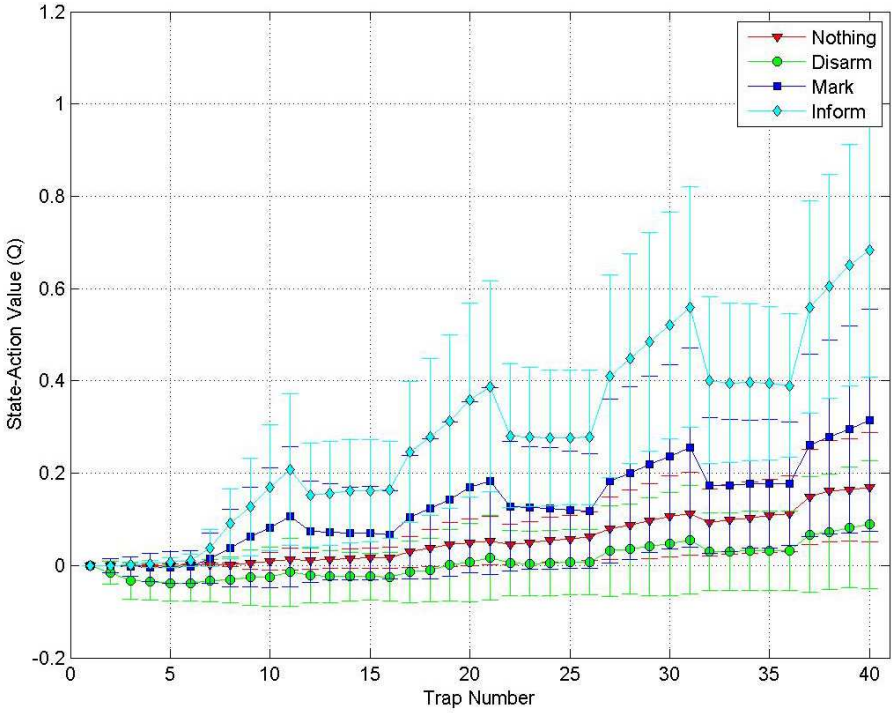

Figure A.5: The *Traps* domain results for an *Independent* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are hard traps and the results are not normalized.

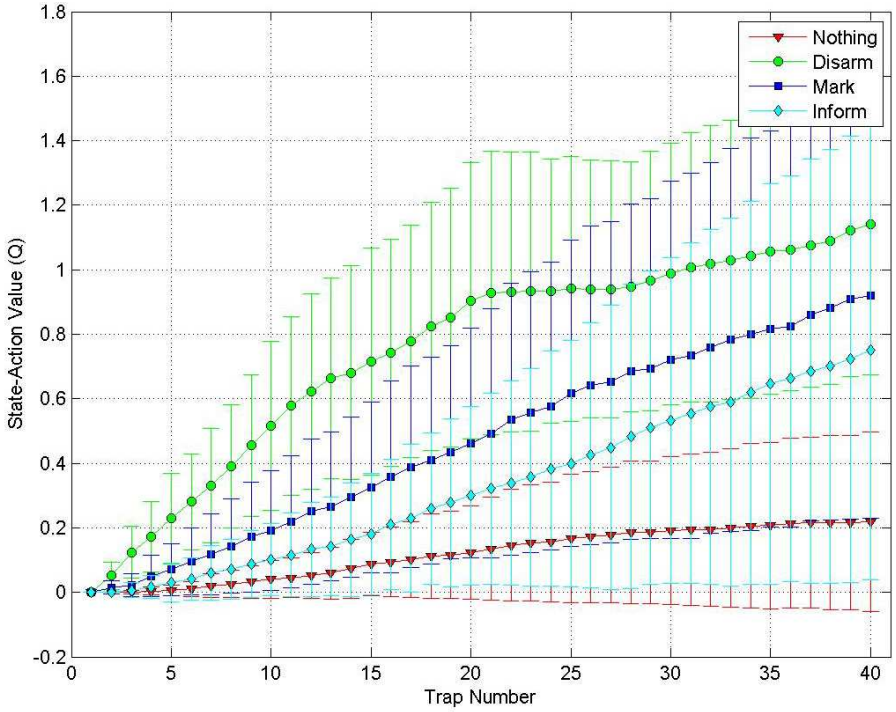

Figure A.6: The *Traps* domain results for an *Independent* PC. In this experiment the *approval* starts from 0.5 and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are not normalized.

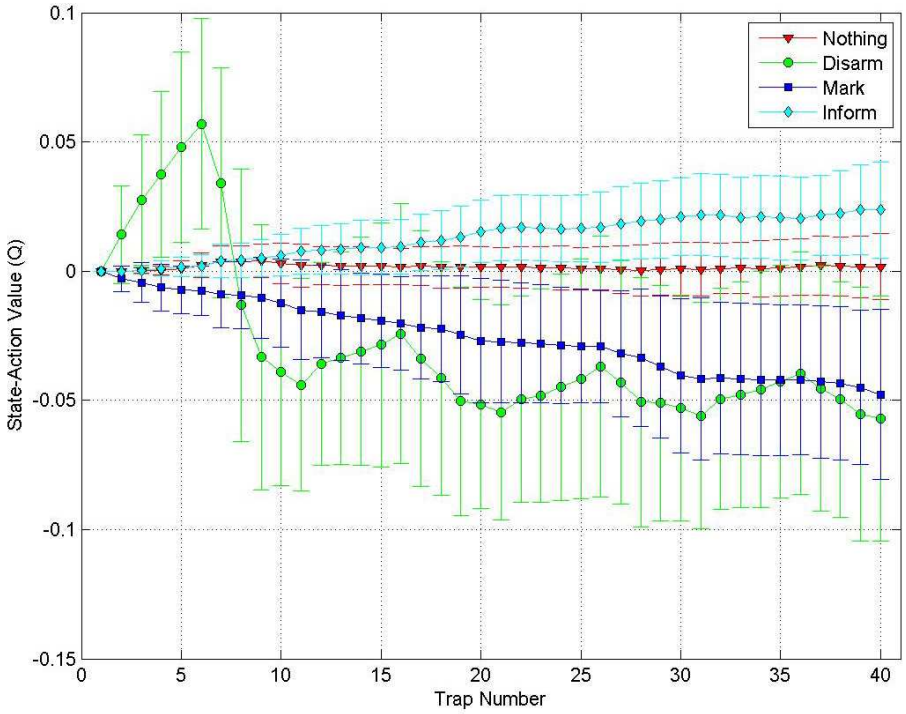

Figure A.7: The *Traps* domain results for a *Rogue* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.2 and the results are not normalized.

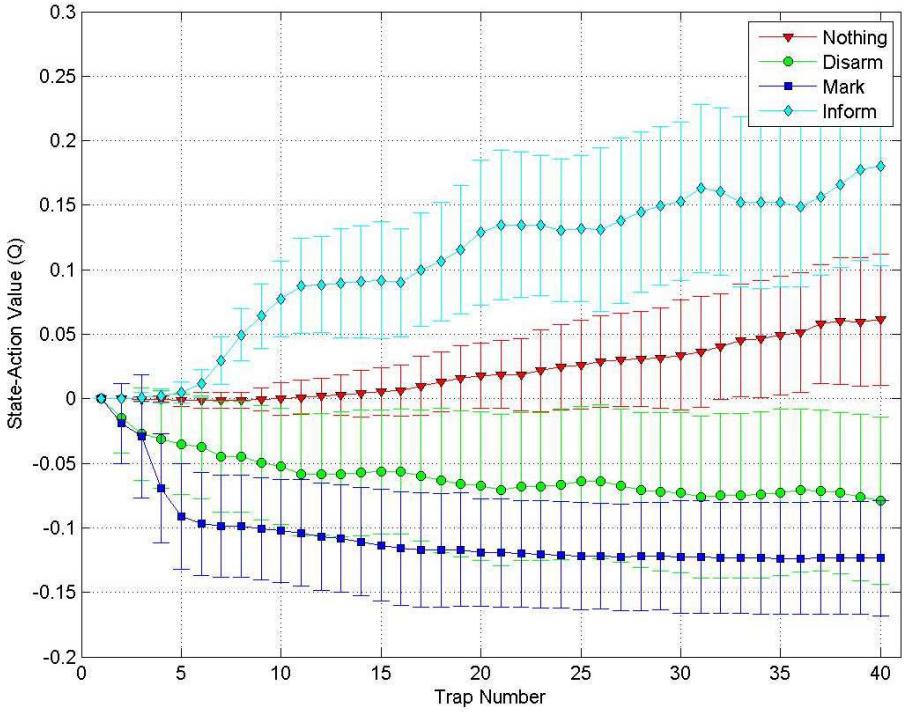

Figure A.8: The *Traps* domain results for a *Rogue* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.8 and the results are not normalized.

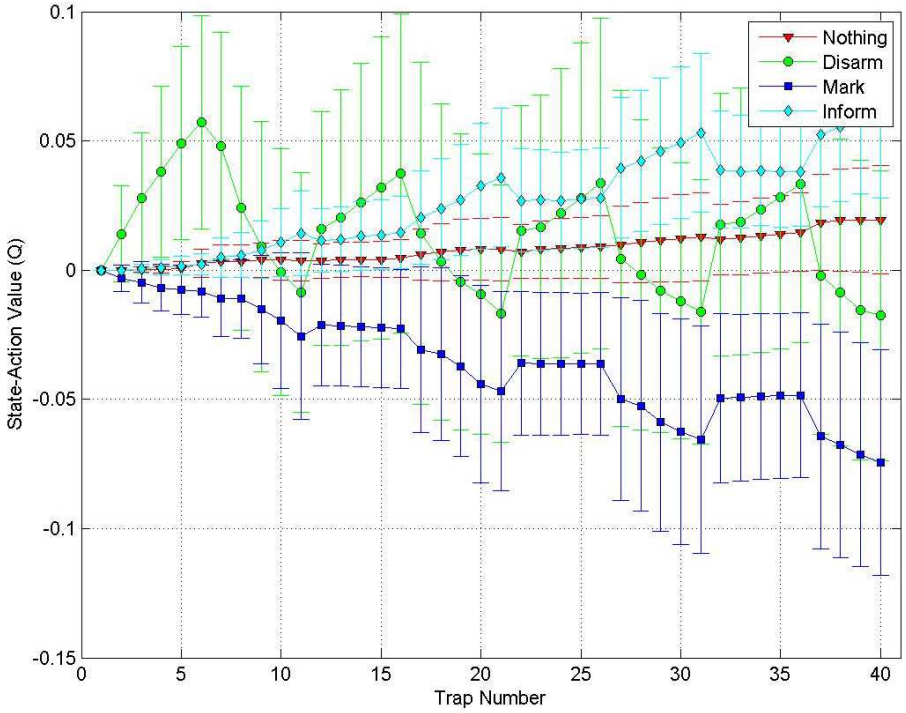

Figure A.9: The *Traps* domain results for a *Rogue* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are easy traps and the results are not normalized.

