Incorporating Human And Environmental Feedback For Robust Performance In Agent Domains

by

Mamatha Aerolla, B.E

A Thesis

In

COMPUTER SCIENCE

Submitted to the Graduate Faculty
of Texas Tech University in
Partial Fulfillment of
the Requirements for the Degree of

MASTER OF SCIENCE

Approved

Dr. Mohan Sridharan
Chairperson of the Committee

Dr. Michael Gelfond

Dr. Richard Watson

Peggy Miller
Dean of the Graduate School

May, 2011

ACKNOWLEDGEMENTS

This thesis would not have been possible without the support of many people. In first place, I owe my deepest gratitude to my professor Dr. Mohan Sridharan, who was abundantly helpful and offered invaluable assistance, support and guidance. I would also like to thank Dr. Michael Gelfond and Dr. Richard Watson for serving on my thesis committee and for their reviews and helpful suggestions.

Special thanks to my family and friends for their endless love and support.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS ii

ABSTRACT v

LIST OF FIGURES vi

1. **INTRODUCTION** 1

  1.1 **Terminology** 2

  1.2 **Problem Statement** 4

2. **RELATED WORK** 5

3. **BACKGROUND** 9

  3.1 **The Tetris Domain** 9

  3.2 **The General TAMER Framework** 11

    3.2.1 **TAMER: The learning algorithm** 12

    3.2.2 **Tetris-specific aspects of the algorithm** 13

    3.2.3 **Domain specifications** 14

    3.2.4 **Experimental Results** 16

  3.3 **QLearning Algorithm** 17

    3.3.1 **QLearning Experimental results** 18

  3.4 **Policy gradient methods** 19

  3.5 **Cross Entropy Method** 21

    3.5.1 **Value Function** 22

3.5.2 **Action selection** 22

3.5.3 **The Cross-Entropy Method** 22

3.5.4 **Experimental Results** 23

4. **PROBLEM FORMULATION** 25

4.1 **Determining the trust factor** 27

5. **THE KEEPAWAY SOCCER DOMAIN** 30

5.1 **RoboSoccer Keepaway Domain** 30

5.1.1 **The Keepers** 34

5.1.2 **The Takers** 36

5.2 **Reinforcement Learning When Applied To The Keepaway Soccer Domain** 38

5.3 **Sarsa(λ)** 38

5.4 **Function Approximation** 40

5.5 **Algorithm Description** 42

5.5.1 **RLstartEpisode** 42

5.5.2 **RLstep** 43

5.5.3 **RLendEpisode** 43

5.6 **Experimental Results** 44

6. **CONCLUSION** 50

BIBLIOGRAPHY 51

ABSTRACT

For an artificial agent to be fully autonomous and robust, it needs to be able to learn from and adapt to the environment. In order to keep the learning costs and complexity low, knowledge transfer from humans to agents becomes essential. Ideally, human users, including those without programming skills (i.e., non-technical users) should be able to teach agents desired behaviors using simple communication methods as quickly and as effortlessly as possible. Past work showed that giving human feedback can greatly reduce the sample complexity required to learn a good policy and can enable lay users to teach agents the behaviors they desire. However, prior work has focused on either training agents using human feedback or enabling agents to learn from environmental feedback. In case of domains with multiple agents, providing extensive human feedback becomes costly and infeasible and thus in such domains, it becomes necessary that the agents learn from limited human feedback. In this thesis, we enable an agent to exploit both environmental feedback and human input when it is available, thereby improving its performance significantly. Two domains are used to evaluate the agent's performance: "Tetris" and "Keepaway Soccer Simulator". While Tetris domain has a single agent, Keepaway Soccer domain is a more complex domain with multiple agents.

LIST OF FIGURES

1.1 The Reinforcement Learning Framework 3

3.1 Experimental Domain - Tetris 10

3.2 The General TAMER Framework 12

3.3 Performance of an agent trained using TAMER framework in Tetris domain: Maximum number of lines ($\approx$150) are cleared after training for $\approx$ 7 episodes 17

3.4 Performance of an agent trained using Q-Learning algorithm in Tetris domain. a maximum of $\approx$ 38 lines are cleared after training for $\approx$ 200 episodes. 19

3.5 Performance of an agent trained using policy gradient method in Tetris domain: $\approx$ 6000 lines are cleared after training for 10000 episodes . . 21

3.6 Performance of an agent trained using cross entropy method in Tetris domain: showing the performance of top 5 policies. Maximum of $\approx$ 18000 lines are cleared after training for 10000 episodes 24

4.1 Comparison of performance of an agent in Tetris domain. Our method (CE+HF) outperforms the cross-entropy method and the method described in [6]. In addition, performance of agent is better when human and environmental feedback are used, in comparison to using just the environmental feedback 26

4.2 Comparison of performance of an agent trained using policy gradient method. Performance of agent when both human and environmental feedback are given(PG+HF) is better than just using environmental feedback(PG) 29

5.1 A screen shot from the middle of a 3 vs. 2 keepaway episode in a 20m x 20m region. 31

5.2 The state variables used for learning with 3 keepers and 2 takers. Keepers and takers are numbered by increasing distance from K1, the keeper with the ball. The 13 lines and angles show the complete set of state variables. 35

5.3 TileCoding 41

5.6 Performance of Keepers trained with just human feedback. 46

5.7 A graph comparing the performance of keepers trained with "just the environmental feedback" and "both human and environmental feedbacks" 47

5.4 The three main routines of our $Sarsa(\lambda)$ implementation presented for a keeper. A taker has the sign of the reward, r, reversed. The set of macro-actions available, $A^s \subseteq A$, depends on the current state, s. For example, the keepers not in possession of the ball must select the Receive action, whereas the keeper with the ball chooses from among HoldBall and PasskThenReceive [48]. 48

5.5 Probability density function f(x) for a gamma(2.0, 0.28) distribution. Reinforcement signal h is received at time 0. If t and t'are times of consecutive time steps, credit for the time step at t is h$\times \int_{t'}^{t} f(x)dx$. 49

CHAPTER 1
INTRODUCTION

For an artificial agent to be fully autonomous and robust, it needs to be able to learn from and adapt to the environment. In order to keep the learning costs and complexity low, knowledge transfer from humans to agents becomes essential. The vast majority of knowledge transfer from humans to learning agents occurs through a programming language interface. This method, unfortunately, can only be harnessed by a small, technically trained subset of the population. In addition, this method is not so efficient in domains with large number of agents, as it is difficult for a single human (or a small set of humans) to give feedback to a large set of agents. Work has been done to create systems that allow humans to give advice to agents or to demonstrate the task for the agent. However, the complexity of these two methods makes it challenging to implement them in a way that is accessible to a user without technical training. Additionally, these methods require that the human be able to perform the task himself/herself.

Prior work [1] introduced methods to design agents that can be interactively shaped by human trainers. For instance, in the TAMER framework introduced in [1], human trainer can merely give positive and negative reinforcement signals (called *reward* in the learning agent community) to the agent. It only requires that a person observe the agent's behavior, judge its quality, and send a feedback signal that can be mapped to a scalar value (e.g., by button press).

Existing work, however, does not allow human training to be combined with autonomous learning based on a coded reward function. This research addresses the key challenge of how to best combine human and environmental feedbacks to improve the agent's performance. Before we discuss the problem statement, let us briefly discuss the terminology used henceforth in this document.

1.1 Terminology

This section briefly discusses the terminology used for the purpose of this thesis work.

1. **Reinforcement Learning :** Reinforcement learning is a computational goal oriented approach to learning from interaction. It is learning what to do, how to map situations to actions, so as to maximize a scalar reward signal. The learner is not told which action to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them [51](see Figure 1.1).

2. **Reward Function :** A reward function defines the goal in a reinforcement learning problem. It maps each perceived state (or state-action pair) of the environment to a single number, a reward, indicating the intrinsic desirability of that state.
   $\mathcal{R} : \mathcal{S} \times \mathcal{A} \times \mathcal{S}' \rightarrow \Re$

3. **Value of a state :** The value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.

4. **Value Function :** It maps each perceived state of the environment to the value of that state indicating the long term desirability of that state.Whereas a reward function indicates what is good in an immediate sense, a value function

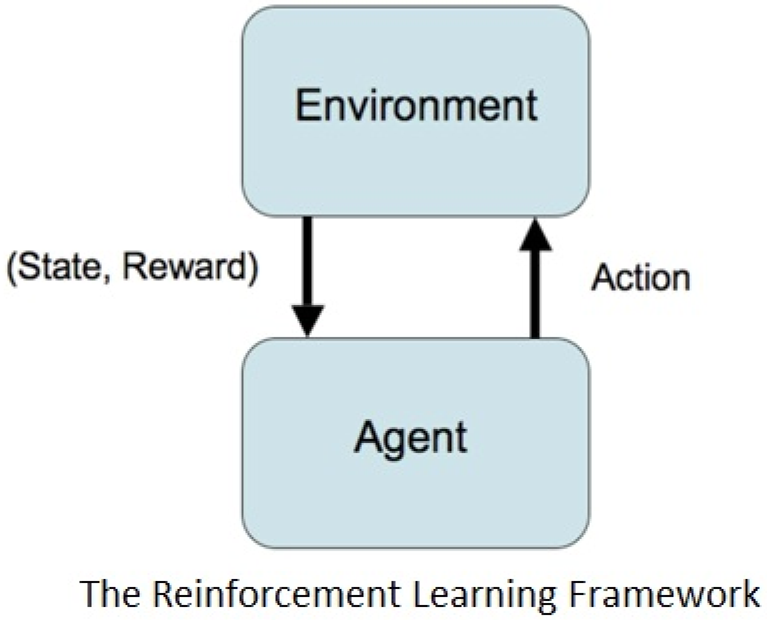

Figure 1.1. The Reinforcement Learning Framework

specifies what is good in the long run.

$\mathcal{V}_\pi : \mathcal{S} \rightarrow \Re$

5. **Policy :** Policy is a mapping from perceived states of the environment to actions to be taken when in those states. It defines the learning agent's way of behaving at a given time.

   $\pi : \mathcal{S} \rightarrow \mathcal{A}$

6. **Environmental Feedback :** If reward comes from the environment, it is called environmental feedback.

7. **Human Feedback :** If reward comes from a human, it is called human feedback

1.2 Problem Statement

In order for the learning agents to be useful to non-technical users, it is important to be able to teach agents how to perform new tasks using simple communication methods as quickly and as effortlessly as possible. The agent should be able to learn both in the presence and absence of human feedback. However, only limited research has been done on modeling the human feedback or training an agent to learn simultaneously from both human and environmental feedbacks.

In this thesis, we enable the agent to efficiently utilize the human feedback and the environmental feedback, such that the agent learns from the environmental interaction and efficiently combines the information with the limited human feedback when it is made available. Two simulated domains are used to evaluate the agent's performance: "Tetris" (with a single agent) and "Keepaway Soccer" (with multiple agents). The related work is summarized in Chapter II while Chapter III describes the work that forms the basis of the work described in this thesis. The performance of the agent trained using our method in Tetris domain is shown in Chapter IV. Chapter V briefly describes Keepaway soccer Domain. The performance of the agent trained by both human and environmental feedbacks in Keepaway Soccer Domain is shown in Chapter VI while Chapter VII presents the conclusions.

CHAPTER 2
RELATED WORK

Human-Robot Interaction (HRI) has received considerable attention in recent times and this chapter presents a brief review of the related work in this field. The goal of this review is to present a unified treatment of HRI-related problems, which is the underlying motivation for this thesis.

Early robot implementations were remotely operated devices with no or minimal autonomy. As stated in [50], Nicola Tesla demonstrated a radio-controlled boat in 1898, which he described as incorporating "a borrowed mind". Tesla described the first race of robots as mechanical men which will do the laborious work of the human race. He even envisioned one or more operators simultaneously directing 50 or 100 vehicles. Other examples include the Naval Research Laboratory's Electric Dog robot from 1923 which attempts to remotely pilot bombers during World War II, the creation of remotely piloted vehicles, and mechanical creatures designed to give the appearance of life. As technology evolved, the capabilities of remotely operated robots have grown (see [18] for a brief history). Complementing the advances in robot mechanics, research in artificial intelligence has attempted to develop fully autonomous robots.

As stated in [50], a breakthrough in autonomous robot technology occurred in the mid 1980s with work in behavior-based robotics [12,16]. A second important breakthrough for autonomy as it applies to HRI is the emergence of hybrid architectures. These architectures simultaneously allow sophisticated reactive behaviors that provide fundamental robot capabilities along with the high-level cognitive reasoning required for complex and enduring interactions with humans. Robot behaviors initially focused on mobility, but more recent contributions seek to develop lifelike anthropomorphic behaviors [32], acceptable behaviors of household robots [22], and desirable behaviors for robots that follow, pass, or approach humans [20,25,31].

