Fast and Robust Incremental Action Prediction for Interactive Agents

Jonathan Dinerstein
jondinerstein@yahoo.com

Dan Ventura
ventura@cs.byu.edu

Parris K. Egbert
egbert@cs.byu.edu

Brigham Young University
Computer Science Department
3366 TMCB, Provo, Utah 84602, USA

Phone: 1-801-422-4029 Fax: 1-801-422-0169

Abstract

The ability for a given agent to adapt on-line to better interact with another agent is a difficult and important problem. This problem becomes even more difficult when the agent to interact with is a human, since humans learn quickly and behave non-deterministically. In this paper we present a novel method whereby an agent can incrementally learn to predict the actions of another agent (even a human), and thereby can learn to better interact with that agent. We take a case-based approach, where the behavior of the other agent is learned in the form of state-action pairs. We generalize these cases either through continuous *k*-nearest neighbor, or a modified bounded minimax search. Through our case studies, our technique is empirically shown to require little storage, learn very quickly, and be fast and robust in practice. It can accurately predict actions several steps into the future. Our case studies include interactive virtual environments involving mixtures of synthetic agents and humans, with cooperative and/or competitive relationships.

Key words: autonomous agents, user modeling, agent modeling, action prediction, plan recognition.

1. Introduction

The use of intelligent software agents is becoming increasingly pervasive. This is especially true in interactive software, where one or more human users may interact with the system at any time. Examples of such agents include training simulator and computer game characters, virtual tutors, etc. However, an important limitation of most agents in interactive software is that they are usually static. In other words, the agent's behavior does not adapt on-line in response to interaction with a human user.

Effective and rapid on-line adaptation of an agent's behavior would be extremely useful for many applications. For example, consider a training simulator where a virtual character is an opponent to a human user (see figure 1). By learning on-line through interaction with the human, the character could adjust its tactics according to those of the human it is competing against and thereby become a more difficult, customized opponent.

Figure 1. A virtual character (skeleton marked with a star) tries to catch the human's avatar (other skeleton). By adapting to the human's tactics, the character can become a more challenging, personalized opponent.

One primary reason for the lack of use of on-line learning for interactive agents is that

learning from and about a human is difficult, due to non-deterministic behavior and a non-stationary policy. Also, humans learn quickly and are impatient, so an agent must also learn quickly to provide a stimulating and effective experience for the user. These issues are discussed in more detail in section 3.

In this paper, we present a novel machine learning method for fast adaptation of interactive agents. Specifically, our technique allows an agent to learn to predict the future actions of a human user (or another synthetic agent). We take an observation-based approach which operates through case-based reasoning. As the interaction proceeds, the agent records state-action pairs of the human's behavior. Each state-action is treated as a single case. The case library represents a non-deterministic mapping of states to actions. We generalize these state-action cases using either continuous *k*-nearest neighbor or a modified minimax search. These alternate approaches to generalization allow us to adjust a confidence/caution tradeoff as we see fit.

The agent can predict several steps into the future by iteratively predicting using the previously predicted state. That is, for each time step into the future, we predict each agent's next action; the predicted actions are then used to predict the next state, which is in turn used for the next iteration of action prediction. Finally, once the desired length of prediction has been computed, the agent uses this information to help in selecting its own actions. Note that our technique can learn quickly, being case-based, and can naturally handle non-determinism in the behavior it is learning. Also, old cases are deleted so that it can learn non-stationary behavior. Further, our technique can be used for either cooperative or competitive relationships between an agent and human, since our learning technique provides the agent with very general information.

We begin by surveying related work. We then examine the issues involved in on-line learning for interactive agents, and propose a set of requirements for such a learning method.

We then present our technique for rapid adaptation of interactive agents through action prediction. Finally, we conclude with our experience from three case studies. Our first two case studies are based on a computer game/simulation of a sport like rugby or American football. We chose this interaction setting since athletics is interesting for studying cooperative and competitive behavior, and is known to be a difficult problem for machine learning (Stone 2000). Our first case study is a simplified problem, whereas the second is significantly more complex. Our third case study is of Capture The Flag using the Gambots (Kaminka 2002) test bed.

2. Related Work

Our case-based action prediction technique overlaps with many topics currently of interest in AI and machine learning, such as plan recognition, learning in multi-agent systems, and agent/user modeling. While our method is novel, it was inspired by previous work which we now review. Note that our method falls within the bounds of *fictitious play* theory (Stone and Veloso 1997).

The use of AI techniques in animating virtual characters has long been popular (Reynolds 1987). By making a virtual character an autonomous agent, it can be self-animating. This topic includes all autonomous agents in interactive virtual environments, such as characters in training simulators and computer games. On-line learning for interactive virtual characters has only begun to be addressed (Evans 2002; Tomlinson and Blumberg 2002; Blumberg et al. 2002), and is currently limited to master-slave relationships and learning high-level behavioral guidelines through immediate and direct instruction and feedback from the human user. Thus these techniques do not solve the problem we are addressing.