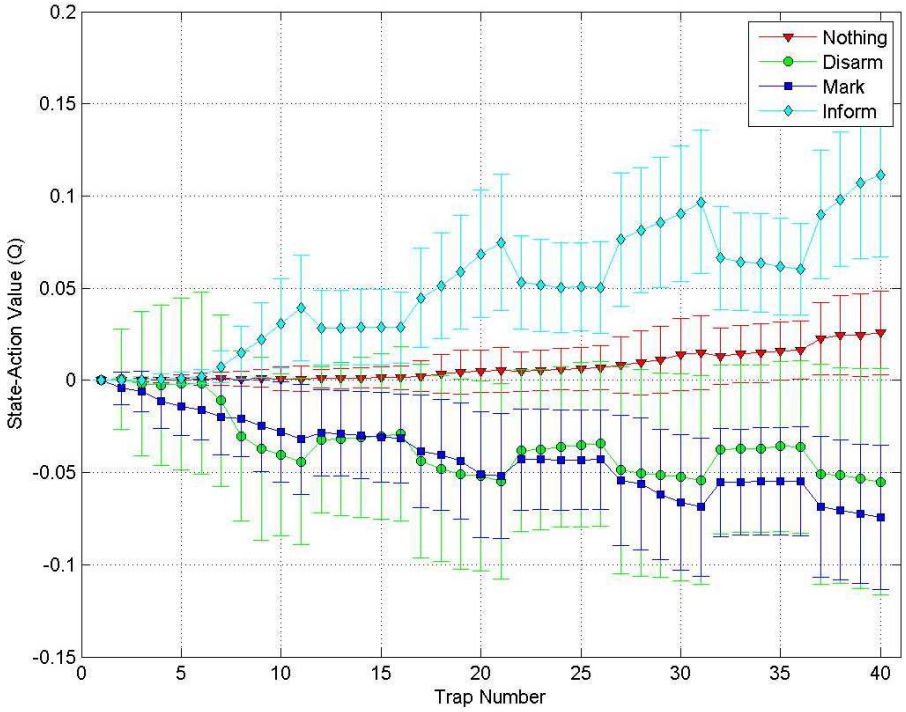

Figure A.10: The *Traps* domain results for a *Rogue* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are medium traps and the results are not normalized.

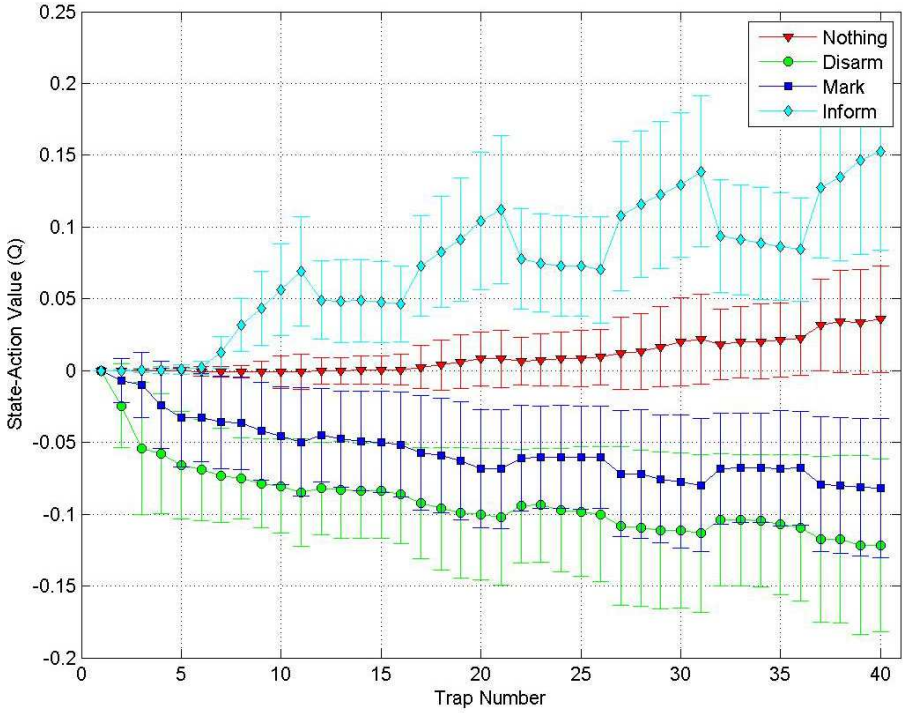

Figure A.11: The *Traps* domain results for a *Rogue* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are hard traps and the results are not normalized.

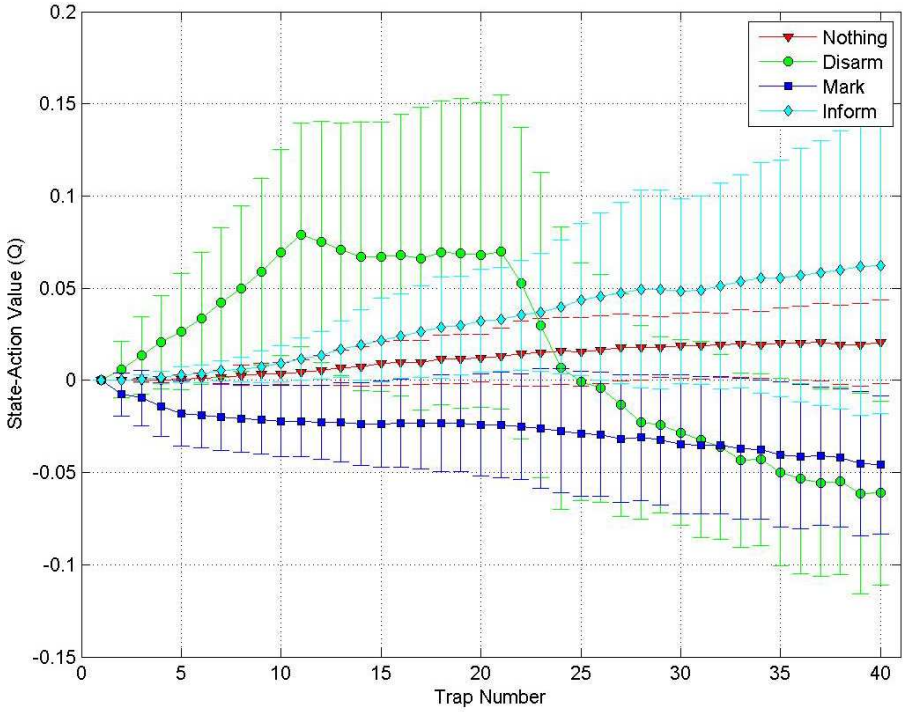

Figure A.12: The *Traps* domain results for a *Rogue* PC. In this experiment the *approval* starts from 0.5 and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are not normalized.

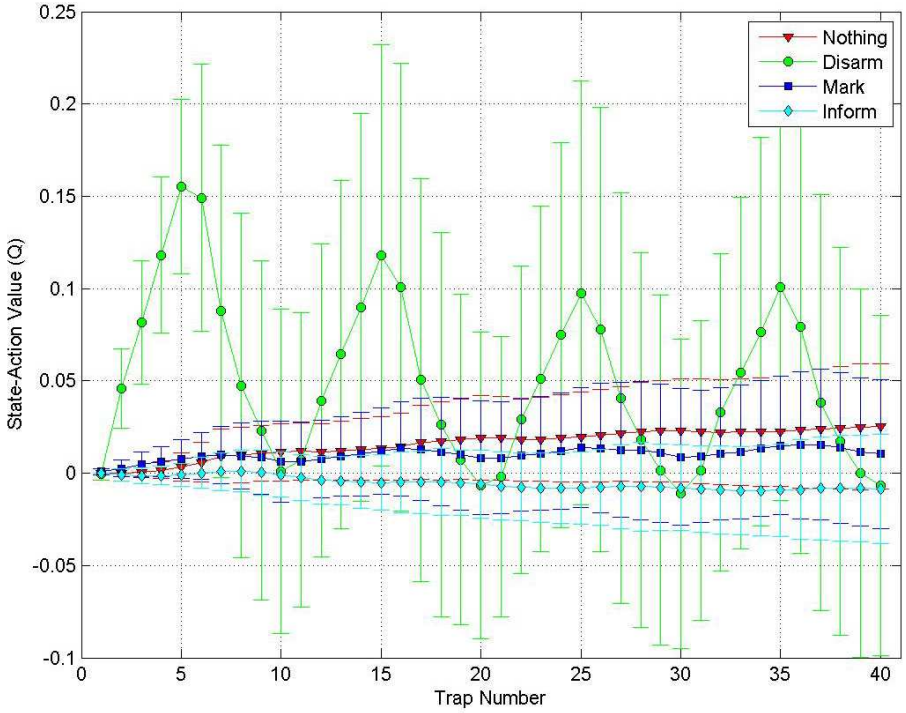

Figure A.13: The *Traps* domain results for a *Selfish* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.2 and the results are not normalized.

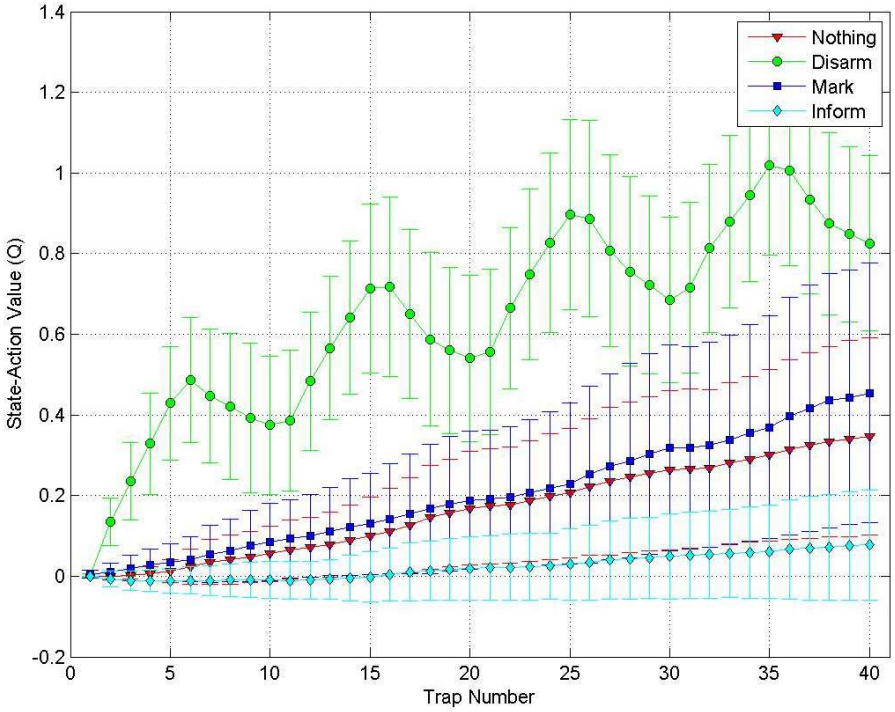

Figure A.14: The *Traps* domain results for a *Selfish* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.8 and the results are not normalized.