Emerging from the early work in robotics, human factors experts have given considerable attention to two paradigms for human-robot interaction: teleoperation and supervisory control. At the teleoperation extreme, a human remotely controls a mobile robot or robotic arm. With supervisory control, a human supervises the behavior of an autonomous system and intervenes as necessary. This is important because robots are becoming part of our everyday social lives - and will increasingly become so. In future years, robots may become caretaking assistants for the elderly, academic tutors for our children, medical assistants, day care assistants, or psychological counsellors. Robots may become our co-workers in factories and offices, or maids in our homes. They may also become our "friends". Here, one of the major issues that is starting to gain attention is the role of a human [29]. While much of the discussion up to this point is with respect to humans and robots performing a task together, there are cases where the robot may have to interact with bystanders or with people who are not expecting to work with a robot. Examples include the tasks where coordination in hazardous, uncertain, and time stressed environments are the critical problems (e.g., search and rescue). In such environments, rescuers must have timely and accurate information on the status of the dangerous and evolving environment. To address these challenges, fundamental research advances are required in the design of distributed subsystems that can effectively coordinate with dispersed humans [8,9].

Even if a robot is capable of learning autonomously, it is likely to require human inputs during the learning phase. Existing HRI research has enabled robot to learn both offline as part of the design process [15,23] and online as part of interaction, especially long-term interaction [17,26]. Such learning includes improving perceptual capabilities through efficient communication between humans and robots [33, 17, 26, 33], improving reasoning and planning capabilities through interaction [30, 14], and improving autonomous capabilities [28]. Approaches to robot learning include teaching or programming by demonstration [34, 35, 36, 37, 38, 39], task learning [14, 26, 23], and skill learning including social, cognitive, and locomotion skills [40, 41, 42, 43, 44]. Some researchers are exploring biologically inspired learning models, including how teaching among humans or social animals can be used to train a robot [10,40, 45]. Others are exploring how learning can become more efficient if it leverages information about how the human brain learns in very few trials [19,11]. Many HRI researchers are also striving to develop systems that allow multiple robots and humans to interact with each other. In order to permit a small number of humans to supervise large robot teams, novel techniques and tools are required to understand and model human-robot interactions. For instance, Goal-Directed Task Analysis is an ongoing research direction. In order to design agents that learn efficiently and effectively from instructions, it is important to understand how people, who are not experts in Machine Learning or Robotics, will try to teach social robots [12].

This thesis focuses on enabling an agent to efficiently combine feedback from humans (who may be non technical users) and the environment. Chapter III briefly describes the existing work that forms the basis of our thesis. Chapter IV presents an instance of the implemented method along with experimental results of training a single agent in the Tetris domain, while Chapter V and chapter VI describe the multiagent Keepaway Soccer domain along with experimental results.

CHAPTER 3
BACKGROUND

This thesis uses two simulated domains, namely “Tetris” and “Keepaway Soccer” to evaluate the agent’s performance. This Chapter describes the Tetris domain along with the algorithms/frameworks that form the basis of the work described in this thesis and the corresponding experimental results. The keepaway Soccer domain is described in Chapter 5.

3.1 The Tetris Domain

Tetris is a game played on a w×h grid in which “tetrominoes” (falling pieces of different shapes) of four blocks, fall one at a time from the top of the grid and stack upon the grid’s base or any blocks below. If the blocks fall such that there is a row completely filled with blocks, then that row is cleared. All the blocks in that row disappear and all the blocks in higher rows shift down by a row. When the blocks stack up beyond the top of the grid, the game ends. The goal of a Tetris player is to maneuver the falling blocks in order to clear as many lines as possible before the game ends. Since clearing a line moves blocks down, away from the top, clearing lines allows a player to play for a longer duration of time. Thus, we measure success in Tetris by the number of lines cleared per game. A screenshot of the experimental domain of Tetris is shown in Figure 3.1.

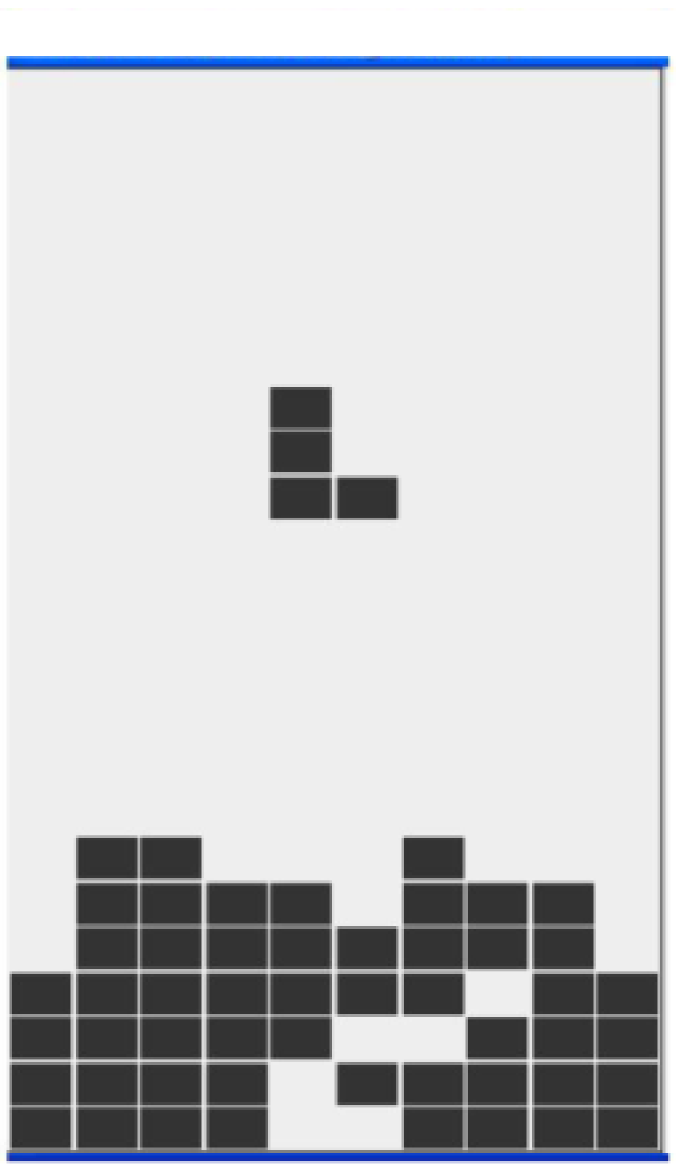

Figure 3.1. Experimental Domain - Tetris

Ours experiments to evaluate Tetris agent are based on an existing general framework called “Training an Agent Manually via Evaluative Reinforcement (TAMER) [1]” that allows a human to train a learning agent to perform a common class of complex tasks simply by giving scalar reward signals in response to the agent’s observed actions. While the “TAMER” framework deals only with training an agent using human feedback in Tetris Domain (described as *shaping problem* in [49]), the goal of this thesis is to combine human and also environmental feedbacks to train the agent in the same domain of Tetris. The TAMER framework described in the following section is an approach to the Shaping Problem that uses established supervised learning techniques to model a human’s reinforcement function.

3.2 The General TAMER Framework

Typically an agent learns autonomously via environmental interaction. However, the TAMER framework [1], allows a human trainer to give feedback, as shown in Figure 3.2. The agent’s interaction with the environment differs from the usual framework because the reward R function has been removed from the task specification, creating an MDP\R. Instead, reward comes from a human trainer who receives information about the current state, most likely via a visual representation. The TAMER framework for shaping agents shares much common ground with reinforcement learning, but there are some key differences. In reinforcement learning, agents seek to maximize return, which is a discounted sum of all future reward. In contrast, a TAMER agent does not seek to maximize a discounted sum of all future human reinforcement. Instead, it attempts to directly maximize the short-term reinforcement given by the human trainer. It does this because the trainers reinforcement signal is a direct label on recent state-action pairs. Correspondingly, the humans reinforcement function H is not an exact replacement for a reward function R within an MDP. Although it may be possible for a reinforcement learning algorithm to use H in place of a reward function, it would be unnecessary extra computation, since H already defines a policy. Based on the TAMERS’s evaluation of the agent’s recent performance, the trainer can choose to give reward in the form of a function that maps to a scalar value. The TAMER action selection scheme is given by $action = argmax_a \hat{H}(s, a, s')$

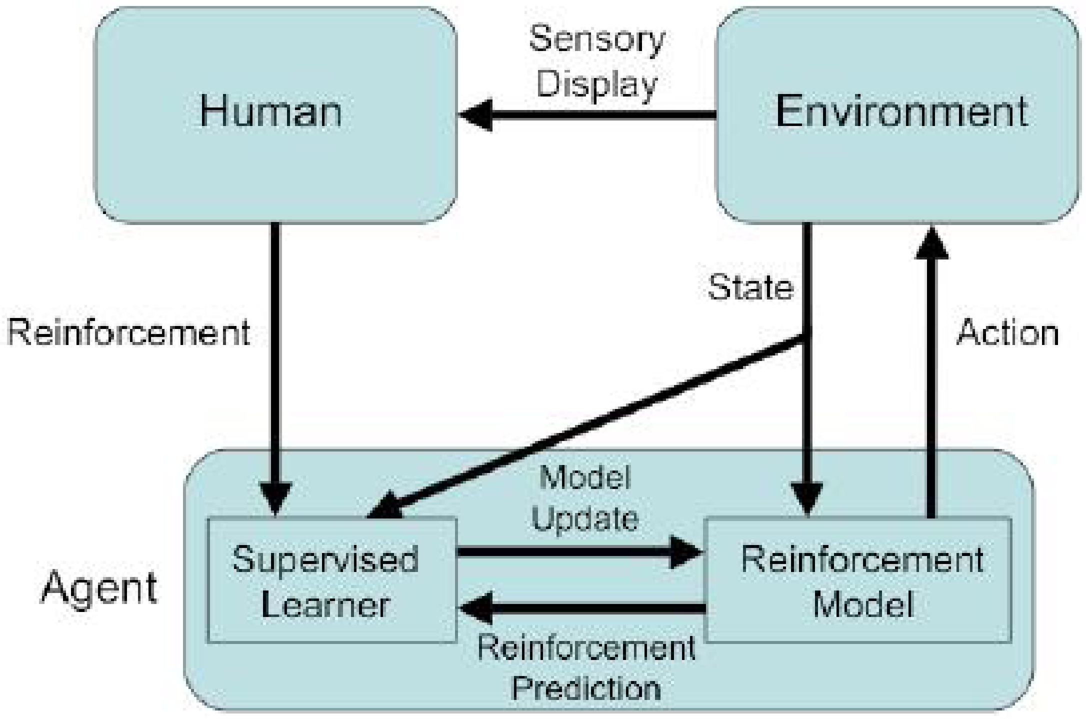

Figure 3.2. The General TAMER Framework

3.2.1 **TAMER: The learning algorithm**

An algorithm which implements the TAMER framework is described in this section (see Algorithm 1). The learning algorithm consists of an overarching function RunAgent() (shown in Algorithm 1) , and two functions that are called by RunAgent(): UpdateRewardModel() and ChooseAction(). RunAgent() begins by initializing the time t, the weights for the reward model and feature vectors. After initialization steps (steps 1−6 of Algorithm 1), ChooseAction(), i.e., Algorithm 3 is called and the first action is taken. RunAgent() then begins an infinite loop that occurs once per time step (line 7, Algorithm 1). In the loop, the agent first obtains a scalar measurement of the human trainer's reinforcement since the previous time step (line 10, Algorithm 1). If the reinforcement value is nonzero, then the error is calculated as the difference between the actual reinforcement and the amount predicted by the agent's reinforcement model (in updateRewardModel()). The calculated error and the most recent feature vector is then used to update the reinforcement model (line 6, updateRewardModel). The agent then obtains the current state description (line 17, Algorithm 1) and greedily takes the action 'a' that, according to the human reinforcement model, yields the largest predicted reinforcement. The new feature vector $\vec{f}$ is calculated and the chosen action is taken (line 16, Algorithm 1) before the loop restarts. The function UpdateReward-Model() i.e., Algorithm 2, updates its model of the human's reward pattern based on feedback on a previous action. The function ChooseAction(), i.e., Algorithm 3, chooses an action based on the current model of human reward. The core of the learning algorithm is a linear function approximator used in line 6 of ChooseAction() and updated in line 6 of UpdateRewardModel(). Unlike many other learning algorithms, only one hand-tuned parameter(the update step-size parameter) is used.

3.2.2 **Tetris-specific aspects of the algorithm**

For a board of size of 20×10, tetris has a state space of $\approx 2^{200}$. The work described in [1], uses 21 feature vectors to describe the Tetris state space of a 10×20 board.