A paper that addresses many of the same concerns we do is (Isbell et al. 2001), where an agent in a text-based distributed computer game — a "MUD" — learns to pro-actively interact with the human users in a desirable manner. Users provide feedback on the agent's behavior, guiding its learning. The agent learns through a modified form of Q-learning, constructing value tables through linear function approximation. This previous work focuses on human-agent interaction in a social context, whereas we are interested in virtual environments where the human and agent interact in a physically oriented manner.

A well-known technique for learning in multi-agent systems is *minimax-Q* (Littman 1994). This technique is a modification of Q-learning, designed for two agent zero-sum Markov games. While effective and formally sound, this technique does not address the problem of interest in this paper because minimax-Q requires too much time and too many experiences to learn (in a simple soccer environment, one million steps).

An interesting work performed in *action prediction* is (Laird 2001). In this technique, the agent attempts to predict the human's future behavior by assuming the human will behave exactly like the agent would in the same situation. This interesting and simple technique has proven effective in practice for a complex virtual environment. However, the agent does not learn about the human's behavior, so it has no ability to adapt. Note that action prediction and *plan recognition* are often considered to be the same problem.

Agent modeling has been a popular approach to learning in multi-agent systems for some time (Bui et al. 1996; Vidal and Durfee 1997; Weiss 1999). One such technique is *M** (Carmel and Markovitch 1996a; Carmel and Markovitch 1996b), a generalization of minimax. Rather than assuming the opponent will always choose the optimal action, the opponent's observed behavior is modeled using a neural net. Thus, given a state as an input, the network produces a de-

terministic prediction of the opponent's next action. While interesting, this approach is not useful for our needs, since the neural net may require too much time and too many examples to learn. Also, the neural net will produce deterministic predictions (i.e. will average conflicting state → action examples), and thus is likely to make notable mistakes with regards to human behavior which is often highly non-deterministic.

Another modeling-based technique is *Maximum Expected Utility* (MEU) (Sen and Arora 1997), also a modification of minimax. The opponent is modeled such that, for any given state, each candidate action has a probability of being selected. Thus, the expected utility of an action performed by the learning agent can be computed as the weighted sum of the rewards with respect to all the actions the opponent can select (where the weights are the probabilities). Thus, unlike *M**, this technique does not ignore worst-case scenarios and does not produce deterministic predictions. However, while this is an interesting approach, it can require massive amounts of storage, is infeasible for continuous state and/or action spaces, and the probabilities can take a long time to learn. Also, if we wish to determine long-term utility (predict far into the future), our search will likely be intractable.

In (Tran and Cohen 2002), an agent models the reputation of sellers in an electronic marketplace to more optimally engage in trade. This technique, while interesting and useful, does not address the problem of interest in this paper since it only models a high-level aspect of the other agents, rather than their actual behaviors.

One of the most well known agent modeling techniques is the *Recursive Modeling Method* (RMM) (Gmytrasiewicz and Durfee 2000; Gmytrasiewicz and Durfee 2001). In RMM, the fact that an agent knows that the other agent is learning about it is modeled explicitly. Specifically, an agent models the other agent, but then must also model how the other agent will

change due to the fact that the first agent has learned. This process can continue for some time, while each agent continues to learn models of the current behavior of the other. Thus each agent contains a recursive set of models, up to a pre-specified depth. In (Gmytrasiewicz and Durfee 2000; Gmytrasiewicz and Durfee 2001), RMM is implemented using a tree of payoff matrices. Because of this, storage and computational requirements increase exponentially, and therefore this technique may not be useful for many practical settings. Also, RMM is limited to discrete state and action spaces. Some initial work has been performed in flattening RMM into a single level learning technique (Hu and Wellman 1998), but this problem is still unresolved.

Recent work (Rovatsos et al. 2003) in agent modeling has examined *open systems*: multi-agent systems where agents enter/leave quite often, and specific agents often never return to the system. To cope with these issues, agents are classified and then each class is modeled. This is a general enough concept that it could be used with most agent modeling techniques.

Recently, *Case-Based Plan Recognition* (CBPR) (Kerkez and Cox 2003) was introduced. Unlike most previous plan recognition methods, CBPR does not require that a plan library be constructed *a priori*. State-action cases of an agent's observed behavior are recorded and then used to predict future behavior. The state-actions are expressed in first-order logic, and are generalized by finding the single case whose associated state most closely matches the query state. However, while CBPR has proven to be interesting, it has some weaknesses that make it inappropriate as a solution for the problem of interest in this paper. Most notably, CBPR is only about 10% accurate in practice for toy problems, and can only predict one time step into the future (prediction chaining is ineffectual due to low accuracy). Further, CBPR makes deterministic predictions, and it cannot handle continuous state and action spaces (which many interactive environments, such as training simulators, have).