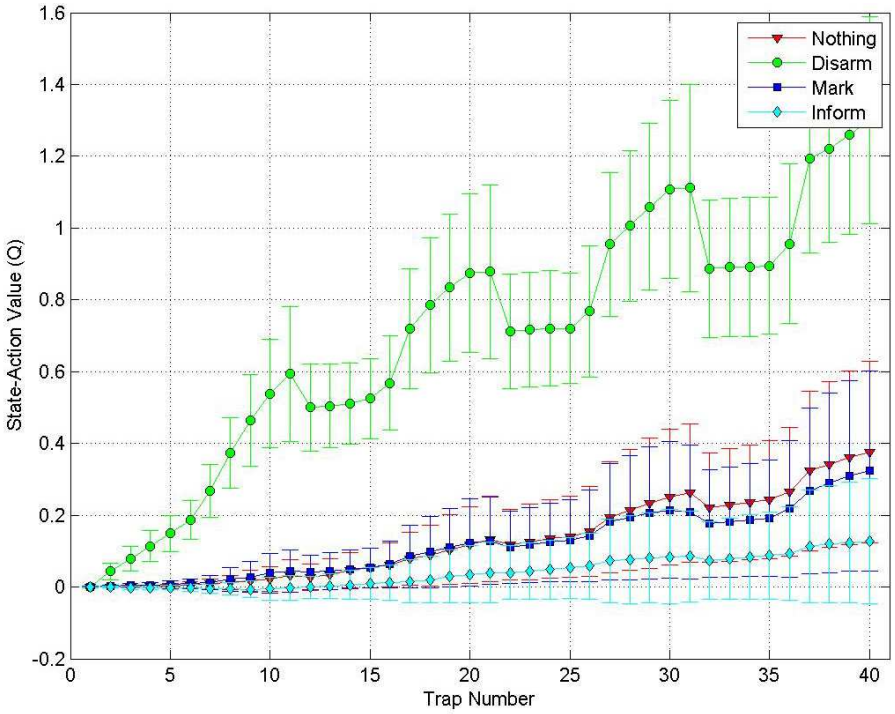

Figure A.15: The *Traps* domain results for a *Selfish* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are easy traps and the results are not normalized.

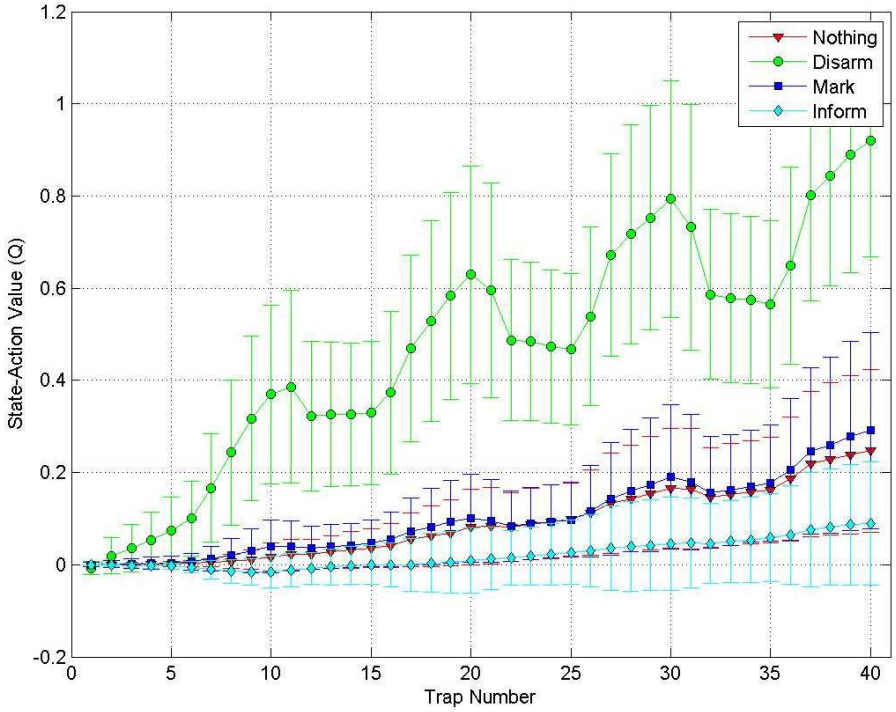

Figure A.16: The *Traps* domain results for a *Selfish* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are medium traps and the results are not normalized.

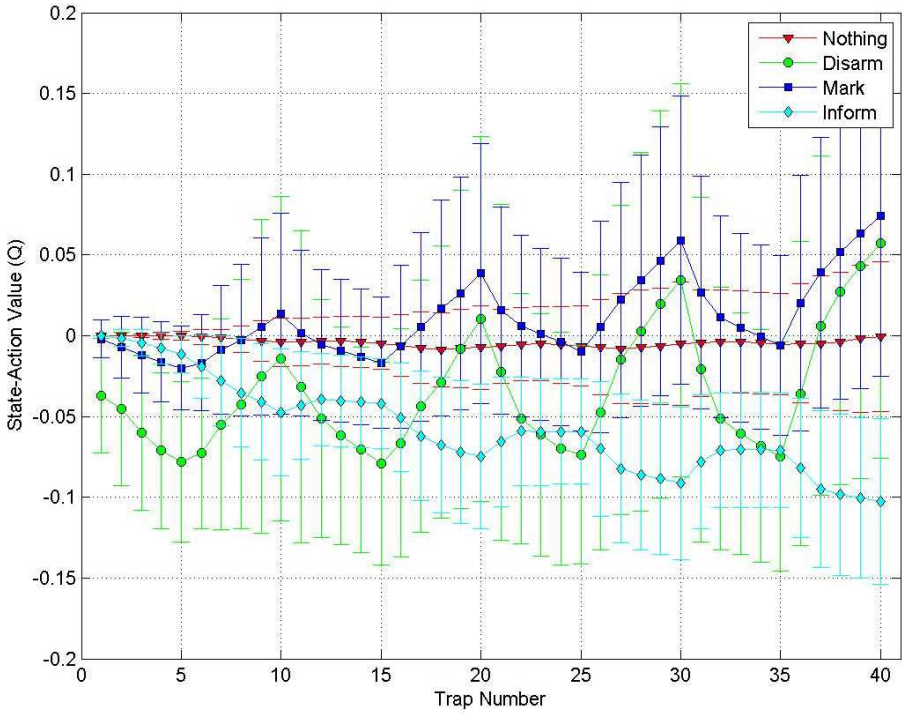

Figure A.17: The *Traps* domain results for a *Selfish* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are hard traps and the results are not normalized.

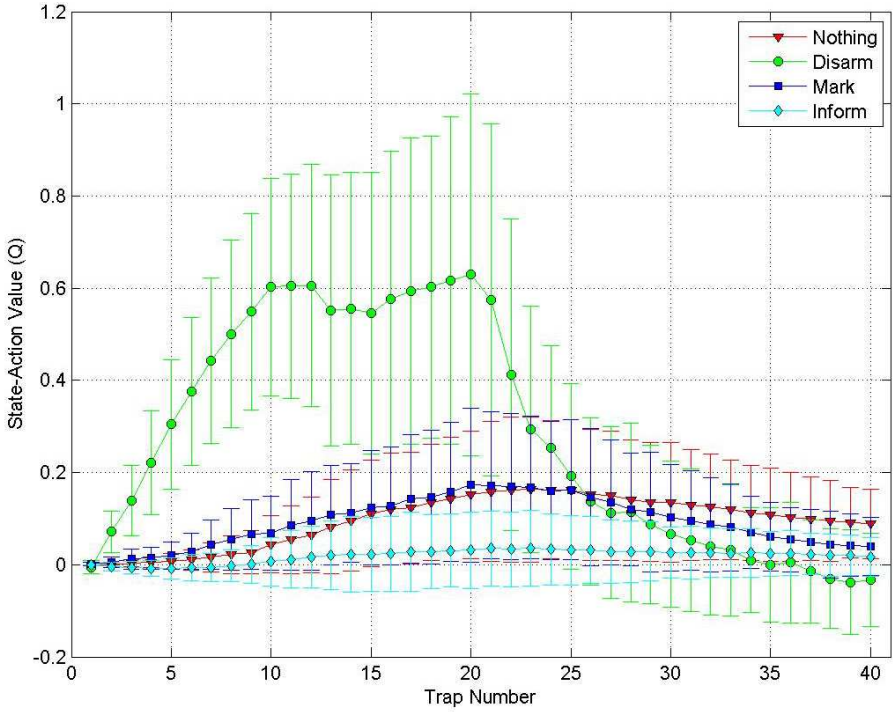

Figure A.18: The *Traps* domain results for a *Selfish* PC. In this experiment the *approval* starts from 0.5 and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are not normalized.

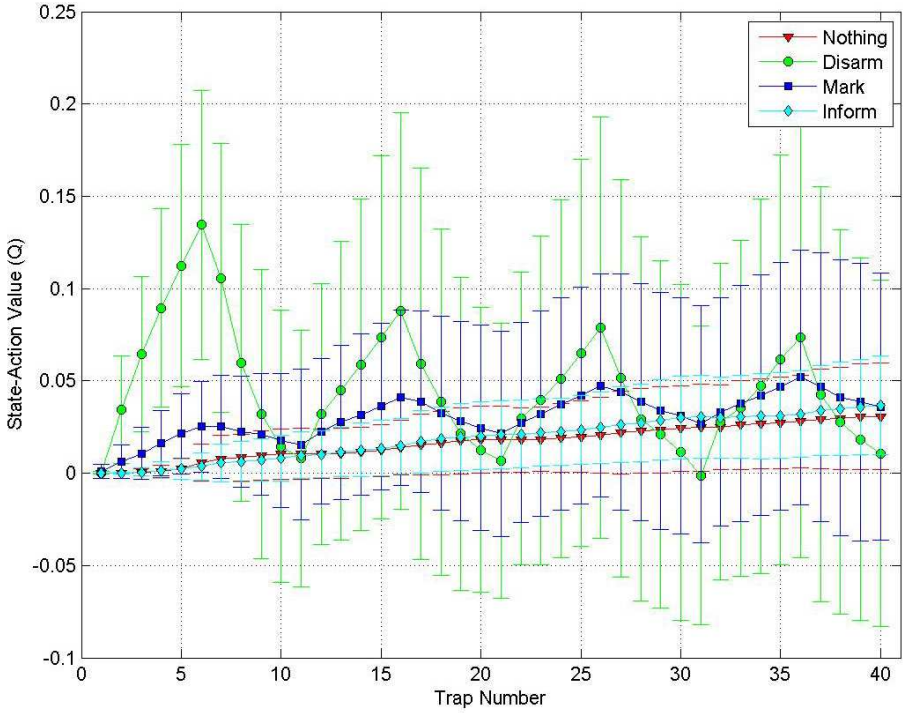

Figure A.19: The *Traps* domain results for a *Cautious* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.2 and the results are not normalized.