Algorithm 1 *RunAgent*()

```
Require: $Input : \alpha$
1:  $t \leftarrow 0$
2:  $\overrightarrow{w} \leftarrow \overrightarrow{0}$
3:  $\overrightarrow{f_{t-2}} \leftarrow \overrightarrow{0}$
4:  $\overrightarrow{f_{t-1}} \leftarrow \overrightarrow{0}$
5:  $a \leftarrow ChooseAction(s_t, \overrightarrow{w})$
6:  $takeAction\{a\}$
7:  while true do
8:      $t \leftarrow t + 1$
9:      if $t \geq 2$ then
10:         $r_{t-2} \leftarrow getHumanFeedback()$
11:         if $r_{t-2} \neq 0$ then
12:             $\overrightarrow{w} \leftarrow UpdateRewModel(r_{t-2}, \overrightarrow{f_{t-1}}, \overrightarrow{f_{t-2}}, \overrightarrow{w}, \alpha)$
13:         end if
14:     end if
15:     $a \leftarrow ChooseAction(s_t, \vec{w})$
16:     $takeAction\{a\}$
17:     $s_t \leftarrow getState()$
18:     $f_{t-2} \leftarrow f_{t-1}$
19:     $f_{t-1} \leftarrow getFeatureVec(S_t)$
20: end while
```

3.2.3 Domain specifications

The Tetris state space is described by the 21 feature vectors [1]. Ten of these are the column heights. One is the height of the tallest column. Nine are the absolute values of the height differences between adjacent columns. One is the number of holes, defined as empty grid cells with at least one block above in the same column.

The action space consists of four actions i.e., *move left*, *move right*, *rotate clockwise* and *drop*. The actions *move left and move right* moves the tetromino by one block to left or right respectively in the 20×10 grid. The *rotate* action rotates the tetromino clockwise and *drop* action drops the tetromino to stack upon the grid's base or any blocks below.

Algorithm 2 *UpdateRewardModel*()

```
Require: $Input : r_{t-2}, \overrightarrow{f_{t-2}}, \overrightarrow{f_{t-1}}, \overrightarrow{w}, \alpha$
1: Set $\alpha$ as a parameter.
2: $\overrightarrow{\Delta f_{t-1,t-2}} \leftarrow \overrightarrow{f_{t-1}} - \overrightarrow{f_{t-2}}$
3: $projectedRew_{t-2} \leftarrow \sum_i (\overrightarrow{w_i} \times \Delta f_{t-1,t-2,i})$
4: error $\leftarrow$r$_{t-2} - projectedRew_{t-2}$
5: for i in range $(0, length(\overrightarrow{w}))$ do
6:     $w_i \leftarrow w_i + \alpha \times error \times \Delta f_{t-1,t-2,i}$
7: end for
8: return $\overrightarrow{w}$
```

Algorithm 3 *chooseAction*()

```
Require: $Input : s_t, \overrightarrow{w}$
1: $\overrightarrow{f_t} \leftarrow getFeatureVec(s_t)$
2: for each a $\in getActions(s_t)$ do
3:     $s_{t+1,a} \leftarrow T(s_t, a)$
4:     $\overrightarrow{f_{t+1,a}} \leftarrow getFeatureVec(s_{t+1,a})$
5:     $\overrightarrow{\Delta f_{t+1,t}} \leftarrow \overrightarrow{f_{t+1,a}} - \overrightarrow{f_t}$
6:     $projectedRew_a \leftarrow \sum_i (w_i \times \Delta f_{t+1,t,i})$
7: end for
8: return $\overrightarrow{w}$
```

Given the current state description, the agent's goal is to choose the action that will receive the most reward from the human. To do this, the agent models the human's "reward function" and greedily chooses actions that it expects will earn the most reward. After learning an accurate model of the human feedback, the agent can continue to perform the task in the absence of the human, choosing actions that are predicted to maximize the received reward if the human were present.

Similar to the experiments reported in [1], we ran experiments to evaluate the agent's ability to learn from human feedback. The experimental results obtained are presented in the following subsection.

3.2.4 Experimental Results

As shown in Figure 3.3, the number of episodes required to train a Tetris agent using TAMER framework is much smaller than standard reinforcement learning methods. Also, maximum number of lines cleared reaches approximately 150 lines. The mean number of lines cleared depends on the human trainer. However, the trainer need not be an expert. Anyone who can judge the performance of the agent by merely watching it, can train the agent. Figure 3.3 shows the performance of an agent trained by three different humans (player 1, player 2 and player 3). It can be seen from Figure 3.3 that the maximum number of lines cleared (in the case of the agent trained by player 1) after around seven training episodes is ≈150 while an agent trained using a standard reinforcement learning algorithm needs 120 training episodes to clear 50 lines [1]. In addition, the mean number of lines cleared is approximately 72 after five training episodes.

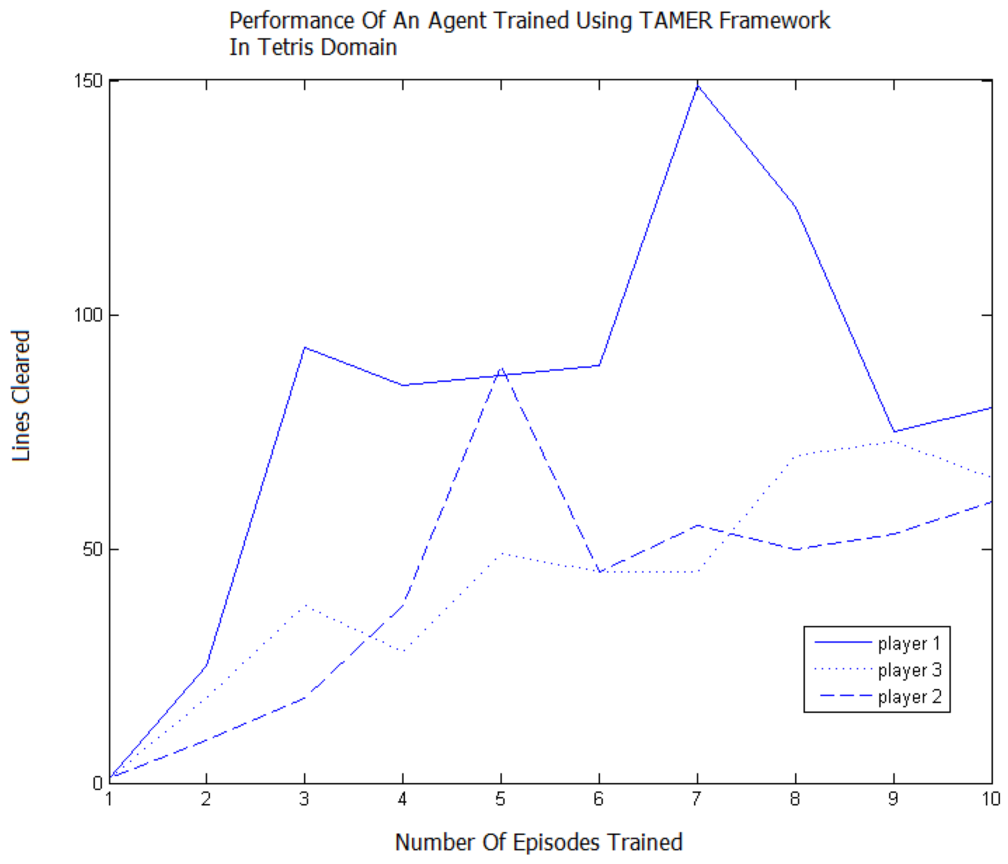

Figure 3.3. Performance of an agent trained using TAMER framework in Tetris domain: Maximum number of lines (≈150) are cleared after training for ≈ 7 episodes

3.3 QLearning Algorithm

The Q-learning algorithm (see Algorithm 4) is a reinforcement learning algorithm, introduced by Watkins in 1989 [5]. It is exploration insensitive, and is so far one of the popular and effective model-free algorithm for learning from delayed reinforcement. However, it does not address the scaling problem, and may converge quite slowly.

The Q-learning rule is given by

$$Q(s,a) = Q(s,a) + \alpha[\gamma + \gamma max_{a'} Q(s',a') - Q(s,a)]$$
$$\pi^*(s) = argmax_{a'} Q(s,a) \tag{3.1}$$

where s is the current state, a is the action taken in state s, s' is the resultant state, a' is the action taken in state s', r is the immediate reward, α is the learning rate,γ is the discount factor, Q(s,a) is the expected discounted reinforcement of taking action a in state s. The tuple <s, a, r, s'> is an experience tuple or transition function.

Algorithm 4 The Q-Learning Algorithm

```
1: For each (s,a), initialize table entry $Q(s,a) \longleftarrow 0$
2: for each episode do
3:     Choose a from s using policy derived from Q
4:     Take action a, observe reward r, and the new state s'
5:     Update the table entry for Q(s, a) using the equation
       $Q(s,a) = Q(s,a) + \alpha[r + \gamma max Q(s',a') - Q(s,a)]$
6:     $s \leftarrow s'$
7: until s is terminal
```

3.3.1 QLearning Experimental results

Based on existing work, we ran experiments to measure the agent's ability to learn from environmental feedback using Q-learning algorithm. Figure 3.4 shows the performance of Q-learning on Tetris domain. It is observed that a maximum of $\approx$ 38 lines are cleared after training for approximately 200 episodes. The poor performance of Q-learning is probably because it requires a good estimate of future rewards in order to function properly, and that the stochastic nature of Tetris severely limits the accuracy of these estimates [2]. We conclude that Q-Learning is not a good choice for training a Tetris agent. Hence, we implemented more appropriate methods such as policy gradient(PG) and cross entropy(CE) on the Tetris domain.

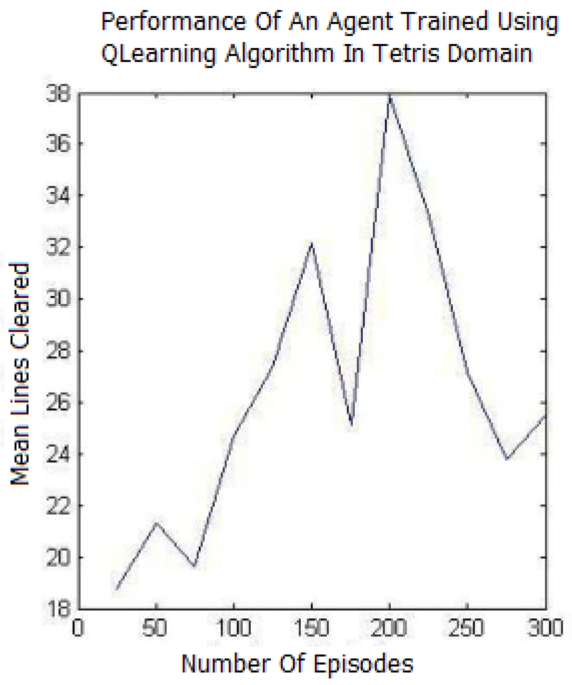

Figure 3.4. Performance of an agent trained using Q-Learning algorithm in Tetris domain. a maximum of ≈ 38 lines are cleared after training for ≈ 200 episodes.

3.4 **Policy gradient methods**

Policy gradient methods [4] are reinforcement learning techniques that rely upon optimizing parameterized policies with respect to the expected return (long-term cumulative reward) by gradient descent. They do not suffer from many of the problems that have been marring traditional reinforcement learning approaches, such as the lack of guarantees of a value function, the intractability problem resulting from uncertain state information and the complexity arising from continuous states and actions. Instead of approximating a value function and using that to compute a deterministic policy, PG methods approximate a stochastic policy directly using an independent function approximator with its own parameters.

Let θ denote the vector of policy parameters and ρ be the the average reward per step of the corresponding policy. Then, in the policy gradient approach, the policy parameters are updated approximately proportional to the gradient:

$$\Delta\theta = \alpha(\frac{\delta\rho}{\delta\theta}) \tag{3.2}$$

where α is a positive-definite step size. If such a gradient can be computed, then θ can usually be guaranteed to converge to a locally optimal policy in the performance measure ρ. Unlike the value-function approach, small changes in θ can cause only small changes in the policy. Many approaches exist for estimating the policy gradient, and these have been implemented in the form of open source libraries. We chose the Libpg library and used it in all the experiments.