Closely related to agent modeling is *user modeling* (Zhu, Greiner, and Haubl 2003). A human user is modeled so that software or an agent can more optimally serve him or her. This is closely related to the problem of interest in this paper. However, most existing techniques in this domain are designed for user interface adaptation, and thus model the user in a very specific/limited manner. In this paper, we are interested in autonomous agents which proactively interact with a user (according to their own goals, cooperative or competitive), and also engage in activities entirely independent from the user.

Our method is also somewhat related to agent *programming by demonstration* (Mataric 2000; Nicolescu 2003), where an agent learns how to behave from human demonstrations. In these techniques, a model is created of the human's behavior for the purpose of reproducing it to control the agent. Similarly, our method is somewhat related to *imitation learning* (Price 2002), where an agent learns by observing and imitating other agents (synthetic and/or biological). However, these techniques do not address the problem of interest in this paper.

There is need for a technique that allows an interactive agent to rapidly learn on-line to better interact with a unique human user. As discussed in section 3, this adaptation should be robust, require no explicit feedback from the human user, and learn fast enough to not tax human patience. Our contribution is a novel machine learning method that fulfills these needs.

3. The Interactive Agent Learning Problem

As mentioned previously, on-line learning for agents which interact in real-time with a human is a difficult problem. We have identified several requirements for an effective interactive agent learning technique, which are detailed below.

Requirements for an Effective Interactive Agent Learning Technique:

1. Fast learning. Human time and patience are scarce resources. Further, slow learning will not make an agent a more difficult opponent, or effective teammate, etc. Therefore, learning must occur quickly based on few experiences.

2. Learn in the presence of non-deterministic and non-stable behavior. In other words, the learning technique must be effective when the agent is interacting with a human.

3. No explicit human feedback. Requiring the human to provide feedback may be unnatural for many types of interactions, and may interrupt the flow of the interaction. Therefore, no explicit human feedback should be required.

4. Must perform at interactive rates on a PC. For the learning technique to be as widely useful as possible, it must be practical for interactive use on current PC CPUs (e.g. 2 GHz). Further, use of storage must be reasonable.

5. Support both cooperative and competitive relationships. While this is not truly a requirement, it is a desirable property so that the learning technique will be as broadly useful as possible.

4. Technique Description

We fulfill the requirements listed in section 3 by using a combination of observation- and case-based approaches to construct a case-based model of the human's behavior. This model is then used to predict the human's future actions, allowing the agent to predict the utility of its own candidate actions.

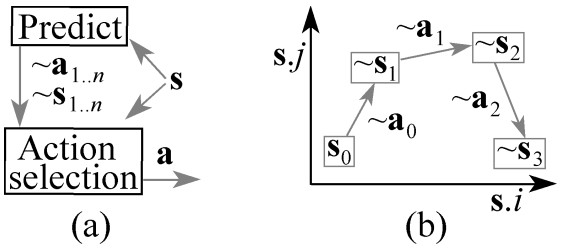

Figure 2. (a) Structure of our learning method. An *n*-step sequence of states and actions into the future is predicted, then the agent uses this information to make more optimal choices. (b) To predict the human's actions several steps into the future, we predict/compute the actions of all agents in the environment, determining a new state, and then repeat.

Our technique is summarized in figure 2a. As the interaction proceeds, the agent records state-action pairs of the human's behavior. Each state-action is treated as a single case. This case library models the human's decision making, representing a non-deterministic mapping of states to actions. We generalize these state-action cases using either continuous *k*-nearest neighbor or a bounded minimax search. These alternate approaches to generalization allow us to adjust a confidence/caution tradeoff as we see fit.

As a formal overview, the use and execution of our adaptation technique involves the following steps:

1. Formulate a compact internal representation of the state space, *S*, and a logically structured internal representation of the action space, *A*.
2. Define a state-action case library: $L = \{(\mathbf{s}, \mathbf{a})_i\}$. Partition the library into regions for fast case lookup: $P_S = \{r_i\}$ such that $r_i \cap r_j = \varnothing \leftrightarrow i \neq j$, $\cup_i r_i = S$. Initialize the library with state-action cases of generic behavior. Set time, $t := 0$.
3. As the interaction proceeds, observe and record state-action pairs. First, at time *t*, get

$(\mathbf{s}_t, \mathbf{a}_t)$. Next, determine r_j such that $(\mathbf{s}_t, \mathbf{a}_t) \in r_j$. Finally, replace $\arg\min_{(\mathbf{s},\mathbf{a})_i \in r_j}(\mathrm{M}((\mathbf{s},\mathbf{a})_i))$ with $(\mathbf{s}_t, \mathbf{a}_t)$, where M is a case usefulness metric with positive scalar weights α and β: $\mathrm{M}((\mathbf{s},\mathbf{a})_i) = -\alpha \cdot \mathrm{age} + \beta \cdot d((\mathbf{s},\mathbf{a})_i, (\mathbf{s}_t, \mathbf{a}_t))$. d is an appropriate distance metric. "age" is the number of time steps for which case i has existed in L.