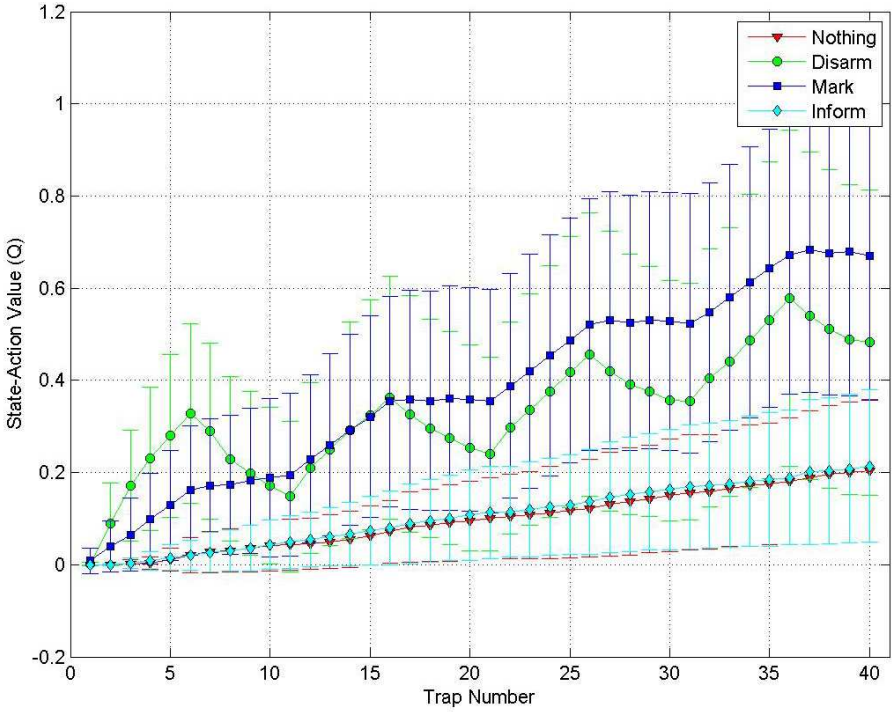

Figure A.20: The *Traps* domain results for a *Cautious* PC. In this experiment the trap difficulty switches back and forth after each 5 traps between easy and hard. The experiment starts with easy traps. The *approval* is fixed to 0.8 and the results are not normalized.

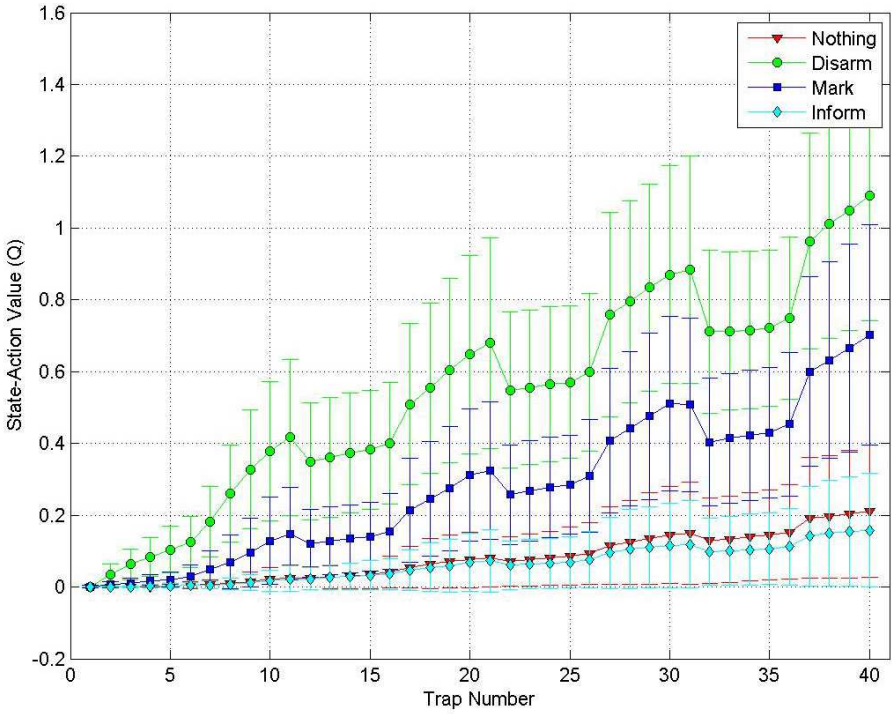

Figure A.21: The *Traps* domain results for a *Cautious* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are easy traps and the results are not normalized.

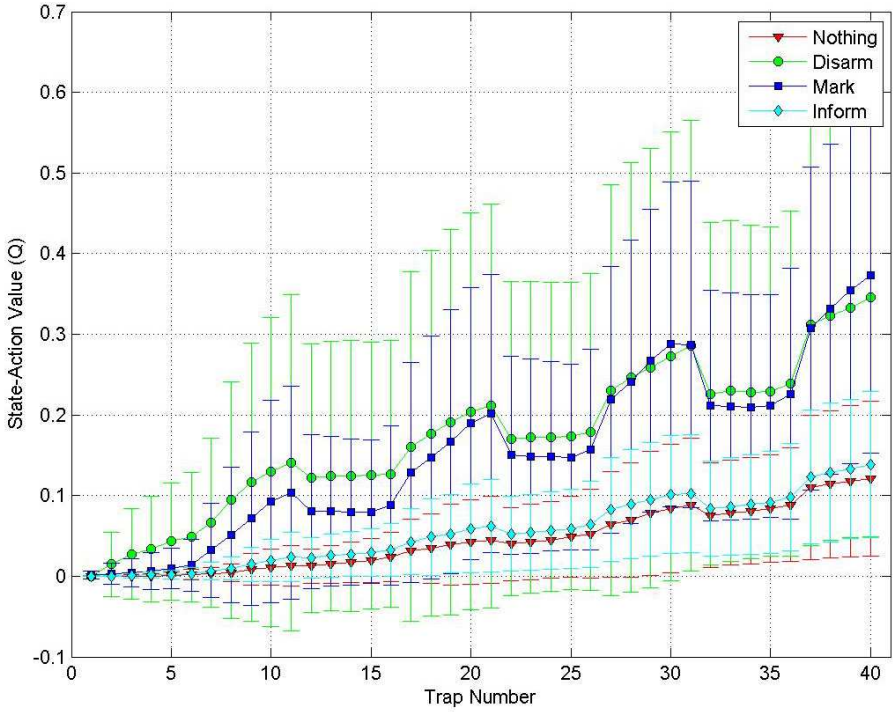

Figure A.22: The *Traps* domain results for a *Cautious* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are medium traps and the results are not normalized.

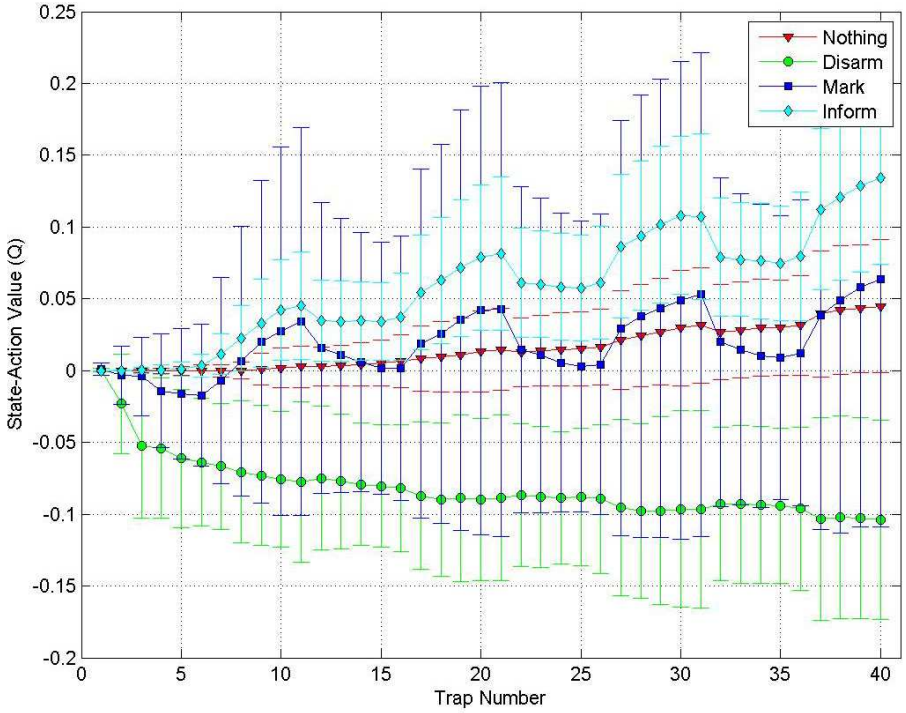

Figure A.23: The *Traps* domain results for a *Cautious* PC. In this experiment the *approval* switches back and forth after each 5 traps between 0.2 and 0.8. The experiment starts with low *approval*. All traps are hard traps and the results are not normalized.

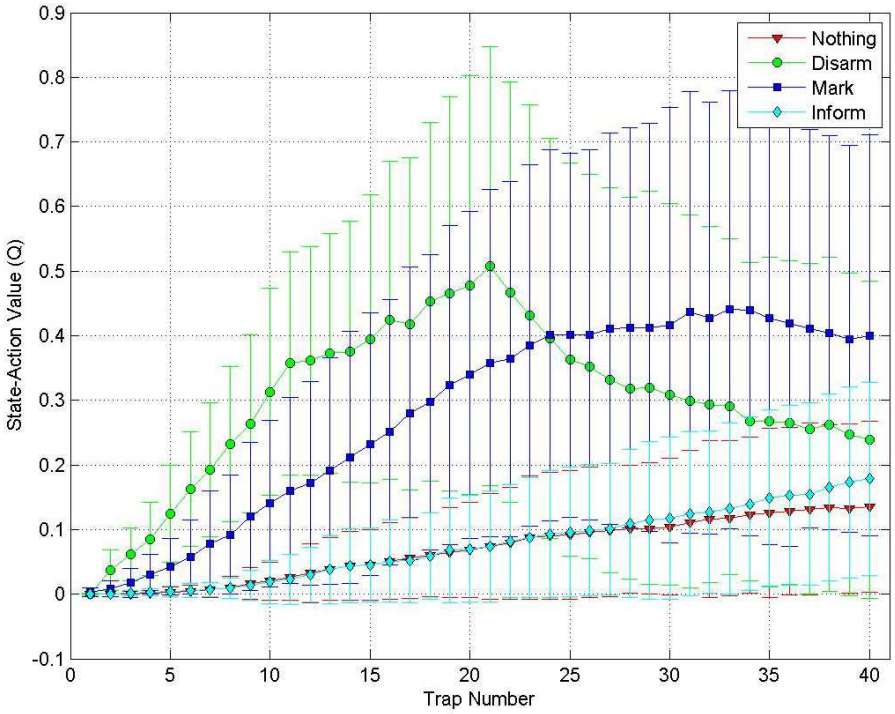

Figure A.24: The *Traps* domain results for a *Cautious* PC. In this experiment the *approval* starts from 0.2 and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are not normalized.