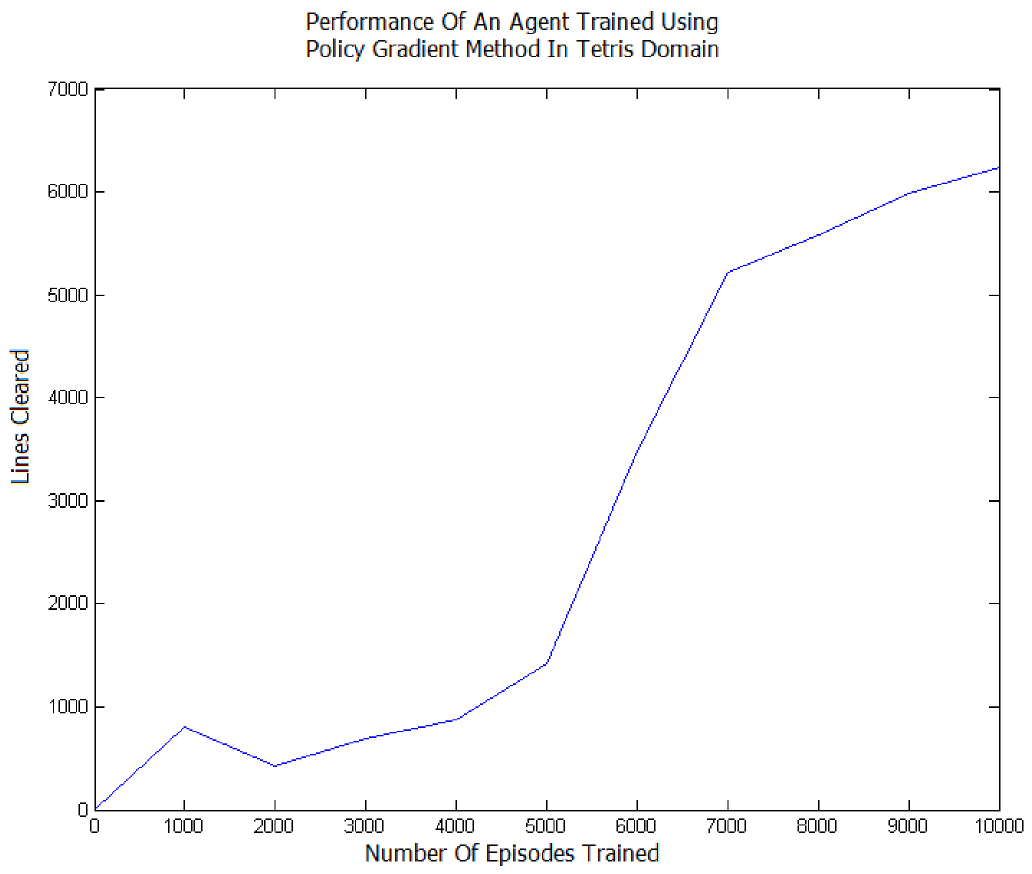

Figure 3.5. Performance of an agent trained using policy gradient method in Tetris domain: ≈ 6000 lines are cleared after training for 10000 episodes

The experimental results are as shown in Figure 3.5. It is observed that the maximum number of lines that can be cleared using policy gradient is ≈6000 lines after training for 10,000 episodes.

3.5 Cross Entropy Method

The cross-entropy method [3] is an efficient and general optimization algorithm. When used in the Tetris domain, it outperforms policy gradient or any RL algorithm. We apply cross entropy(CE) to learn the weights of the feature vectors, drawing each weight from an independent gaussian distribution.

The CE method deals with combinatorial optimization problems in an efficient manner. The basic steps of CE consist of generating random data samples and maintaining a distribution of good samples according to some scoring mechanism in order to generate new samples.

3.5.1 Value Function

CE can be applied to reinforcement learning by learning (optimizing) a value function given by

$$V(S) = \sum_{k=1}^{n} \omega_k \phi_k(s) \tag{3.3}$$

where ω_k are weights to be learnt and ϕ_k are the feature vectors.

3.5.2 Action selection

The learned value function V_w is used to decide where to place a falling tetromino. In order to do so, we evaluate the value of the resulting state (using V_w) when it is placed in each column and for every possible rotation. Finally, we choose the column and rotation with the highest value.

3.5.3 The Cross-Entropy Method

The task of the cross-entropy(CE) method is to optimize the weights of a value function as described in Equation 4, which determines the actions of the agent. The CE method then optimizes $S(\overrightarrow{\Omega})$, which is a real valued scoring function of the performance of the agent using weight vector $\overrightarrow{\Omega} = |\omega_1, ..., \omega_n|$

$$\overrightarrow{\Omega^*} = argmax_{\overrightarrow{\Omega}} S(\overrightarrow{\Omega}) \tag{3.4}$$

3.5.4 Experimental Results

The experimental results are shown in Figure 3.6. The graph gives the performance of the five best performing policies. It is observed that the maximum number of lines that can be cleared using policy gradient is ≈ 18000 after training for 10,000 episodes.

Thus, after running experiments on giving human feedback in the Tetris domain similar to prior work, we noticed that the TAMER framework [1], increased the learning speed of the agent. The results obtained when reinforcement learning algorithms are implemented on Tetris domain show that policy gradient and cross entropy methods perform better than Q learning algorithm. The next Chapter desribes our effort to combine human and environmental feedbacks along with experimental results.

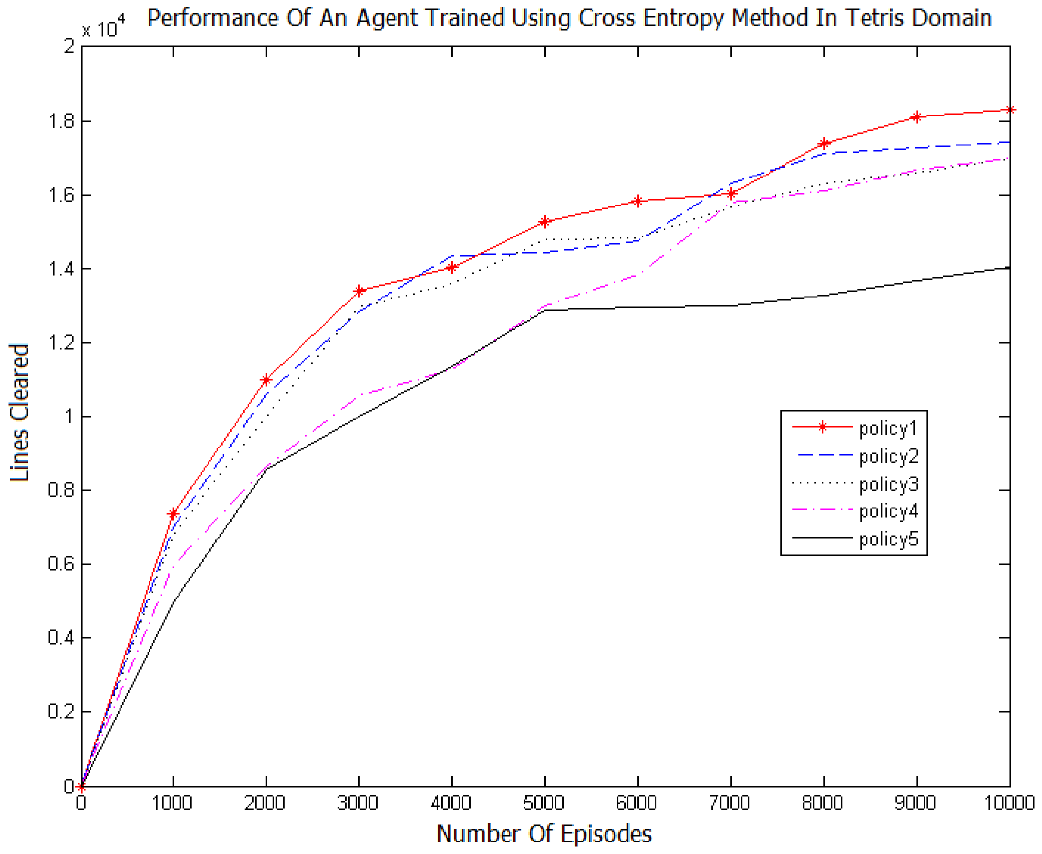

Figure 3.6. Performance of an agent trained using cross entropy method in Tetris domain: showing the performance of top 5 policies. Maximum of ≈ 18000 lines are cleared after training for 10000 episodes

CHAPTER 4
PROBLEM FORMULATION

As stated earlier in Chapter 2, in order to permit a small number of humans to supervise large team of agents, novel techniques and tools are required to understand and model human-agent interactions. This thesis focuses on enabling an agent to learn by efficiently combining human and environmental feedbacks. The technique we implemented to combine human and environmental feedback is that of a function approximator given by

$$a = argmax_a f(R, H) \tag{4.1}$$

where R is the environmental feedback and H is the human feedback. When evaluating the agent in Tetris domain, we considered the weighted linear combination of both the feedbacks. it can be mathematically written as

$$a = argmax_a[R + weight * H] \tag{4.2}$$

In case of Keepaway Soccer domain, we considered exponential combination and the weighted linear combination of both the feedbacks(Chapter 5)

Our experiments in Tetris domain are based on the assumption that the rewards from human are instantaneous, i.e., $\mathcal{R} : \mathcal{S} \times A \times \mathcal{S}' \rightarrow \Re$

Human feedback is rich in information yet flawed [6]. The human can get tired with time or may not be good at judging the performance of the agent. The weight is the trust factor(described in Section 3.1), which tells us how much to trust the human feedback. This makes our method do well despite the flaws in human feedback.

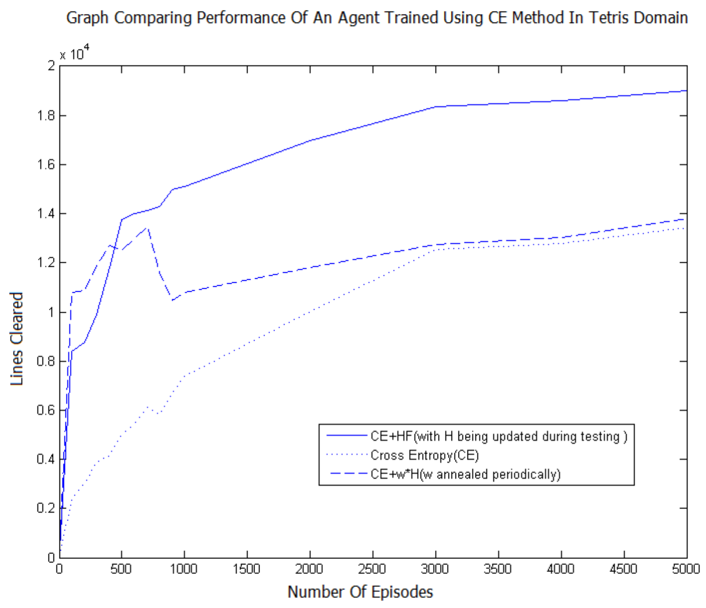

Figure 4.1. Comparison of performance of an agent in Tetris domain. Our method (CE+HF) outperforms the cross-entropy method and the method described in [6]. In addition, performance of agent is better when human and environmental feedback are used, in comparison to using just the environmental feedback

When the performance of the weighted combination of the human and environmental feedback is compared with just the environmental feedback, it is observed that the weighted combination of human and environmental feedback has much better performance than just the environmental feedback. This observation holds true for the CE (see Figure 4.1) and policy gradient (Figure 4.2) methods and even in the case where only the human feedback is considered.

Figure 4.1 also gives the performance of the agent when trained by a method described in [6] where human and environmental feedbacks are combined such that

$a = argmax_a[R + weight * \hat{H}]$, where R(s,a,s') is the environmental reward. The weight here is an input parameter unlike the trust factor in our case and is annealed periodically by some factor at the end of each episode. It is observed from Figure 4.2 that the agent trained using our method (CE+HF with H being updated during testing) performs better than the agent trained by cross entropy method or by the method described in [6]. It can also be seen from Figure 4.2 that the agent trained by giving both human and environmental feedback (in case of policy gradient) performs better than the agent trained using just the environmental feedback.

4.1 Determining the trust factor

Trust factor tells us how much to trust the human feedback. Here the performance measure is the number of lines cleared. In order to determine this trust factor, we followed the steps given below:

1. We first obtain the top five best performing policies(P_1, P_2, P_3, P_4, P_5) by just using the environmental feedback and computing the corresponding number of lines cleared. Let L_1, L_2, L_3, L_4, L_5 be the number of lines cleared in case of P_1, P_2, P_3, P_4, P_5 respectively.
2. For each policy repeat steps 3,4.
3. Let a human trainer train the agent.
4. Keep count of the average number of times the best action chosen by human feedback matches the best action based on the environmental feedback.
5. Let the averages computed in step 4 be m_1, m_2, m_3, m_4, m_5 for the policies P_1, P_2, P_3, P_4, P_5 respectively.