4. Predict the human's (or other agent's) future behavior by generalizing state-action cases: $\sim \mathbf{a}^{\mathrm{H}}_{t+1} = f(\mathbf{s}_t, L)$. After predicting, the agent uses this information in selecting its own action to perform: $\mathbf{a}_{t+1} = g(\mathbf{s}_t, \sim \mathbf{a}^{\mathrm{H}}_{t+1})$. Increment current time, $t := t + 1$.
5. Repeat steps 3-5 for the duration of the interaction.

Note that steps 1 and 2 are performed by the designer/programmer in preparation for execution.

It is important that we predict the human's actions several steps into the future, so that our agent can maximize its utility based on non-local information. As shown in figure 2b, we do this iteratively by predicting using the previously predicted state. At each iteration, we either compute, predict, or assume the actions of all agents in the environment. The simulated actions of the predicting agent are either assumed or computed without the aid of prediction to avoid branching or recursion (although we discuss in section 4.4 how branching can be used if desired to increase prediction accuracy). In our case studies, we have found that predicting between 5 to 15 steps into the future works well, is accurate enough to be useful, and is tractable. Finally, once the desired length of prediction has been computed, the agent uses the prediction in selecting its own actions, whether it intends to cooperate with or compete against the human.

Our goals for our learning method are somewhat different than in traditional machine learning. Where most traditional techniques are designed to learn *well* from scratch, our technique is designed to learn *fast* to adapt an agent to better interact with a unique human user. As a

result, we rely somewhat on prior knowledge. Specifically, we assume that the programmer provides a reasonably compact state definition, and (if using minimax case generalization) a gradient fitness function.

4.1 State and Action Representations

Most interactive environments involving both software agents and human users are *virtual environments*: synthetic digital worlds which define the roles and possible actions of the human users and agents. Some of these virtual environments are presented to the user visually through computer graphics (e.g. training simulators, computer games, etc). In our case studies in this paper, we focus on this sort of environment. Thus the current state is a snapshot of the configuration of the virtual world. One benefit of this sort of environment is that sensing the current state is easy. However, our learning technique is not limited to this subset of interactive environments. As long as the current state and the behavior of the other agent (which we are modeling) can be perceived, our learning technique is applicable.

For any given agent and environment, the state space may have to be continuous. This is because, in a stimulating environment where the agent and human are competing or cooperating intimately, it is possible that a small difference in state can determine a large difference in behavior. A continuous state space can also help in achieving a realistic virtual environment. For example, a discrete state space seems very unnatural for a car driving training simulator. Therefore, our technique uses a continuous internal representation of states (and actions). Since this is a very general representation, all continuous and most discrete state spaces are supported.

We represent the state space S as a real-valued, n-dimensional feature vector. Thus $S \subset$

$\mathbb{R}^n$, and a state $\mathbf{s} \in \mathbb{R}^n$ is a point within this feature space. These features can be specified manually, or automatically discovered through principal component analysis. We assume that the designer has by some means provided a good, compact state representation for the given agent and environment. In other words, we assume that the state space dimensionality n is small. This is important because a compact state space will help the agent adapt quickly and better generalize what it has learned. However, our technique still operates successfully with a non-compact state representation, though learning may not be as fast as desired. As an example of a compact and effective state representation, in our 1-on-1 simplified rugby case study, the state is simply the translation-invariant separation of the agent and the human's avatar.

The state definition should be organized in such a way that states that are similar (according to the Euclidean metric) are usually associated with similar or identical actions. This makes generalization possible.

Like states, actions are internally represented by real-valued vectors, $\mathbf{a} \in \mathbb{R}^m$, so that both discrete and continuous action spaces are supported. If possible, the action space should be defined in such a way that actions can be combined into some sort of "intermediate" action (e.g. a 'left' vector [-1, 0] and a 'forward' vector [0, 1] become a 'diagonal' vector $[-1/\sqrt{2}, 1/\sqrt{2}]$). More formally, it is useful if the actions are organized such that they can be blended into valid intermediate actions using valid binary operators, e.g. $\mathbf{a}' = \mathbf{a}_1 \Delta \mathbf{a}_2$ (for some operator Δ). As we detail shortly, blending is required for case generalization through continuous k-nearest neighbor. However, blending is not performed with generalization through minimax, and thus is not strictly necessary.