Appendix B

Single Runs of the Experiments

In this section, there are 10 single runs of the experiments presented. These diagrams show the actual interaction between the NPC and the PC, their decisions, and the outcomes of their actions. In all of these experiments the NPC first encounters 10 easy traps, then 10 medium traps, and finally 20 hard traps. The *approval* starts with an initial value and then varies afterwards. In the first 5 figures, the *approval* starts from 0.5 and in the second 5 figures, the *approval* starts from 0.2. The type of the PC for these runs is not revealed in the thesis. The purpose of presenting these results is to let the reader make an unbiased evaluation of the behavior of the NPC and the PC-NPC interactions over a single run rather than the average runs presented in Appendix A.

The values on the vertical axis of the diagrams shows the $Q(s, a)$ values for each action at the given state and time of the game. The values for all the available actions after each *Decision Event* are shown in the diagrams. Each step on the horizontal axis shows a *Decision Event* and the consequences of that decision event. There are 5 lines of characters under the horizontal axis of each diagram. The first line (closest one to the diagram) shows the *Decision Event*. The character "*C*" in the first line denotes that the NPC heard a command from the PC. The character "*T*" in the first line denotes that the NPC has encountered a new trap.

The second line from the top shows the type of the command that the PC gave to the NPC. The character "*M*" in this line means that the PC ordered the NPC to perform action "`[T]` Mark". Similarly, the character "*D*" in this line means that the PC ordered the NPC to perform action "`[T]` Disarm". It can be seen that there is no character in the second line under trap detection, designated by character "*T*" in the first line, since it is not a command and consequently no command type.

The third line from the top under the horizontal axis shows the action that the NPC performed after the *Decision Event* above it. The character "*n*" stands for performing action "[T] Nothing" and the character "*d*" stands for "[T] Disarm", "*m*" stands for "[T] Mark", "*i*" stands for "[T] Inform PC", and "*r*" stands for "[T] Refuse".

The fourth line of characters from the top under the horizontal axis shows the outcome of the action above it. The character "*S*" shows an action with *Success* outcome. The character "*F*" shows an action with *Fail* outcome, and the character "*C*" shows an action with *Critical Fail* outcome.

The fifth and last line of characters from the top under the horizontal axis shows the existence and type of the verbal reward. An empty space means that the PC did not give a verbal reward to the NPC after the NPC performed the action above it. The character "*P*" in this line means that the PC gave *Positive* verbal feedback to the NPC, and the character "*N*" means that the PC gave *Negative* verbal feedback to the NPC.

Each vertical line in these diagrams show a time step from the *Traps* decision domain perspective. On the vertical line above the characters, only the values of the actions that are available for the decision event at the step shown. Each time the action with the highest value is not selected, it means the NPC is exploring the other actions. In these experiments ϵ is set to 0.3 which means 30% exploration. The parameter τ is set to 0.2, α is set to 0.1, γ is set to 0.95, and λ is set to 0.

Notice from the graphs that the companion exhibits reasonable behavior. During the easy and medium traps, the companion tends to disarm them. As the traps get more difficult, the companion selects another option such as nothing, inform, or mark, depending on the behavior of the PC. For example in Figure B.1 to Figure B.5 the PC constantly commands the companion to disarm the traps and the companion gets hurt. By the end of the trace, the companion refuses to disarm the traps. In fact, in Figure B.3 to Figure B.5 the companion does not even reveal the existence of traps to prevent the PC from issuing a disarm command. By now the reader can probably infer the PC model that was used to generate the first 5 traces.

In Figure B.6 to Figure B.10, the PC gives fewer commands. In all five traces, the companion learns not to disarm the hard traps. However, the preferred action varies across the five traces. In figure B.6 and Figure B.8, the companion learns to do nothing since the PC does not give many mark commands. The PC gave zero mark commands in Figure B.6 and one mark command in Figure B.8, which resulted in a critical failure. However, in Figures B.9 and B.10 the PC gives several mark commands that result in success. That is

why mark becomes the preferred action for the PC. In Figure B.6 the PC gives several mark commands that resulted in critical failure so the companion learns to do nothing instead of marking. In Figures B.6 to B.10 the PC is not always giving disarm commands. Due to the variance in number of commands across these five traces, it is not clear whether this is a single PC model or not. In fact, it does not matter, since the companion responds appropriately based on the commands actually given and their outcome.

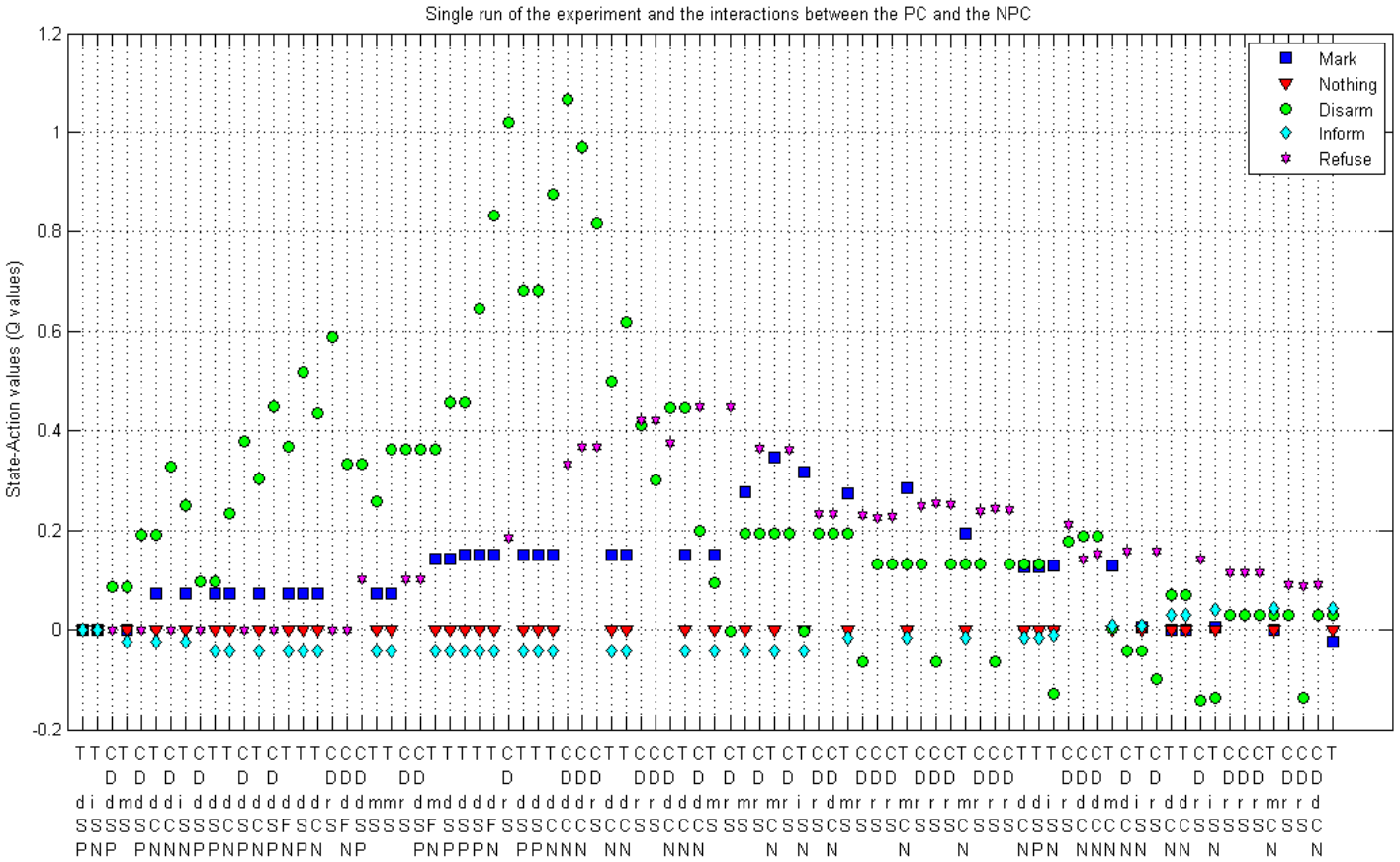

Figure B.1: Singe run of the game experiment. This diagram shows the PC-NPC interactions and the available actions and their values. In this experiment the *approval* starts from an initial value and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are not normalized.

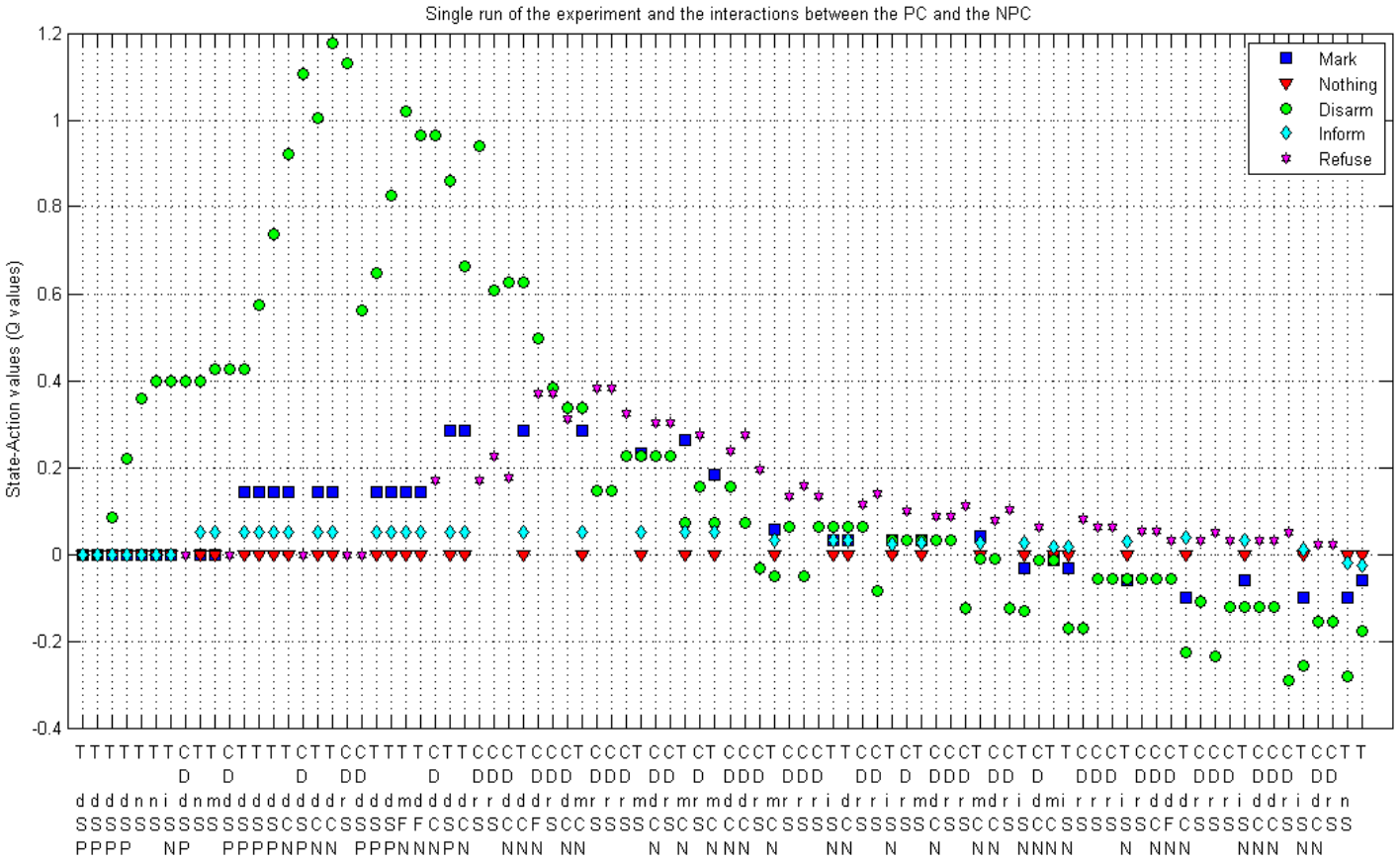

Figure B.2: Singe run of the game experiment. This diagram shows the PC-NPC interactions and the available actions and their values. In this experiment the *approval* starts from an initial value and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are not normalized.