6. Now the trust factor is given by

$$\text{trust factor} = \frac{(L_1 * m_1 + L_2 * m_2 + L_3 * m_3 + L_4 * m_4 + L_5 * m_5)}{(L_1 + L_2 + L_3 + L_4 + L_5)} \tag{4.3}$$

The human reward model can now be updated using human feedback even during the testing phase. In other words, instead of separate training and testing phases, the trust factor is updated incrementally based on the degree of match between the action choices based on human feedback and those based on environmental feedback. If this match count is m, the number of episodes is k, number of lines cleared in (k-2)th episode is l_1, and number of lines cleared in (k-1)th episode is l_2, the trust factor for k^{th} episode is given by:

$$\text{trust factor for } k^{th} \text{ episode} = \frac{(l_1 * trustfactor + l_2 * m)}{(l_1 + l_2)} \tag{4.4}$$

In case of incorrect feedback from human, this incremental update mechanism ensures that, the agent can quickly adapt by revising the corresponding trust factor.

Hence, our method of linear combination of human and environmental feedbacks improves the performance of the agent in Tetris Domain. Considering the fact that the Tetris domain is relatively simple, a more challenging domain such as Keepaway Soccer is considered in the next chapter.

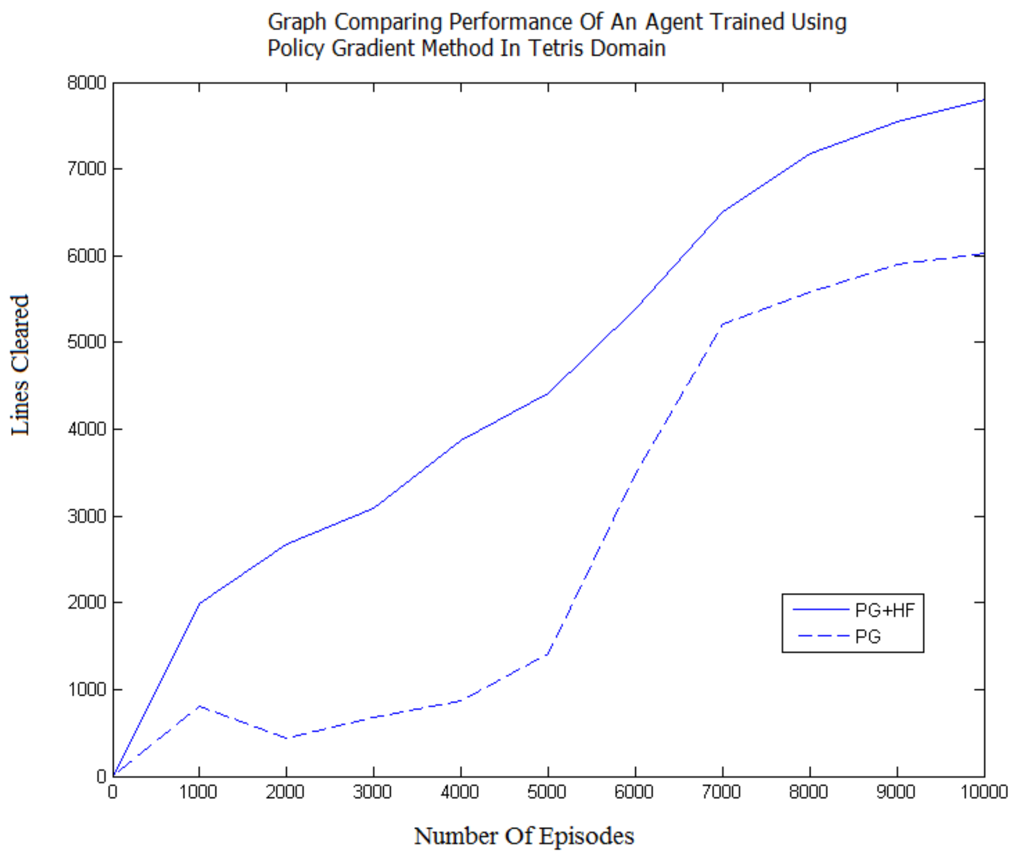

Figure 4.2. Comparison of performance of an agent trained using policy gradient method. Performance of agent when both human and environmental feedback are given(PG+HF) is better than just using environmental feedback(PG)

CHAPTER 5
THE KEEPAWAY SOCCER DOMAIN

This chapter investigates the combination of human and environmental feedbacks in the more complex domain of keepaway soccer. The hypothesis is that the combination of feedbacks will enable agents in the Keepaway domain to perform better than the individual application of either feedback mechanism. The keepaway soccer domain [48] is described below.

5.1 **RoboSoccer Keepaway Domain**

Keepaway is a subtask of robot soccer (involving $5-9$ players rather than the full 22) in which one team, the keepers, tries to maintain possession of a ball within a limited region, while another team, the takers, tries to gain possession. Whenever the takers take possession or the ball leaves the region, the episode ends and the players are reset for another episode (with the keepers being given possession of the ball again). The domain is implemented within the RoboCup soccer simulator [48]. Parameters of the task include the size of the region, the number of keepers, and the number of takers. Figure 5.1 shows a screen shot of an episode with 3 keepers and 2 takers (called 3 vs. 2, or 3v2 for short) playing in a 20m $\times$ 20m region.

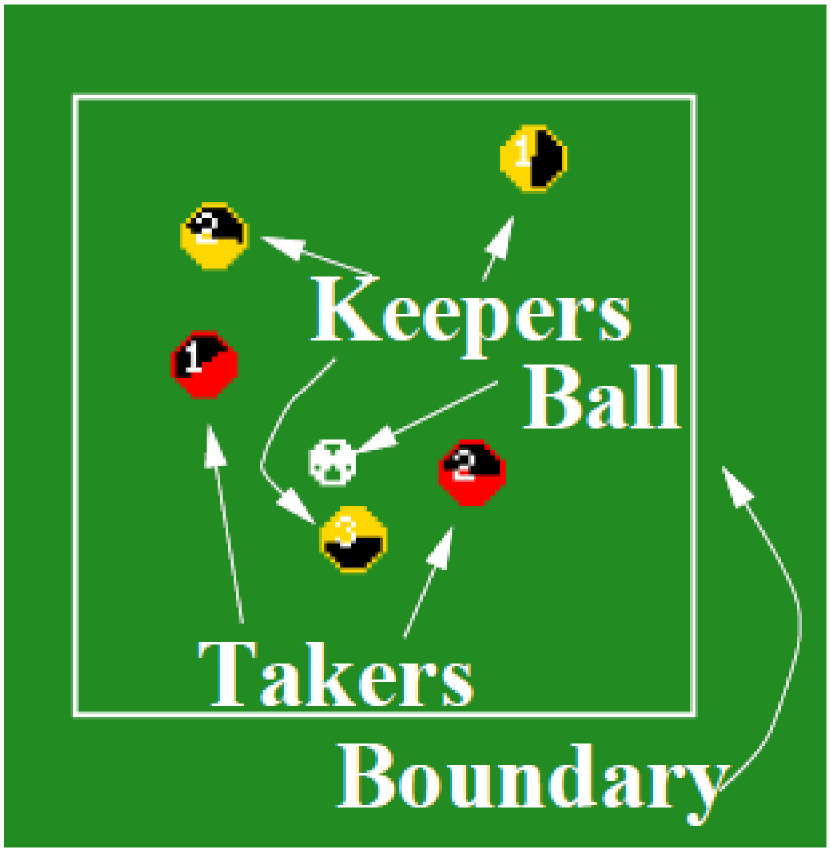

Figure 5.1. A screen shot from the middle of a 3 vs. 2 keepaway episode in a 20m x 20m region.

Keepaway is a challenging machine learning task for several reasons:

1. The state space is far too large to explore exhaustively.
2. Each agent has only partial state information.
3. The action space is continuous.
4. Multiple teammates need to learn simultaneously.

RoboCup simulated soccer has been used as the basis for successful international competitions and research challenges[46]. As described in detail in [47], it is a fully distributed, multiagent domain with both teammates and adversaries. There is hidden state, i.e., each agent has only a partial world view at any given moment. The agents also have noisy sensors and actuators, i.e., they do not perceive the world exactly as it is, nor can they affect the world exactly as intended. In addition, the perception and action cycles are asynchronous, prohibiting the traditional AI paradigm of using perceptual input to trigger actions. Communication opportunities are limited, and the agents must make decisions in real-time. These domain characteristics combine to make simulated robot soccer a realistic and challenging domain.

For the keepaway task, an omniscient coach agent manages the play, ending episodes when a taker gains possession of the ball or when the ball goes outside of the region. At the beginning of each episode, the coach resets the location of the ball and of the players semi-randomly within the region of play. The takers all start in one corner (bottom left). Three randomly chosen keepers are placed one in each of the three remaining corners, and any keepers beyond three are placed in the center of the region. The ball is initially placed next to the keeper in the top left corner.

As described in [48], keepaway problem maps fairly directly onto the discrete-time, episodic, reinforcement-learning framework. Agents in the RoboCup simulator receive visual perceptions every 150 msec indicating the relative distance and angle to visible objects in the world, such as the ball and other agents. They may execute a parameterized primitive action such as *turn(angle)*, *dash(power)*, or *kick(power,angle)* every 500 msec (unlike 100msec in the work described in [48]). This 500msec in between actions is assumed to be enough span for a human to give feedback for an action taken by the agent. As a way of incorporating domain knowledge, the agents choose not from the simulator's primitive actions, but from higher level macro-actions based closely on skills which include:

1. **HoldBall():** Remain stationary while keeping possession of the ball in a position that is as far away from the opponents as possible.
2. **PassBall(k):** Kick the ball directly towards keeper k.
3. **GetOpen():** Move to a position that is free from opponents and open for a pass from the balls current position .
4. **GoToBall():** Intercept a moving ball or move directly towards a stationary ball.
5. **BlockPass(k):** Move to a position between the keeper with the ball and keeper k.

An episode consists of a sequence of states, actions, and rewards selected and occurring at the macro-action boundaries:

$$s_0, a_0, r_1, s_1, ..., s_i, a_i, r_{i+1}, s_{i+1}, ..., r_j, s_j$$

where action a_i is chosen based on some, presumably incomplete, perception of state s_i, and s_j is the terminal state in which the takers have possession or the ball has gone out of bounds. The reward r_i is set to the the number of primitive time steps that elapsed while following action a_{i-1}. Thus $r_i = t_i - t_{i-1}$. The keepers' goal at each learning step is to choose an action such that the remainder of the episode will be as long as possible, and thus to maximize total reward.

5.1.1 The Keepers

The keepers' learning process searches a constrained policy space characterized only by the choice of action when in possession of the ball [48]. Examples of policies within this space are provided by our benchmark policies:

1. **Random:** Choose randomly among the n macro-actions, each with probability 1/n.

2. **Hold:** Always choose HoldBall macro action

3. **Hand-coded:** A hand-coded policy that selects from among the n macro-actions based on an intuitive mapping from the same state features that are used for learning.

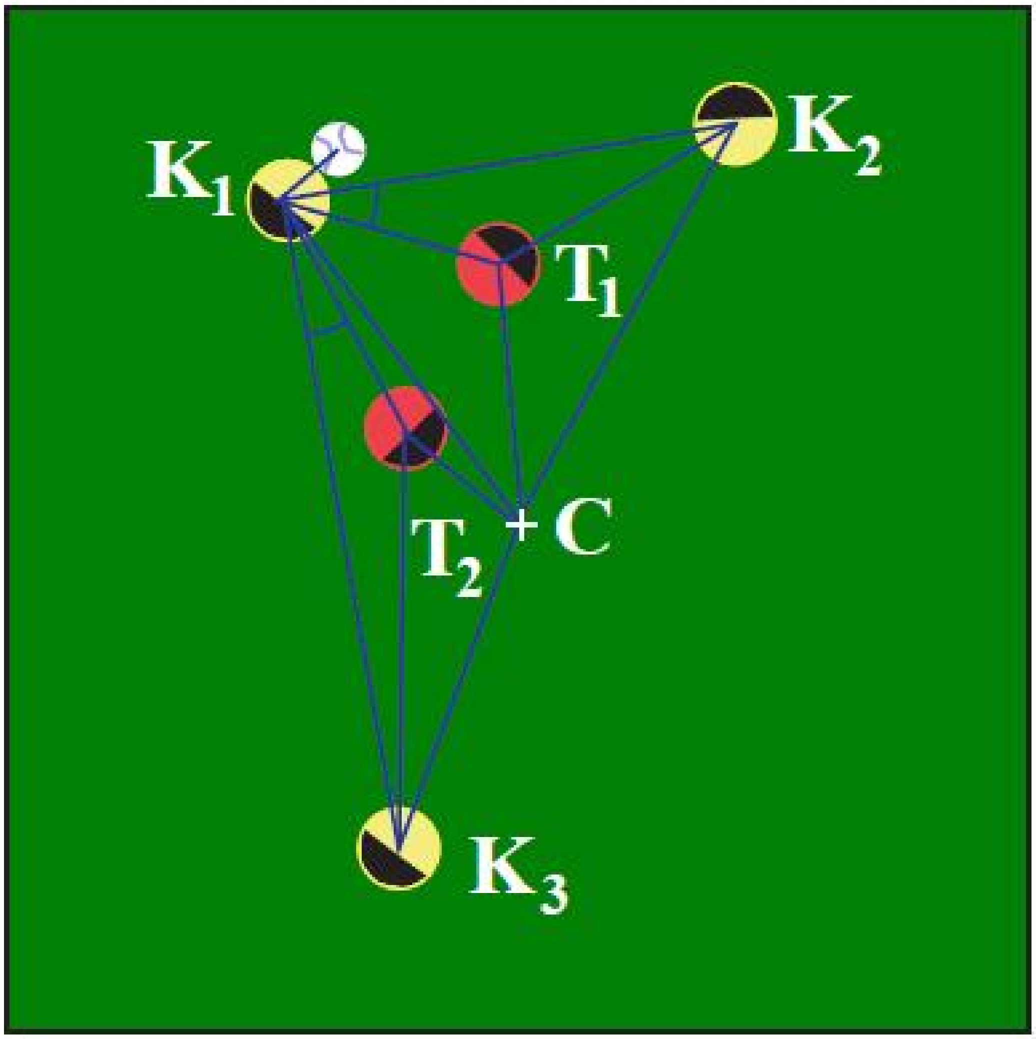

Figure 5.2. The state variables used for learning with 3 keepers and 2 takers. Keepers and takers are numbered by increasing distance from K1, the keeper with the ball. The 13 lines and angles show the complete set of state variables.