4.2 Learning State-Action Cases

The observed state-action pairs of the human's behavior are recorded on-line. That is, at each time step, the current state and the action selected by the human are saved. For simplicity, we use a small constant time step for sampling the human's behavior. For example, in our complex rugby case study, the time step matches the frame rate of the computer graphics animation (15 Hz). These cases represent a discrete sampling of a Markovian model of the human's behavior. While most cognitive scientists postulate that human behavior is non-Markovian (Nadel 2003), Markovian models have often proven effective in the agent/user modeling literature (for more information on this topic see (Carberry 2001; Kerkez and Cox 2003)).

For our method to be entirely accurate, it would be necessary to record any salient features of the human's internal state as part of the current state in each state-action pair. However, the human's internal state is likely inaccessible. Thus our compact state space may be incomplete. Nevertheless, an approximate state space is acceptable; as we empirically show in the experimental results section, the accuracy of our technique is sufficient to produce effective agent behavior.

Each recorded state-action pair is treated as a case in a case-based reasoning engine. A library of useful cases is maintained. Since the state space is continuous, the library is organized as a (possibly hierarchal) partitioning of the state space to facilitate fast lookup of cases. Automatic partitioning techniques (e.g. a kd-tree) can be used to great effect, or partitioning can be performed by the programmer so that human insight may be applied. The library and its partitioning are defined more formally in part 2 of the overview given in section 4.

The case library is originally populated with state-action examples of "generic" human

behavior. These are gradually replaced with user-specific examples, as they are observed by the character. In particular, a limited number of cases are allowed for each region of the state space, and (nearly) duplicate cases are not allowed. Cases are selected for replacement based on their age and unimportance. In other words, if a case was recorded long ago, and/or is very similar to the new case we are adding (in both the state and action), it is likely to be removed. Thus the character has the ability to "forget", which is very important in learning something as non-stationary as human behavior. Also, since any given region of the state space will always contain the same number of cases, there is no need to repartition on-line. This can limit the agent's ability to learn detailed information in unanticipated regions of the state space, but allows for simple and fast maintenance of the case library. The case replacement procedure is defined more formally in part 3 of the overview given in section 4. In our implementation, we use the Euclidean metric for computing the distance between two cases. Specifically, the similarity metric is: $d((\mathbf{s},\mathbf{a})_i,(\mathbf{s},\mathbf{a})_j) = \left\|\mathbf{s}_i - \mathbf{s}_j\right\| + \left\|\mathbf{a}_i - \mathbf{a}_j\right\|$.

4.3 Generalization of Cases

To predict the human's future behavior, the library of cases must be generalized so that, for any given query state, an associated action can be computed. Note that an important question about how to generalize is whether to focus on confidence or caution. For example, if we are confident in our acquired knowledge, we can attempt to strongly exploit our knowledge by assuming that the human will behave as he usually has done when in/near the current state. However, this will ignore unusual behaviors, even worst-case scenarios. Alternatively, we can be cautious and assume that the human will choose the worst-case action (from the agent's perspec-

tive) from those actions he has previously chosen in/near the current state. Both approaches have their merits, and we have developed generalization techniques for both.

To focus on exploitation (i.e. exercise confidence in our acquired knowledge), we use the *continuous* k-*nearest neighbor* algorithm, as shown in figure 3a. That is, the k cases closest to the query state (according to the Euclidean metric) are found and a distance-weighted normalized sum of the associated actions is computed:

$$\sim \mathbf{a} = \frac{\sum_{i=1}^{k} (w_i \cdot \mathbf{a}_i)}{\sum_{i=1}^{k} w_i}, \qquad \text{where } w_i = \frac{1}{d_i^{\,2}}$$

In our case studies we have found $1 \leq k \leq 3$ is effective. $k = 1$ is good for exactness, as no blending of actions occurs. However, $k = 3$ is good if there is no closely matching case, and/or for attaining a more general impression of the human's behavior. Therefore, it is useful to vary k depending on the nearness of the closest case, and/or the required specificity of a prediction. Also, it is helpful to normalize the axes of the state space, so that they will contribute equivalently in computing distance.

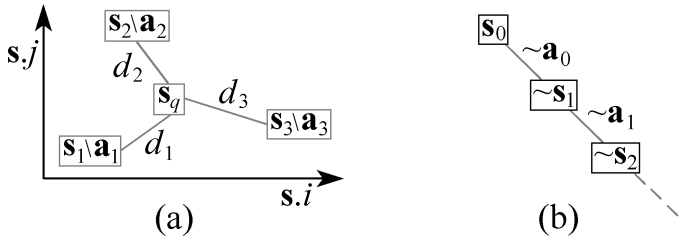

Figure 3. (a) To focus on exploitation (i.e. exercise confidence in our acquired knowledge), we use continuous k-nearest neighbor to generalize cases. (b) Since only one action is predicted for a given state, we predict linearly into the future.