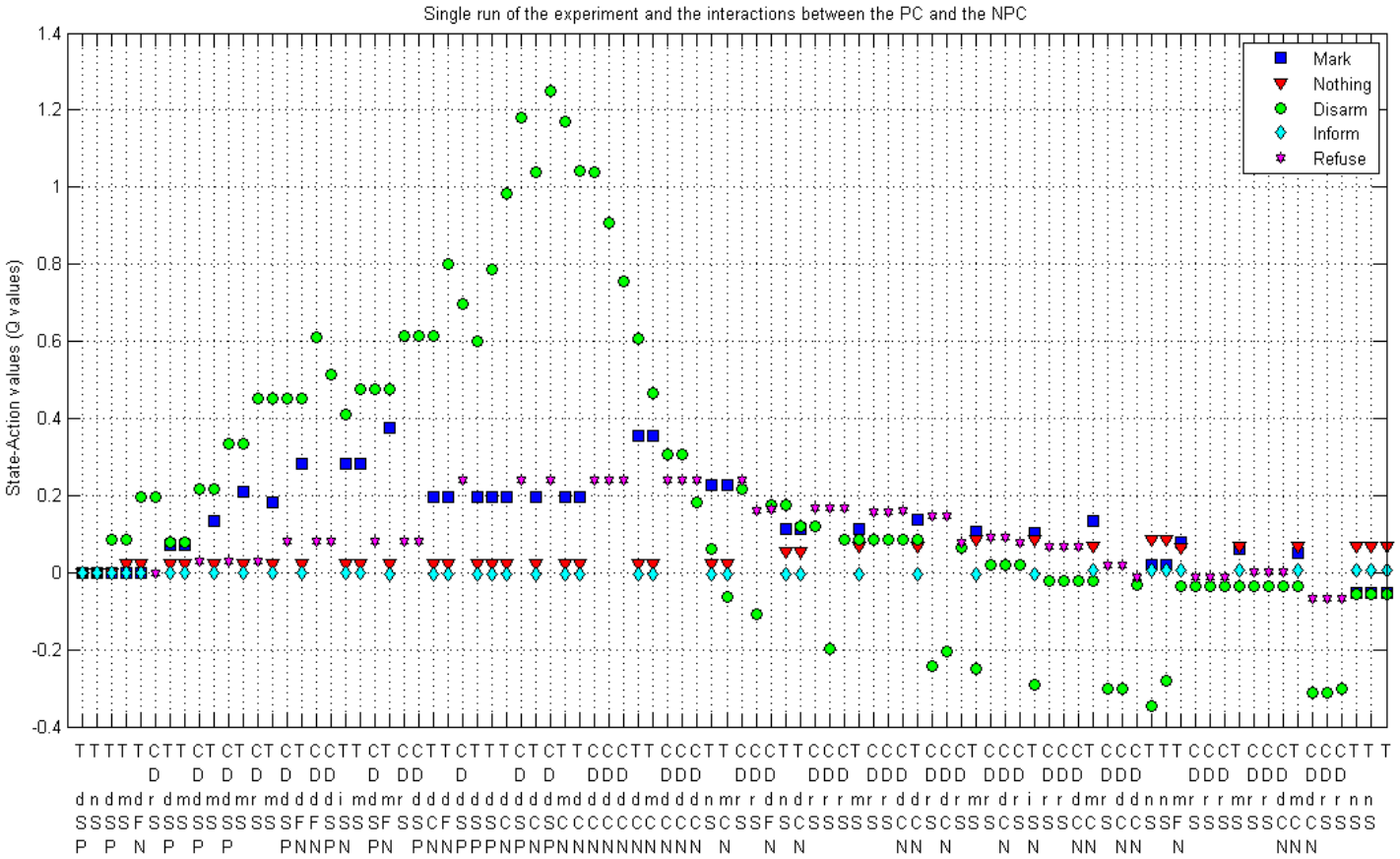

Figure B.3: Singe run of the game experiment. This diagram shows the PC-NPC interactions and the available actions and their values. In this experiment the *approval* starts from an initial value and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are not normalized.

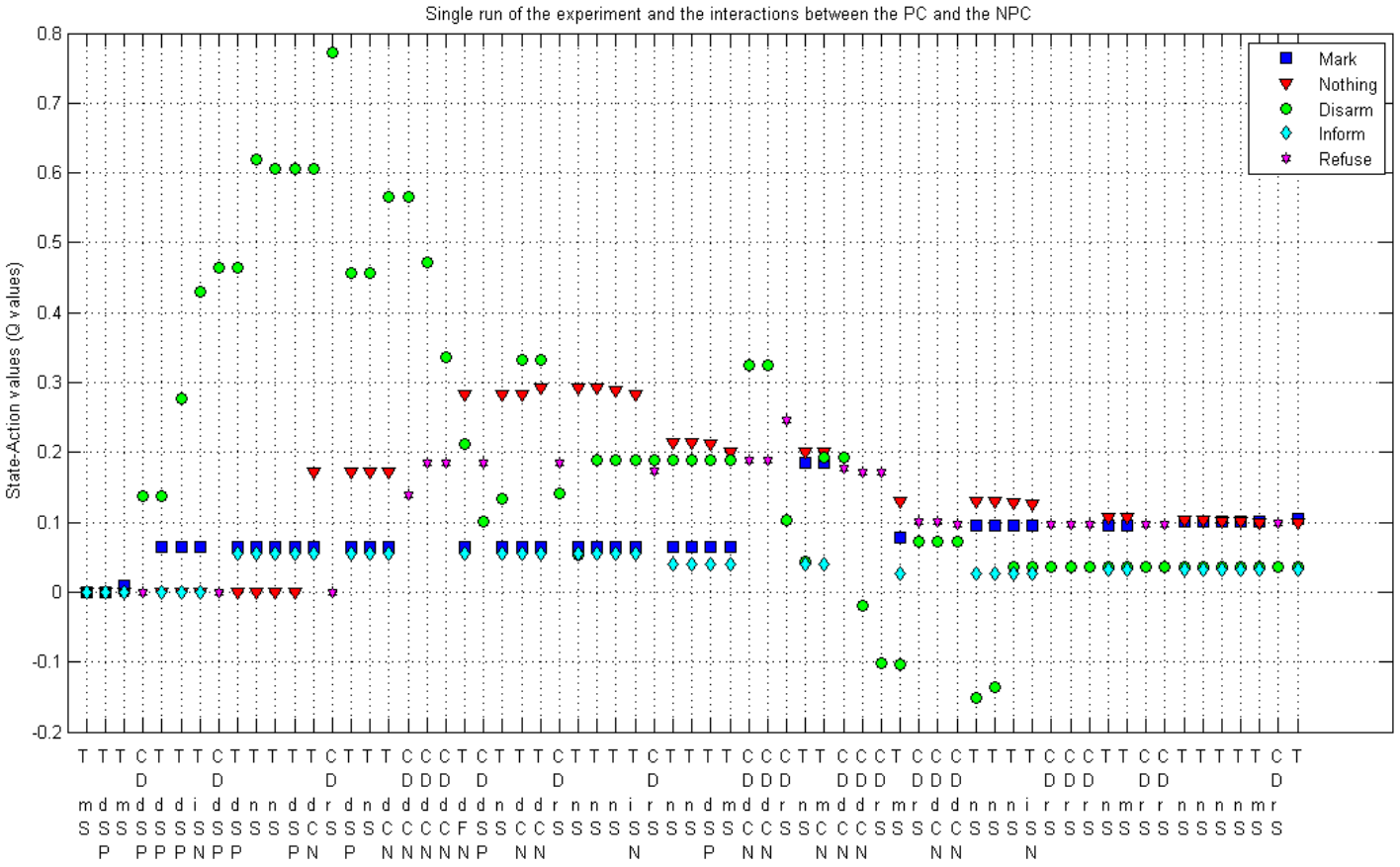

Figure B.4: Singe run of the game experiment. This diagram shows the PC-NPC interactions and the available actions and their values. In this experiment the *approval* starts from an initial value and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are not normalized.