Figure 5.2 illustrates the representation of states used by the keepers. With 3 keepers and 2 takers, we use the following 13 state variables(see Figure 5.2):

1. dist(K_1,C); dist(K_2,C); dist(K_3,C);

2. dist(T_1,C); dist(T_2,C);

3. dist(K_1, K_2); dist(K_1, K_3);

4. dist(K_1, T_1); dist(K_1, T_2);

5. Min(dist(K_2, T_1), dist(K_2, T_2));

6. Min(dist(K_3, T_1), dist(K_3, T_2));

7. Min(ang(K_2, K_1, T_1), ang(K_2, K_1, T_2));

8. Min(ang(K_3, K_1, T_1), ang(K_3, K_1, T_2)).

This list generalizes naturally to additional keepers and takers, leading to a linear growth in the number of state variables.

5.1.2 **The Takers**

The takers are relatively simple, choosing only macro-actions of minimum duration (one step, or as few as possible given server misses) that exactly mirror low-level skills. When a taker has the ball, it tries to maintain possession by invoking *HoldBall()* for a step. Otherwise, it chooses an action that invokes one of *GoToBall()*, *BlockPass(K_2)*, *BlockPass(K_3)*, . . . , *BlockPass(K_n)* for one step or as few steps as permitted by the server. In case no keeper has the ball (e.g., during a pass), K_1 is defined here as the keeper predicted to next gain possession of the ball. We define the following three policies as taker benchmarks, characterized by their behavior when not in possession:

1. **Random-T**: Choose randomly from the n macro-actions, each with probability 1/n.

2. **All-to-ball**: Always choose the GoToBall action.

3. **Hand-coded-T**:

   If no other taker can get to the ball faster than this taker, or this taker is the closest or second closest taker to the ball: choose the GoToBall action;

   Else let k be the keeper with the largest angle with vertex at the ball that is clear of takers: choose the *BlockPass(k)* action.

Note that the All-to-ball and Hand-coded-T policies are equivalent when there are only two takers, since Hand-coded-T specifies that the two closest takers at any given time should go to the ball.

The state variables of the takers are similar to those of the keepers. As before, C is the center of the region. T_1 is the taker that is computing the state variables, and $T_2 - T_m$ are the other takers ordered by increasing distance from K_1. $K_i$mid is the midpoint of the line segment connecting K_1 and K_i for $i \in [2, n]$ and the K_i are ordered based on increasing distance of K_imid from T_1. That is, $\forall i, j$ s.t. $2 \leq i < j$, dist$(T_1, K_imid) \leq dist(T_1, K_jmid)$. With 3 keepers and 2 takers, we use the following 16 state variables:

1. dist(K_1,C); dist(K_2,C); dist(K_3,C);

2. dist(T_1,C); dist(T_2,C);

3. dist$(K_1, K_2); dist(K_1, K_3)$

4. dist$(K_1, T_1); dist(K_1, T_2);$

5. dist$(T_1, K_2mid); dist(T_1, K_3mid);$

6. dist$(K_2mid, T_2); dist(K_3mid, T_2);$

7. ang$(K_2, K_1, T_2); ang(K_3, K_1, T_2)$;

8. number of takers closer to the ball than T_1.

Once again, this list generalizes naturally to different numbers of keepers and takers.

In our experiments, we evaluate the agent in Keepaway soccer domain with 3 keepers and 2 takers. Sarsa(λ) reinforcement learning algorithm is used to get environmental feedback.

5.2 Reinforcement Learning When Applied To The Keepaway Soccer Domain

We use the SMDP(Semi Markov Decision Process) version of Sarsa(λ) reinforcement learning algorithm for training our keepers. This part of the work, where we train the keepers with just the environmental feedback, is based on the work described in [48]. In the next section, we introduce Sarsa(λ) before presenting the full details of the learning algorithm used to train the keepers using just environmental feedback.

5.3 Sarsa(λ)

As stated in [48], Sarsa(λ) is an on-policy learning method, i.e., the learning procedure estimates Q(s, a), the value of executing action a from state s, subject to the current policy being executed by the agent. Meanwhile, the agent continually updates the policy according to the changing estimates of Q(s, a).

In its basic form, Sarsa(λ) is defined as follows [51]:

Algorithm 5 The Sarsa(λ) Algorithm

```
1: Initialize Q(s,a) arbitrarily and e(s,a)=0 for all s,a
2: FOReach episode
3: Initialize s
4: Choose a from s using policy derived from Q
5: **for** each step of episode **do**
6:     Take action a, observe reward r, and the new state s'
7:     Choose a' from s' using policy derived from Q
8:     $\delta \longleftarrow$ r + $\gamma$Q(s',a')-Q(s,a)
9:     $e(s,a) \longleftarrow e(s,a) + 1$
10:    For all s,a: $Q(s,a) \longleftarrow Q(s,a) + \alpha\delta$e(s,a) $e(s,a) \longleftarrow \gamma\lambda$e(s,a)
11:    $s \leftarrow s'$; $a \leftarrow a'$
12: **until** s is terminal
```

Here, α is a learning rate parameter and is a discount factor governing the weight placed on future, as opposed to immediate rewards. The values in e(s, a), known as eligibility traces, store the credit that past action choices should receive for current rewards. The parameter λ governs how much credit is delivered back to them. A typical policy derived from Q, and the one we use in this thesis is an ϵ-greedy policy in which a random action is selected with probability ϵ, and otherwise, the action with maximum Q-value Q(s, a) from state s is selected. In the Keepaway Soccer domain, the simulator retains the control and occasionally presents state perceptions and action choices to the agent, as stated in [48]. This alternate orientation requires a different perspective on the standard algorithm. We need to specify three routines: 1) RLstartEpisode, to be run by the player on the first time step in each episode in which it chooses a macro-action, 2) RLstep, to be run on each SMDP step, and 3) RLendEpisode, to be run when an episode ends. These three routines are described in detail in Section 5.5

5.4 **Function Approximation**

The basic Sarsa(λ) algorithm assumes that each action can be tried in each state infinitely often so as to fully and accurately populate the table of Q-values. A key challenge for applying RL in environments with large state spaces is to be able to generalize the state representation so as to make learning work in practice despite a relatively sparse sample of the state space. In particular, in keepaway we cannot expect the agent to directly experience all possible sets of values of the variables depicted in Figure 5.2. Rather, the agent needs to learn to act in new situations, based on limited experiences. To do so, the table of Q-values must be approximated using some representation with fewer free variables than there are states, a technique commonly known as function approximation.

Here we use general tile coding to specify how the feature sets, F_a, are used for learning. Tile coding allows us to take arbitrary groups of continuous state variables and lay infinite, axis-parallel tilings over them (Figure 5.3). The tiles containing the current state in each tiling together make up a feature set F_a, with each action indexing the tilings differently. The tilings are formally infinite in extent, but in our case, all state variables are in fact bounded. Nevertheless, the number of possible tiles is extremely large, only a relatively few of which are ever visited (in our case about 10,000). Thus the primary memory vector, $\vec{\theta}$, and the eligibility trace vector $\vec{e}$ have only this many nonzero elements. Using open-addressed hash-coding only these nonzero elements need to be stored.

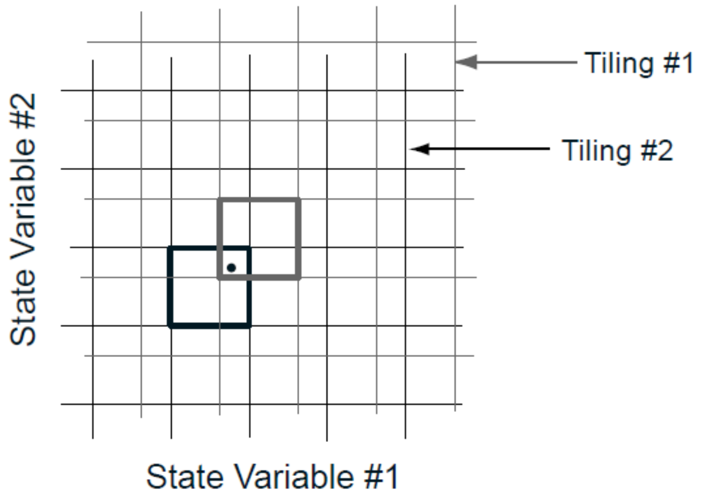

Figure 5.3. TileCoding

An advantage of tile coding is that it allows us ultimately to learn weights associated with discrete, binary features, thus eliminating issues of scaling among features of different types. The most straightforward way to get binary features is to break the state space into discrete bins. However, doing so can lead to over-generalization based on the fact that points in the same bin are required to have the same value and under-generalization due to the fact that points in different bins, no matter how close, have unrelated values. By overlaying multiple tilings it is possible to achieve quick generalization while maintaining the ability to learn fine distinctions [48].

In our experiments we primarily used single-dimensional tilings, i.e., simple stripes or intervals along each state variable individually. For each variable, 32 tilings were overlaid, each offset from the others by 1/32 of a tile width. In each tiling, the current state is in exactly one tile. The set of all these active tiles, one per tiling and 32 per state variable, is what makes up the F_a vectors. In the 3v2 case, there are 416 tiles in each F_a because there are thirteen state variables making thirteen single-variable groups, or $13\times32 = 416$ total. For each state variable, we specified the width of its tiles based on the width of generalization that we desired. For example, distances were given widths of about 3.0 meters, whereas angles were given widths of about 10.0 degrees. The choice of state variables, widths, groupings, and so on, was done based on existing work [48].

5.5 Algorithm Description

In this section, we present the full details of our approach as well as the parameter values we chose based on [48]. The pseudocode for the three top-level subroutines, RLstartEpisode, RLstep, and RLendEpisode is shown in Figure 5.4.

5.5.1 RLstartEpisode

RLstartEpisode is run by each player on the first time step in each episode in which it chooses a macro-action. In line 1, we iterate over all actions available in the current state. For each action, and for each tiling of each state variable, we find the set of tiles, F_a, activated in the current state (line 2). Next, in line 3, the action value for action a in the current state is calculated as the sum of the weights of the tiles in F_a. We then choose an action from the set of available macro-actions by following an ϵ-greedy policy (line 4). The chosen action is stored in LastAction and the current time is stored in LastActionTime (lines 4-5). In line 6, the eligibility trace vector is cleared. Finally, in lines 7-8, the eligibility traces for each active tile of the selected action are set to 1, allowing the weights of these tiles to receive learning updates in the following step.

5.5.2 RLstep

RLstep is run on each SMDP step (i.e., only when some keeper has the ball). First, in line 9, the reward for the previous SMDP step is computed as the number of time steps since the macro-action began execution. Second, in line 10, we begin to calculate the error in our action value estimates by computing the difference between r, the reward we received, and QLastAction, the expected return of the previous SMDP step. Next, in lines 11-13, we find the active tiles and use their weights to compute the action values for each action in the current state. In lines 14-15, the next action is selected as in RLstartEpisode. In line 16, we finish our calculation of the error that began on line 10. Here, we add the new $Q_{LastAction}$, the expected return of choosing action *LastAction* in the current state. Next, in line 17, we adjust the weights by the learning factor α times our error estimate δ, for tiles with non-zero eligibility traces. Since the weights have changed, in line 18, we must recalculate $Q_{LastAction}$. In line 19, the eligibility traces are decayed. Note that the traces decay only on SMDP time steps. In effect, we are using variable λ [51] and setting = 1 for the missing time steps. In lines 20-25, the traces for the chosen action are set to 1, and the traces for the remaining available actions are cleared. Note that we do not clear the traces for actions that are not in A^s because they do not apply in this state. This scheme, known as replacing traces, is one reasonable way to handle eligibility traces for SMDPs.