To focus on caution, we use a bounded minimax search, as shown in figure 4. The k

cases closest to the query state (according to the Euclidean metric) are found. Then, the utility of the k actions associated with the retrieved cases is computed using the agent's own fitness function. Finally, the action that results in the minimum fitness for the agent is assumed to be the one the human will select. More formally:

$$\sim \mathbf{a} = \arg \min_{\mathbf{a}_i \in k \text{ neighbors}} (\text{fit}(\mathbf{s}, \mathbf{a}_i)).$$

What is unique about our modified minimax search is that we do not consider all possible actions. Rather, we consider only those actions we have previously seen the human perform in that region of the state space. This not only makes minimax tractable for large or continuous state and action spaces, but also still allows us to exploit our knowledge of the human at the same time as we focus on caution. In addition, note that the information that must be learned for our minimax approach can be learned very rapidly, since we only need to observe some representative cases of the human's preferred behavior.

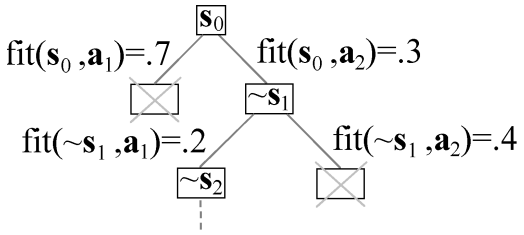

Figure 4. To focus more on caution than confidence, we use a bounded minimax search. The k nearest cases are found, and the associated action that minimizes the agent's fitness function is assumed to be the one the human will select. Since we only expand the minimum node, this technique predicts linearly like when we use k-NN.

When $k = 1$, the minimax approach degenerates to 1-nearest neighbor. That is, it predicts that the human will do exactly what was done previously in the closest matching case. However, as we increase k, this technique becomes more cautious. In fact, in the limit as $k \rightarrow \infty$, our

bounded minimax becomes a standard minimax (assuming that we have cases to cover all possible actions). In our case studies, we have found that small values of k (e.g. ≤ 3) are useful when we want to be more cautious than in k-nearest neighbor, yet still wish to strongly exploit our knowledge. Alternatively, we have found that larger values of k (e.g. $5 \leq k \leq 16$) are useful for being extremely cautious.

Only our minimax approach to generalization requires a fitness function; k-nearest neighbor does not. Further, it must be a *gradient* fitness function. In other words, informative feedback should be given for all states, with fitness leading toward goal states. However, gradient fitness functions are well known, and have been used to great effect in reinforcement learning for speeding up convergence (Sutton and Barto 1998). An interesting point we discovered in our experiments is that, sometimes, it is better to use a fitness function that directly measures the human's fitness, rather than the agent's. Using such a fitness function, minimax would seek to maximize rather than minimize fitness. We have found this approach to be preferable if the human's and agent's goals are not necessarily exact opposites.

While generalization through k-NN blends actions (for $k > 1$), minimax does not. As a result, our action prediction method is still applicable for agents with non-blendable actions, though generalization must be performed through minimax (or k-NN with $k = 1$).

Generalization of cases is introduced formally in part 4 of the overview given in section 4. We use k-NN or bounded minimax for the prediction function f discussed in the overview.

4.4 Using Case-Based Action Prediction In Practice

Our learning technique can be used by agents which cooperate with or compete against the human user. This broad applicability is possible because our technique provides an agent with very general information (predictions of the human's behavior). How to use this information is up to the agent. Even our minimax approach to generalization can be used for coopera-

tion, by seeking to maximize the human's fitness function, or assuming the human will seek to maximize the agent's own fitness function. In other words, the agent and human are trying to maximize a shared fitness function.

The accuracy of the learning in our technique has proven to be very promising (see the figures and tables in the results section). It also has a small memory footprint (usually ≤ 2 MB per agent). Also, the performance of our technique is well within interactive speeds (see table 3). If desired, there are ways to further speed up our technique at the cost of accuracy. These include:

- Predict actions for only a subsample of time steps into the future, holding constant in between (e.g. predict a new action once every 4 time steps).
- Rather than predicting, assume constant actions for some agents (e.g. null or last observed action).
- Ignore agents that will likely have no effect on the current decision making of this agent (e.g. only predict actions for those other agents that are spatially close to this agent in the environment).

Note that the function of our action prediction technique is to supply an agent with supplementary information. Therefore, the way this information is used can be unique for any given agent. For example, given an *n*-step prediction of the human's actions into the future, the agent could perform an informed tree search to plan its own actions through deliberation. Alternatively, this *n*-step prediction could be used as extra inputs into a reactive action-selection method, even a black box implementation.