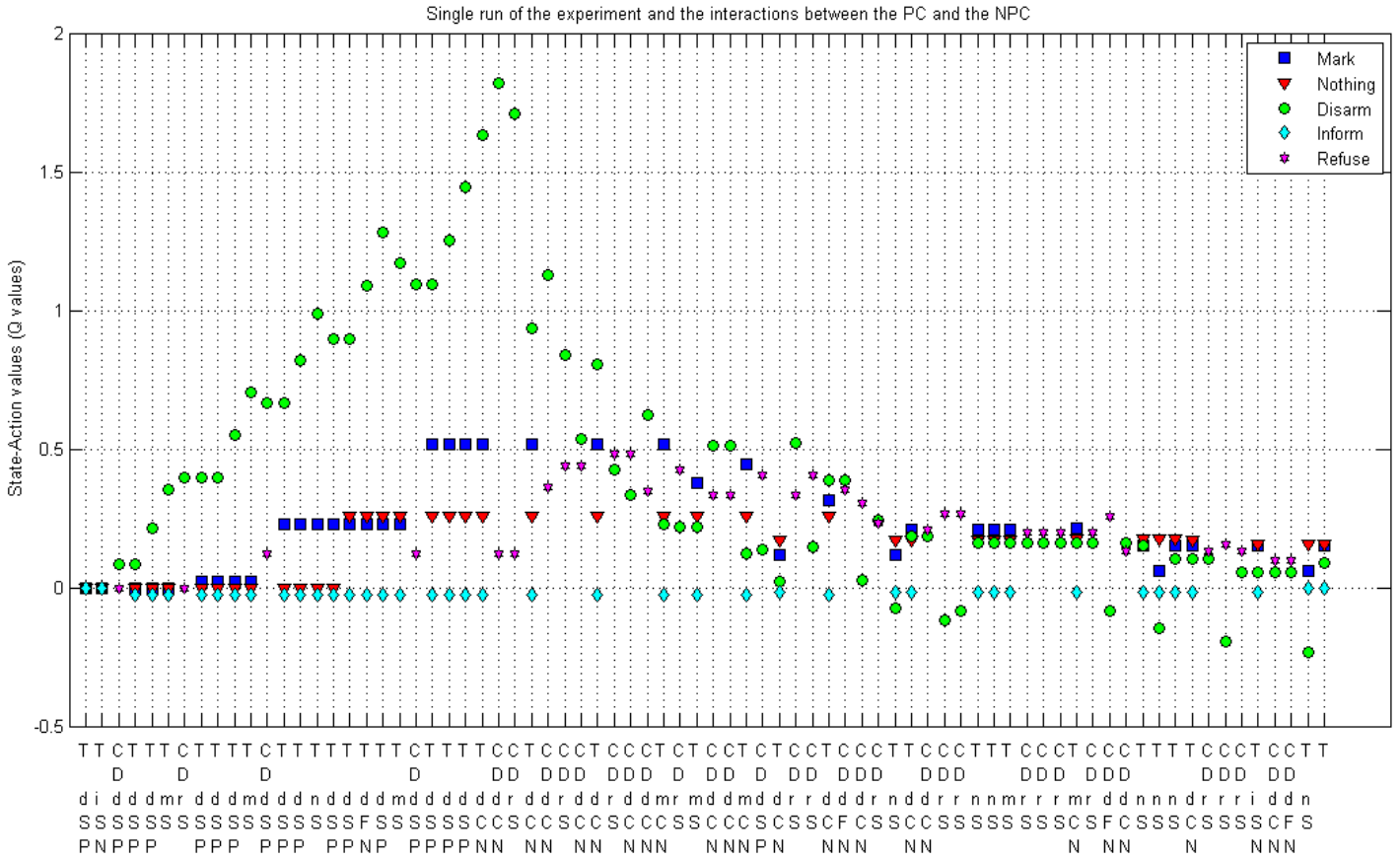

Figure B.5: Singe run of the game experiment. This diagram shows the PC-NPC interactions and the available actions and their values. In this experiment the *approval* starts from an initial value and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are not normalized.

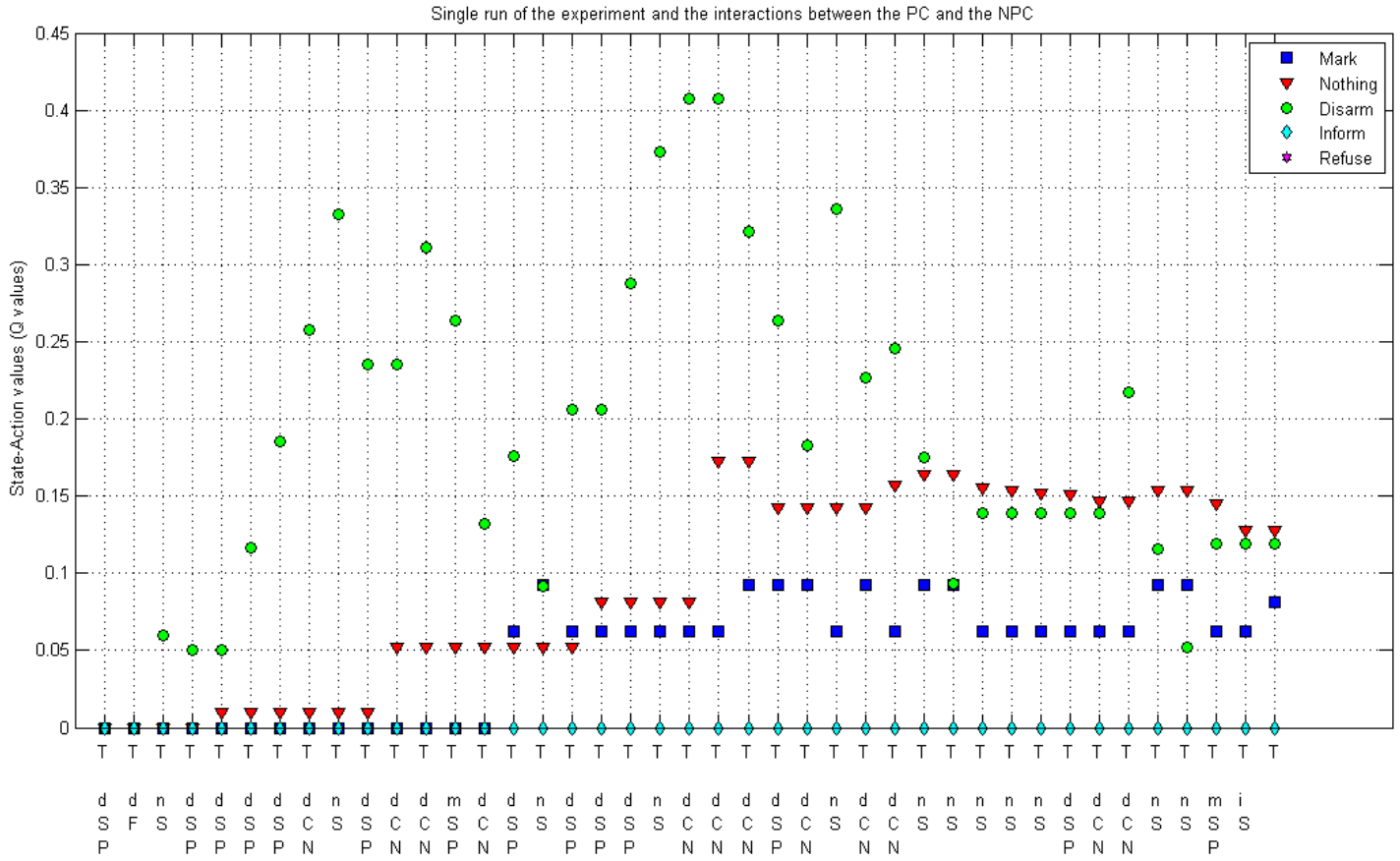

Figure B.6: Singe run of the game experiment. This diagram shows the PC-NPC interactions and the available actions and their values. In this experiment the *approval* starts from an initial value and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are not normalized.

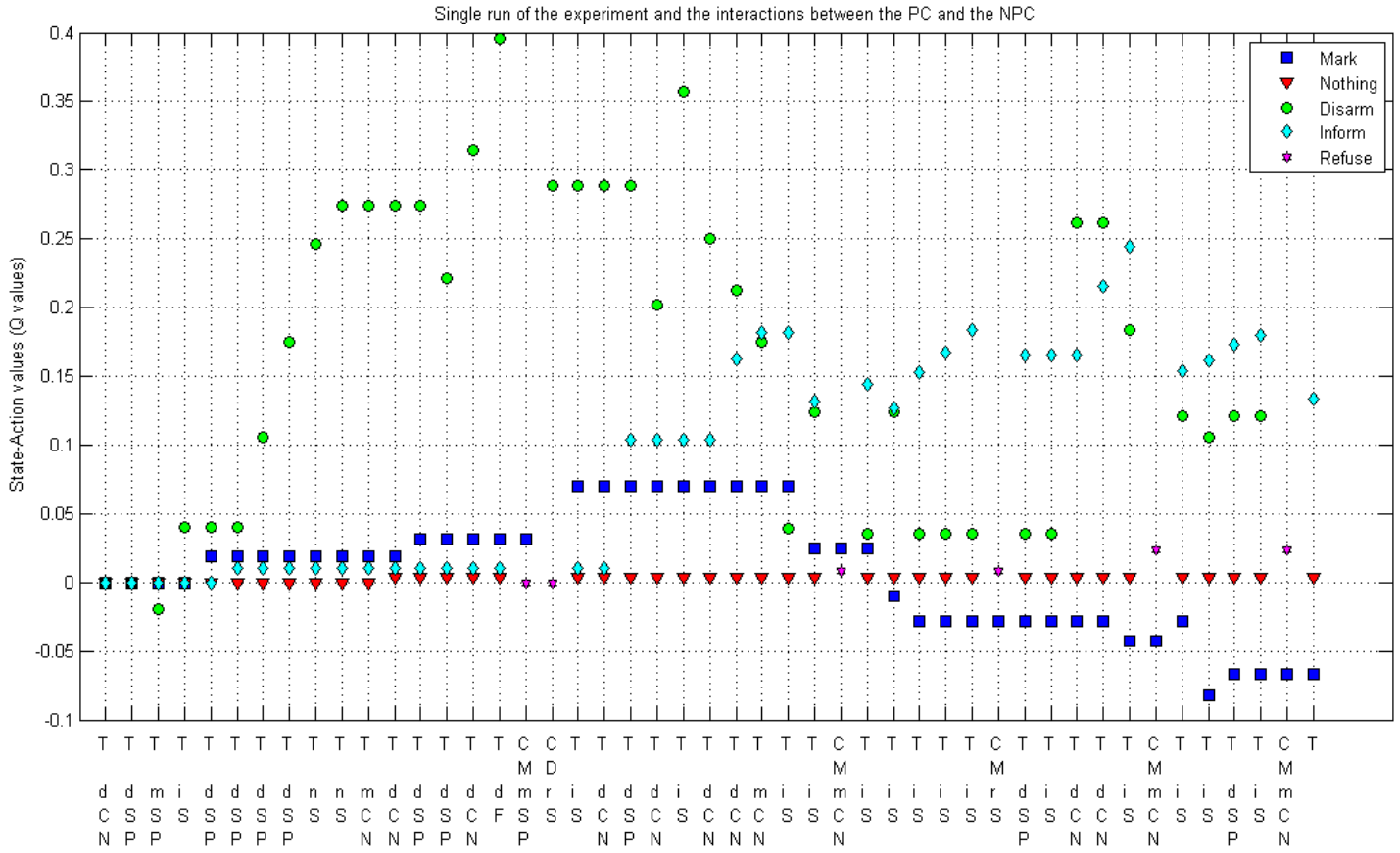

Figure B.7: Singe run of the game experiment. This diagram shows the PC-NPC interactions and the available actions and their values. In this experiment the *approval* starts from an initial value and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are not normalized.