5.5.3 RLendEpisode

RLendEpisode is run when an episode ends. First, in line 26, we calculate the reward for the last SMDP step. Next, in line 27, we calculate the error δ. There is no need to add the expected return of the current state since this value is defined to be 0 for terminating states. In line 28, we perform the final weight update for this episode.

For the results described in this thesis, we used the following values of the scalar parameters: $\alpha = 0.125$, $\epsilon = 0.01$, and $\lambda = 0$. The parameter values were chosen based on documentation of prior experiments conducted in the domain [48]

5.6 Experimental Results

All our experiments are run considering 3v2 keepaway Soccer in a 20×20 region. For the takers, we used the Hand-coded policy (note that with just 2 takers, this policy is identical to All-to-ball). The keepers are trained in three different conditions

1. Keepers are trained in the presence of just human feedback.
2. Keepers learn from just the environmental feedback.
3. Keepers are trained by giving both environmental and human feedback.

When evaluating agents(keepers), we considered two different types of function approximation

1. Linear: Here we considered weighted linear combination of both the feedbacks (Equation 6) similar to that in case of Tetris Domain.

2. Exponential: Here we considered the exponential functions of human and environmental feedbacks. It can be mathematically written as

$$a = argmax_a[R(1 + H^{weight})] \tag{5.1}$$

where the weight is the trust factor(as described in Section 3.1).

Reward Propagation Assumption: The human feedback given is not instantaneous. The feedback given at time t is propagated over a duration of 500ms. Unlike Tetris, Keepaway soccer domain require actions at a much faster rate. Thus a human trainer cannot accurately label individual state-action pairs in real time and a credit assignment scheme that distributes the reinforcement across recent time steps is required. Since it is difficult to estimate the exact time or event that the human is responding to, the human feedback is assumed to be in response to a set of prior (and possibly subsequent) time steps. Based on the study described in [52], we choose to model the credit assignment as a gamma distribution function (Figure 5.5). Specifically, let time 0 be when a reinforcement h is given and let f(x) be the corresponding gamma distribution used for credit assignment. For two consecutive time steps t and t' that occurred before reinforcement, the credit is computed as: h$\times \int_{t'}^{t} f(x)dx$

Figure 5.6 shows the performance of keepers trained with just the human feedback. Based on the agent's recent performance, the trainer provides a reward that is mapped to a function and hence a scalar value of specific time steps. The agent's goal is to take an action so as to maximize the human reward and hence increase the episode duration (time for which the keepers can keep the ball with them before takers take the possession of the ball or the ball goes out of the 20$\times$20 field). It can be mathematically written as $action = argmax_a \hat{H}(s,a)$. The y-axis is the average time that the keepers are able to keep the ball from the takers (average episode duration) while the x-axis is the number of episodes trained.

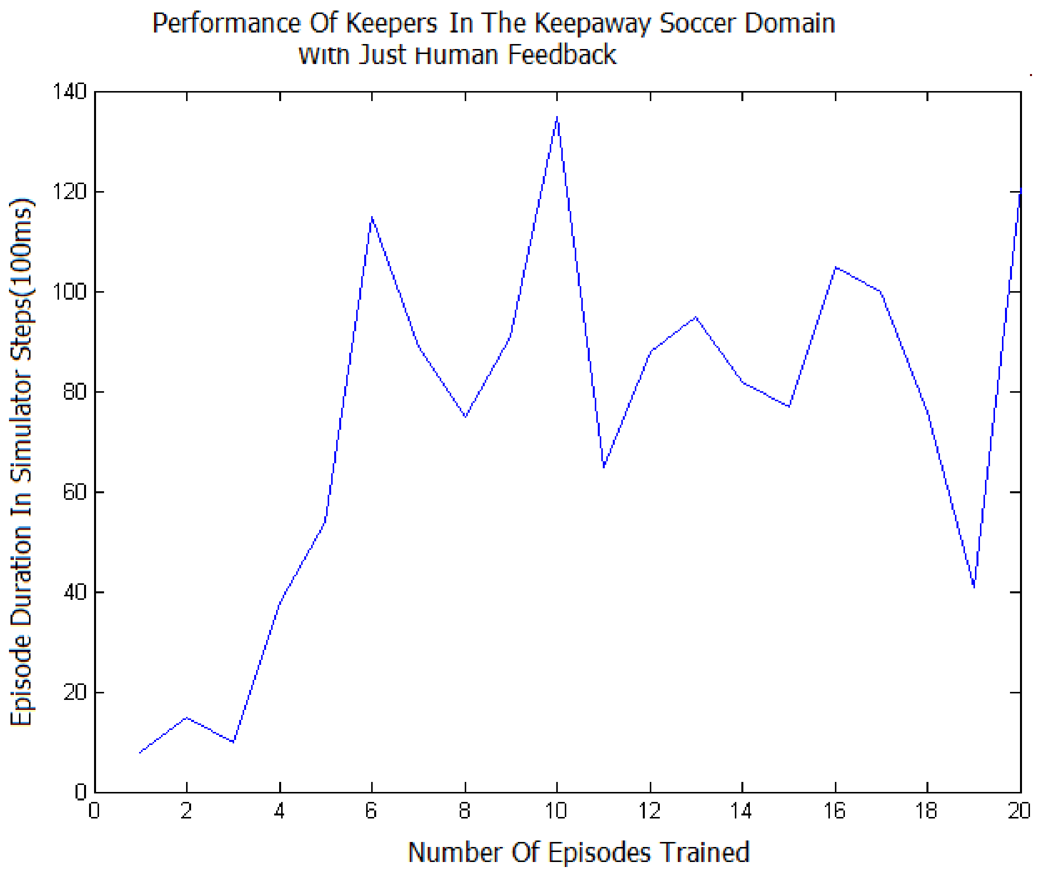

Figure 5.6. Performance of Keepers trained with just human feedback.

Figure 5.7 shows experimental results of the performance of keepers when trained with both human and environmental feedbacks using the weighted linear combination scheme of Equation 6, in comparison to the exponential combination scheme of Equation 9. The comparison also includes the cases where the keepers learn from just the environmental feedback or human feedback. Experiments were conducted separately with and without the use of gamma distribution for credit assignment. Similar to the results obtained in Tetris domain, it is observed that the keepers perform much better when trained with both human and environmental feedbacks compared to that of training with just the environmental feedback. It is also observed that the gamma distribution based credit assignment further improves the performance. The best performance is seen to be achieved by training the agents using the weighted linear combination of Human and environmental feedbacks using gamma distribution for credit assignment.

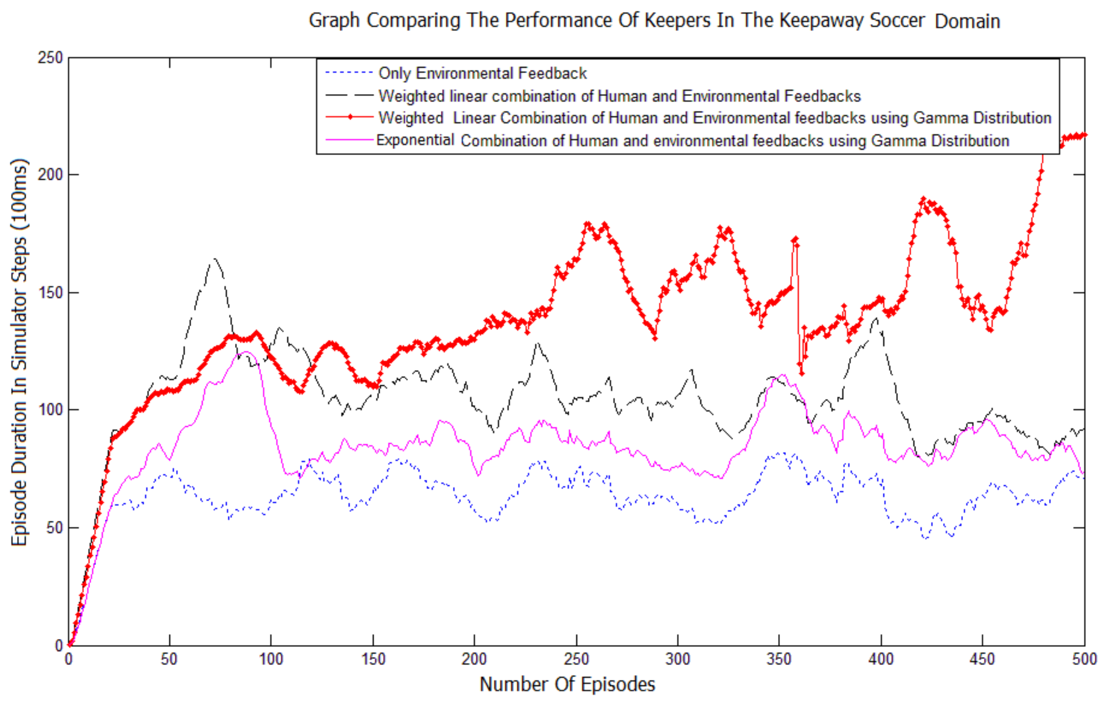

Figure 5.7. A graph comparing the performance of keepers trained with “just the environmental feedback” and “both human and environmental feedbacks”

```
RLstartEpisode:
1   For all $a \in \mathcal{A}^s$:
2       $\mathcal{F}_a \leftarrow$ set of tiles for $a, s$
3       $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
4   $LastAction \leftarrow \begin{cases} \arg\max_a Q_a & \text{w/prob. } 1 - \epsilon \\ \text{random action} & \text{w/prob. } \epsilon \end{cases}$
5   $LastActionTime \leftarrow CurrentTime$
6   $\vec{e} = \vec{0}$
7   For all $i \in \mathcal{F}_{LastAction}$:
8       $e(i) \leftarrow 1$

RLstep:
9   $r \leftarrow CurrentTime - LastActionTime$
10  $\delta \leftarrow r - Q_{LastAction}$
11  For all $a \in \mathcal{A}^s$:
12      $\mathcal{F}_a \leftarrow$ set of tiles for $a, s$
13      $Q_a \leftarrow \sum_{i \in \mathcal{F}_a} \theta(i)$
14  $LastAction \leftarrow \begin{cases} \arg\max_a Q_a & \text{w/prob. } 1 - \epsilon \\ \text{random action} & \text{w/prob. } \epsilon \end{cases}$
15  $LastActionTime \leftarrow CurrentTime$
16  $\delta \leftarrow \delta + Q_{LastAction}$
17  $\vec{\theta} \leftarrow \vec{\theta} + \alpha \, \delta \, \vec{e}$
18  $Q_{LastAction} \leftarrow \sum_{i \in \mathcal{F}_{LastAction}} \theta(i)$
19  $\vec{e} \leftarrow \lambda \vec{e}$
20  If player acting in state $s$:
21      For all $a \in \mathcal{A}^s$ s.t. $a \neq LastAction$:
22          For all $i \in \mathcal{F}_a$:
23              $e(i) \leftarrow 0$
24      For all $i \in \mathcal{F}_{LastAction}$:
25          $e(i) \leftarrow 1$

RLendEpisode:
26  $r \leftarrow CurrentTime - LastActionTime$
27  $\delta \leftarrow r - Q_{LastAction}$
28  $\vec{\theta} \leftarrow \vec{\theta} + \alpha \, \delta \, \vec{e}$
```

Figure 5.4. The three main routines of our $Sarsa(\lambda)$ implementation presented for a keeper. A taker has the sign of the reward, r, reversed. The set of macro-actions available, $A^s \subseteq A$, depends on the current state, s. For example, the keepers not in possession of the ball must select the Receive action, whereas the keeper with the ball chooses from among HoldBall and PasskThenReceive [48].