An alterative way to use action prediction (rather than predicting an entire chain of actions given an initial state, as in figures 3b and 4) is for the agent to request individual predictions for certain states. This can be especially useful for agents that perform decision making through deliberation with a tree search, as the agent can request information specific to any state it encounters while deliberating. However, this requires more CPU than a single linear prediction, and it has not proven to be significantly more accurate in our case studies.

Our action prediction technique is not limited to small environments with only one human. Indeed, it may be appropriate for very complex environments of many agents (more than one human user, etc). However, for adaptation to perform well, the state space definition must always be reasonably compact. This circumvents the curse of dimensionality, thereby allowing our adaptation technique to be used for interesting, difficult problems.

To further counteract the curse of dimensionality, we have found it useful to modularize the action prediction where possible. For example, consider an agent who can perform two independent actions simultaneously (e.g. walk in a given direction while looking at something else). We can split this into two separate problems, with the adaptation for each performed separately. This can help simplify both the state and action spaces. It may also be useful to split up the state space into uncorrelated or independent regions.

Recall that the state-action model of the human's behavior is initialized to "generic" human behavior. This generic information is then gradually forgotten as new user-specific cases are gathered. Specifically, we replace all generic cases before replacing any user-specific cases. Also, for generalization through *k*-nearest neighbor, we more heavily weight user-specific cases over any remaining generic cases.

It is possible to share all acquired knowledge between all adapting agents in a given envi-

ronment. In other words, we can use a single repository for acquired knowledge, which all adapting agents share. This is useful for reducing storage requirements, as well as allowing every agent to behave optimally according to what has been learned. However, this will not be plausible for agents in all categories of multi-agent systems, especially physical agents with limited communication abilities.

We have presented two separate generalization techniques, one focused on confidence and the other on caution. This tradeoff can be further adjusted by varying k, as shown in the results section. Because this tradeoff represents the "personality" of the agent, it is not possible to formally dictate which approach is best. Indeed, as we show in the results section of this paper, both techniques have unique benefits from a fitness perspective. Therefore, the choice of how to generalize is left up to the agent designer. Indeed, this could even be altered dynamically for an agent, depending on the most important goal of the agent at the moment and/or what strategy is proving most effective. Moreover, each technique has unique requirements: if action blending is not possible, then minimax must be used; if a gradient fitness function is not available, then k-nn must be used.

5 Experimental Results

We now present our case studies in detail. In all of our case studies, we used the l_2-norm for our distance metric d. Specifically, $d((\mathbf{s},\mathbf{a})_i,(\mathbf{s},\mathbf{a})_j) = \left\|\mathbf{s}_i - \mathbf{s}_j\right\| + \left\|\mathbf{a}_i - \mathbf{a}_j\right\|$. Also, we used parameter values of $\alpha = \beta = 1$ for the case usefulness metric M. For each adapting agent, the action selection function g was composed of an A* (Russell and Norvig 2003) implementation and our action prediction technique. Specifically, at each time step, A* was used to compute a new plan

with respect to the human's or opponent's predicted actions. The agent then performed only the first action in that plan. The reason we continually reformulated plans was to avoid poor agent behavior due to incorrect action predictions.

5.1 Simplified Rugby Case Study

Our first case study involves two software agents engaging in a sport such as rugby or American football (see figure 5). In this game, the "ball runner" agent attempts to score by running with the ball to a designated end of the playing field. The "tackler" agent attempts to stop the ball runner by tackling him. The environment is discrete, divided into squares. Each player can be in only one square at a time. If the two agents move to the same square, a tackle (end of game) occurs.

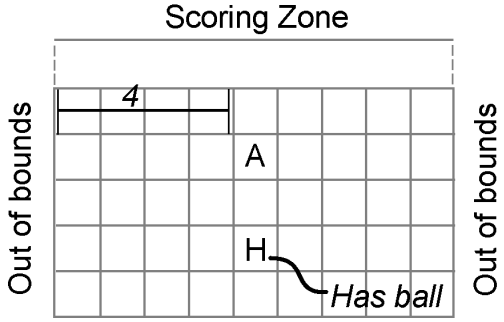

Figure 5. Environment used in the simplified rugby case study.

The game field is 9 squares wide (movement outside of this boundary is not allowed), with a scoring zone at the top of the field. The possible actions available to the software agents are moving to an adjacent square (including diagonally), or staying in place. Thus there are nine

total possible actions. As the game proceeds according to a fixed discrete time step, the agents act simultaneously once per time step. There is no explicit communication between the agents. The current state is fully perceivable to the agents within the bounds of *sensory honesty*: even though the agents' sensors are virtual, they are forced to perform realistically like physical sensors — see (Isla et al. 2001) for a detailed discussion. As a brief example, an agent is not allowed to see through the back of its head, or through solid objects in the environment.