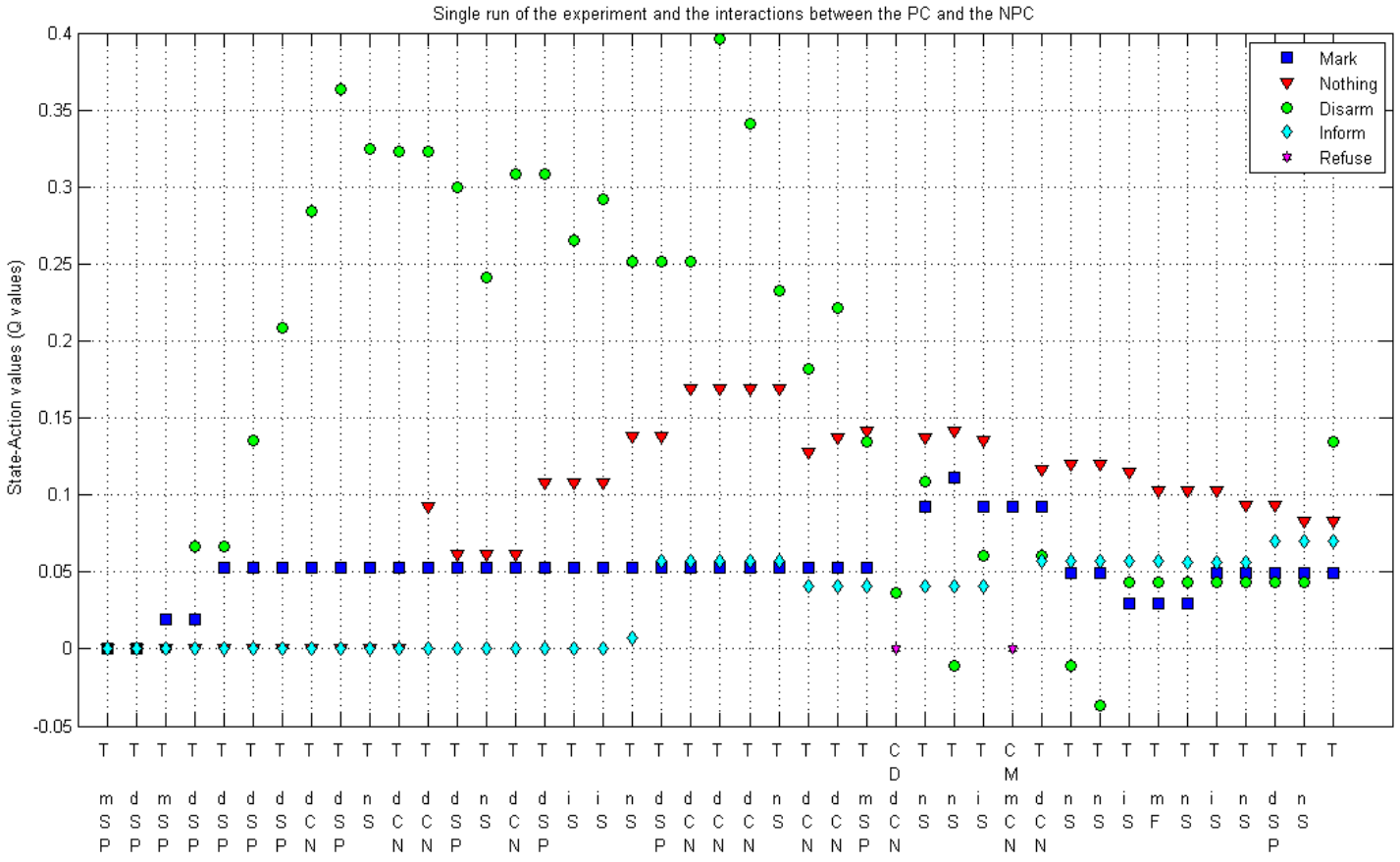

Figure B.8: Singe run of the game experiment. This diagram shows the PC-NPC interactions and the available actions and their values. In this experiment the *approval* starts from an initial value and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are not normalized.

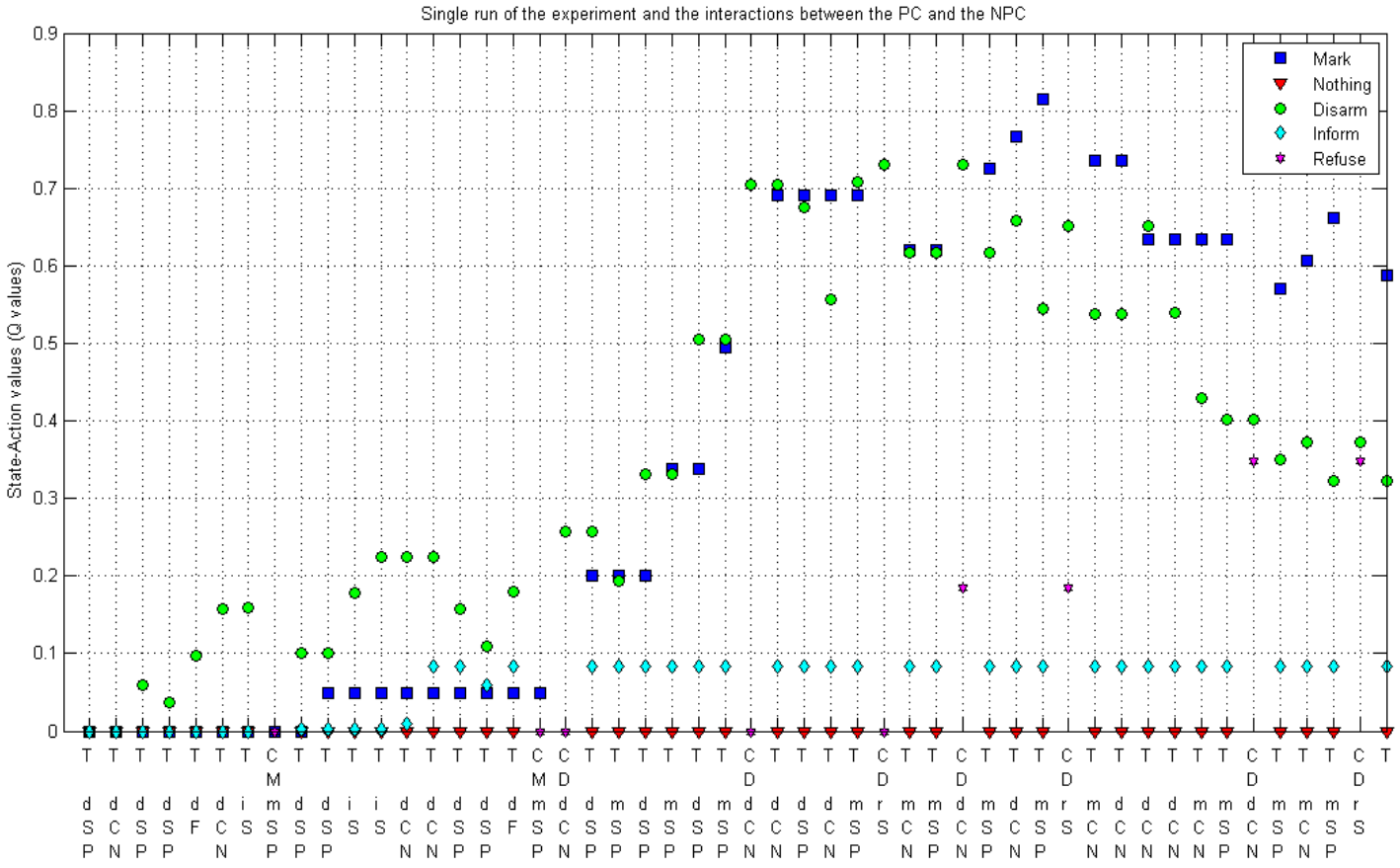

Figure B.9: Singe run of the game experiment. This diagram shows the PC-NPC interactions and the available actions and their values. In this experiment the *approval* starts from an initial value and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are not normalized.

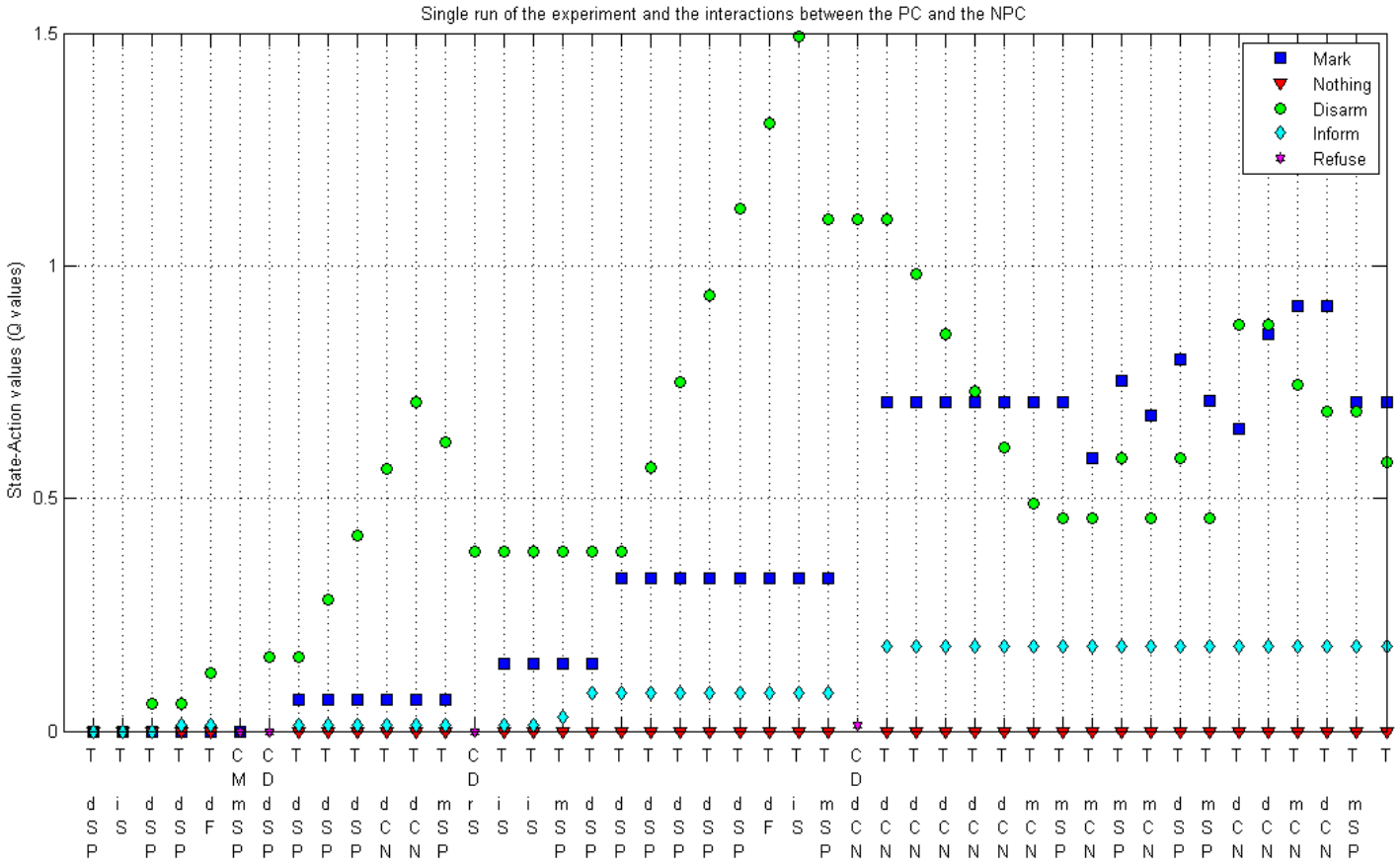

Figure B.10: Singe run of the game experiment. This diagram shows the PC-NPC interactions and the available actions and their values. In this experiment the *approval* starts from an initial value and varies afterwards. The NPC faces 10 easy traps, then 10 medium, and finally 20 hard traps. The results are not normalized.