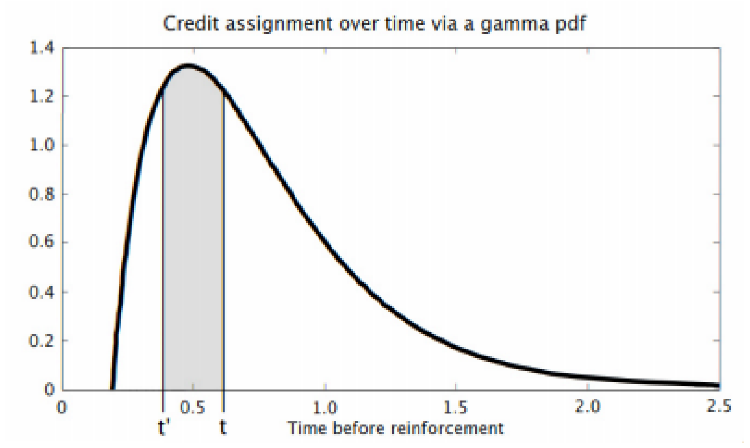

Figure 5.5. Probability density function f(x) for a gamma(2.0, 0.28) distribution. Reinforcement signal h is received at time 0. If t and t'are times of consecutive time steps, credit for the time step at t is h$\times \int_{t'}^{t} f(x)dx$

CHAPTER 6
CONCLUSION

For a human with no technical background, training an agent poses a major challenge in the field of human agent interaction. One of the main aims of this thesis is to find more reliable ways of combining human and environmental feedbacks. The experimental results obtained in the Tetris and Keepaway Soccer domain show that better performance can be achieved by effectively combining human feedback and environmental feedback, in comparison to using each of these feedbacks individually. In addition, the agent can learn smoothly both in the presence and in the absence of the human trainer. This research would be of great help in training agents in domains where continuous human feedback is not possible but effective/reliable learning by the agent is essential.

BIBLIOGRAPHY

[1] W. Bradley Knox and Peter Stone, *Tamer: Training an agent manually via evaluative reinforcement*. In IEEE 7th International Conference on Development and Learning, 2008.

[2] Jan Ramon, Kurt Driessens and Thomas Gartner, *Graph kernels and gaussian processes for relational reinforcement learning*. In proceedings of Inductive Logic Programming, 13th International Conference, ILP 2003, pp. 146–163, 2003.

[3] Istvan Szita and Andras Lorincz, *Learning tetris using noisy cross entropy method*. Neural Computation, vol. 18, no. 12, pp. 2936–2941, 2006.

[4] Jan Peters and Stefan Schaal, *Policy gradient methods for robotics*. In proceedings of the IEEE International Conference on Intelligent Robots and Systems, 2006.

[5] Richard S. Sutton and Andrew G. Barto, *Reinforcement Learning:An Introduction*. Cambridge, MA: MIT Press, 1998.

[6] W. Bradley Knox and Peter Stone. *Combining Manual Feedback with subsequent MDP Reward Signals for Reinforcement Learning*. In Proceedings of 9th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2010),2010.

[7] Andrea L.Thomaz and Cynthia Breazeal, *Assymmetric Interpretations Of Positive And Negative HumanFeedback For Social Learning Agents*. In 16th IEEE International Symposium on Robot and Human interactive Communication, pp. 720–725, 2007.

[8] Michael Lewis, Katia Sycara and Illah Nourbakshsh, *Developing a Testbed for Studying Human-Robot Interaction in Urban Search and Rescue*. In proceedings of the 10th International Conference on Human Computer Interaction (HCII'03), Crete, Greece, June 22-27, 2003.

[9] H.Kitano, S. Tadokora, I. Noda, H. Matsubara, T. Takahashi, A. Shinjoh, S. Shimada, *Robocup Rescue: Search and Rescue in Large-Scale Disasters as a Domain for Autonomous Agents Research,Proc.1999*, IEEE Intl.Conf. on Systems,Man and Cybernetics,vol. 6, pp. 739–743, 1999.

[10] Andreal L. Thomaz, Guy Hoffman and Cynthia Breazeal, *Reinforcement Learning with Human Teachers: Understanding How to Teach Robots*. in Proceedings

of the 15th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN). Univ. of Hertfordshire, Hatfield, UK. pp. 352–357, 2006.

[11] Y. Wang, M. Huber, V.N. Papudesi, and D.J. Cook, *User-Guided Reinforcement Learning of Robot Assistive Tasks for an Intelligent Environment.* in International Conference on Intelligent Robots and Systems 1, pp. 424–429, 2003.

[12] R.C. Arkin, *Behavior-Based Robotics.* Cambridge, MA, USA: The MIT Press,1998.

[13] C. Breazeal, D. Buchsbaum, J. Gray, D. Gatenby, and B. Blumberg, *Learning from and about others: Towards using imitiation to bootstrap the social understanding of others by robots.* in Artificial Life Journal, (L. Rocha and F. Almedia e Costa, eds.), Cambridge, MA, USA: MIT Press, 2005.

[14] C. Breazeal, G. Hoffman, and A. Lockerd, *Teaching and working with robots as a collaboration.* in Third International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS), ACM Press, 2004.

[15] O. Brock, A. Fagy, R. Grupen, R. Platt, M. Rosenstein, and J. Sweeny, *framework for learning and control in intelligent humanoid robots.* International Journal of Humanoid Robotics, vol. 2, no. 3, 2005.

[16] R.A. Brooks, *A robust layered control system for a mobile robot.* IEEE Journal of Robotics and Automation, vol. 2, pp. 14–23, 1986.

[17] J.A. Fails and D.R. Olsen Jr., *Interactive machine learning.* in Proceedings of the 8th International Conference on Intelligent User Interfaces, Miami, FL, USA: ACM Press, 2003.

[18] T. Fong and C. Thorpe, *Vehicle teleoperation interfaces.* Autonomous Robots, vol. 11, no. 1, pp. 9–18, 2001.

[19] D.W. Franklin, T.W. Milner, and M.Kawato, *Single trial learning of external dynamics: What can the brain teach us about learning mechanisms in Brain-Inspired IT III.* Invited and Selected Papers of the 3rd International Conference on Brain-Inspired Information Technology, pp. 67-70, 2007.

[20] R. Gockley, J. Forlizzi, and R. Simmons, *Natural person-following behavior for social robots.* in ACM/IEEE International Conference on Human-Robot Interaction, 2007.

[21] J.P. Jinsul Kim, Y.K. Hwant, and M. Lee, *Advanced grasp planning for handover operation between human and robot: Three handover methods in esteem etiquettes using dual arms and hands of home-service robot.* in International Conference on Autonomous Robots and Agents, 2004.

[22] H.J. Kim, H. Lee, and C. Kim, *Autonomous behavior design for robotic appliance.* in ACM/IEEE International Conference on Human-Robot Interaction, 2007.

[23] M.N. Nicolescu and M.J. Mataric, *Learning and interacting in humanrobot domains.* IEEE Transactions on Systems, Man, and Cybernetics, Part A, vol. 31, no. 5, pp. 419-430, 2001.

[24] B. Ogden and K. Dautenhahn, *Robotic etiquette: Structured interations in humans and robots.* in Symposium on Intelligent Robotic Systems, 2000.

[25] E. Pacchierotti, H.I. Christensen, and P. Jensfelt, *Evaluation of passing distance for social robots.* in IEEE International Workshop on Robot and Human Interactive Communication, 2006.

[26] D. Roy, *Learning words and syntax for a visual description task.* Computer Speech and Language, vol. 16, no. 3, pp. 353-385, 2002.

[27] N. Roy, G. Baltus, D. Fox, F. Gemperle, J. Goetz, T. Hirsch, D. Magaritis, M. Montemerlo, J. Pineau, N. Roy, J. Schulte, and S. Thrun, *Towards personal service robots for the elderly.* in Workshop on Interactive Robotics and Entertainment, Pittsburgh, PA, USA, 2000.

[28] J. Saunders, C. Nehaniv, and K. Dautenhahn, *Using self-imitation to direct learning.* in 15th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), Hatfield, UK: University of Hertfordshire, 2006.

[29] J. Scholtz, M. Theofanos, and B. Antonishek, *Theory and evaluation of human robot interactions.* in 36th International Conference on Systems Sciences, Hawaii: IEEE, 2002.

[30] J.G. Trafton, A.C. Schultz, D. Perznowski, M.D. Bugajska, W. Adams, N. L. Cassimatis, and D. P. Brock, *Children and robots learning to play hide and seek.* in First ACM International Conference on Human-Robot Interaction, Salt Lake City, UT, USA: ACM Press, 2006.

[31] M.L. Walters, K. Dautenhahn, S.N. Woods, and K.L. Koay *Robot etiquette: Results from user studies involving a fetch and carry task.* in ACM/IEEE International Conference on Human-Robot Interaction, 2007.

[32] F. Yamaoka, T. Kanda, H. Ishiguro, and N. Hagita, *Lifelike behavior of communication robots based on developmental psychology findings.* in IEEERAS International Conference on Humanoid Robots, 2005.

[33] Z. Zenn Bien and H.E. Lee, *Effective learning system techniques for human robot interaction in service environment.* Knowledge-Based Systems, vol. 20,pp. 439-456, 2007.

[34] C.G. Atkenson and S. Schoal, *Robot learning from demonstration*, in international conference on machine learning, 1997.

[35] J.R. Chen, *Constructing task-level assembly strategies in robot programming by demonstration*, International Journal Of Robotics Research, vol. 24, no. 12, pp. 1073-1085, 2005.

[36] Y.Demiris and A.Billad, eds., IEEE *transactions on systems, man and cybernetics, Part B.Special issue on Robot Learning by Observation, Demonstration and Imitation*, 2007.

[37] R. Dillman, M. Ehrenmann, P. Steinhaus, O. Rogolla, and R. Zoellner, *Human friendly programming of humanoid robots* , 2002.

[38] M. Kaiser and R. Dillman, *Building elementary robot skills from human demonstration*, in IEEE conference on Robotics and Automation, Orlando, FL, vol. 24,no. 12, pp. 1073–1085, 2006.

[39] H. Onda, K. Kitagaki and T. Suehiro *Visualization and simulation of sensory events as a representation of state for state based teaching by demonstration in VR*, in IEEE, Robotics Society of Japan International Conference on Intelligent Robots and Systems, 2005.

[40] M. Asada, K.F. MacDoman, H. Ishiguro and Y. Kuniyoshi, *Cognitive developmental robotics as a new paradigm for the design of humanoid robots*, Robotics and Autonomous Systems, vol. 37, no. 2-3, pp. 185–193, 2001.

[41] A. Billard and K. Dautenhahn, *Experiments in social robotics:Grounding and use of communication in autonomous agents*, Adaptive Behaviour, vol. 7,no. 3–4, pp. 415–438, 1999.

[42] J. Nekanishi, J. Morimoto, G. Endo, G. Cheng, S. Schaal and M. Kawato, *Learning from demonstration and adaption of biped locomotion*, Robotics and Autonomous Systems, vol. 47,no. 2–3, pp. 79–91, 2004.

[43] N.S. Pollard, J.K. Hodgins, M.J. Riley and C.G. Atkenson, *Adapting human moion for the control of a humanoid robot*, in IEEE conference on Robotics and Automation, 2002.

[44] J.M. Bradshaw, P.J. Feltovich, H. Jung, S. Kulkarni, W. Taysom and A. Uszok, *Dimensions of adjustable autonomy and mixed-initiative interaction*, in Agents and Computational Autonomy: Potential, Risks and Solutions,(M.Nickles, M.Rovatos, and G.Weiss, eds.), pp. 17–39,Berlin/Heidelberg:Springer, 2004.

[45] J. Saunder, C.L. Nehaniv and K. Dautenhahn, *Teaching robots by moulding behaviour and scaffolding the environment*, in ACM SIGCHI/SIGART International Conference on Human-Robot Interaction, 2006.

[46] H. Kitano, M. Tambe, P. Stone, M. Veloso, S. Coradeschi, E. Osawa, H. Matsubara, I. Noda and M. Asada, *The robocup synthetic agent challenge.* In Proceedings of the Fifteenth International Joint Conference on Artificial Intelligence(IJCAI97), pp. 24–49, San Francisco, CA, 1997.

[47] P. Stone, Richard R.Sutton and Satinder Singh *Reinfocement Learning for 3vs. 2 Keepaway.* In P. Stone, T. Balch and G. Kraetszchmar (Eds.), RoboCup-2000: Robot soccer world cup IV, pp. 249–258. Berlin: Springer Verlag, 2001.

[48] P. Stone, Richard R. Sutton and Gregory Kuhlmann *Reinforcement Learning for RoboCup Soccer Keepaway.* Adaptive Behavior, vol. 13, no. 3, pp. 165–188, 2005.

[49] P. Stone, W. Knox *Interactively Shaping Agents via Human Reinforcement: The TAMER framework* In Proceedings of The Fifth International Conference on Knowledge Capture. 2009.

[50] Michael A. Goodrich and Alan C. Schultz *HumanRobot Interaction: A Survey.* Foundations and Trends in Human-Computer Interaction, vol. 1, no. 3, pp. 203–275, 2007.

[51] Richard S. Sutton and Andrew G. Barto *Reinforcement Learning: An Introduction.* Cambridge, MA: MIT Press. 1998.

[52] W. Bradley Knox, Ian Fasel and Peter Stone. *Design Principles for Creating Human-Shapable Agents.* AAAI Spring 2009 Symposium on Agents that Learn from Human Teachers. 2009.