For the learning technique, the compact state definition is simply the translation-invariant separation of the two agents. Therefore, it is a two-dimensional vector of integer values. An action is stored as a two-dimensional integer velocity vector. Thus *k*-nearest neighbor can blend several actions into a valid action. After predicting an action with *k*-NN, its components are in floating point; we round to get integer components.

At the start of each game, the agents are placed in the center of the field, one square apart, as shown in figure 5. The ball runner has the ball, and attempts to score by running past the tackler to the far end of the field. The tackler agent performs its decision making through A* (Russell and Norvig 2003), whereas for the purpose of experimentation the behavior of the ball runner is exhaustively enumerated. Moreover, only the tackler adapts. If prediction information is available, the tackler uses it in its deliberation. However, if prediction information is not available, it assumes the ball runner will continue to perform the last observed action. The tackler's fitness function simply directs it to get as close to the ball runner as possible, without letting the ball runner pass it by (and thereby be able to score). This fitness function is given in pseudo-code below, where $\mathbf{P}^C$ is the position of the ball carrier and $\mathbf{P}^T$ is the position of the tackler agent:

```
fitness_tackler(P^C, P^T)
{
    if (P^C.y > P^T.y)  {
        /* Ball carrier has passed tackler, so he can score easily. */
        return (–100 – || P^C – P^T ||);
    }
    else if (P^C == P^T)  {
        /* Close enough to tackle. */
        return (500);
    }
    else {
        /* Tackler is in front of ball carrier (where it should be). */
        return (100 – || P^C – P^T ||);
    }
}
```

We performed several experiments in this case study, varying k and the generalization technique used. Note that in all these experiments, we forced the ball runner to exhaustively try all behaviors of length seven (i.e. a game lasting seven time steps). Each behavior was presented to the tackler three times; the first two times to learn, and then on the third iteration we measured the performance of the tackler. We gathered statistics on the effectiveness of the tackler's learning, with regards to ratio of tackles vs. scores, and to average forward progress made. These results are presented in tables 1 and 2.

Table 1. Average ratio of tackles to scores for all behaviors of length 7. With no learning, the ratio was only 5.54 : 1.

	$k = 1$	$k = 2$	$k = 6$	$k = 12$
k-*NN*	69.16 : 1	25.61 : 1	23.69 : 1	29.825 : 1
Minimax	69.16 : 1	239.5 : 1	963.8 : 1	1729 : 1

Table 2. Average forward progress made by ball runner before end of game for all behaviors of length 7.

	$k = 1$	$k = 2$	$k = 6$	$k = 12$
k-*NN*	−0.0553	−0.0625	−0.0629	-0.0444
Minimax	−0.0553	0.00957	0.0314	0.0325

With no learning for the agent (i.e. no observed state-action cases added to the agent's action prediction library, just the default cases), the ratio of tackles to scores is only 5.54 : 1. This is significantly lower than the ratios achieved by using our learning technique.

Note that for $k = 1$, the experimental results are equal for both generalization techniques, because the two techniques are equivalent when only using a single case. Also note that *k*-NN effectively holds the ball runner to negative forward progress, whereas our bounded minimax allows positive forward progress for $k > 1$. This is because, while *k*-NN strongly exploits (especially for smaller values of *k*), minimax is more conservative. However, minimax allows the human to score less often than *k*-NN, since it more effectively covers worst cases. As can be seen in the tables, minimax becomes more conservative and cautious as *k* increases. *k*-NN can be less effective for larger *k* since it produces a deterministic prediction (i.e. averages conflicting actions). However, *k*-NN with $k > 1$ can be useful for producing stable predictions over time,

since it averages local cases.

Performance numbers for our technique (for both case studies) are given in table 3. We also performed experiments on the effect of clearing vs. not clearing the case library between behaviors — this proved to have little effect.

Table 3. Typical performance results of action prediction in our two rugby case studies (for one adapting agent). Case learning time is negligible, and therefore is not listed. We used a 1.7 GHz PC with 512 MB RAM.

	Simplified Rugby	*Virtual Rugby*
Action selection freq.	1 Hz	15 Hz
Action prediction time	21 μ sec	30 μ sec
Total avg. CPU usage	0.12%	6.82%
Memory usage	$\ll$ 1 MB	$\leq$ 2 MB

5.2 Complex Virtual Rugby Case Study

Our second case study is similar to the first, but is significantly more complex and involves a human user. In the simplest configuration, there are two players. The human has the role of ball runner, while the agent is the tackler. However, in our experiments we varied the roles and number of members of each team, as we describe shortly.

The virtual environment is continuous, both in terms of the state and action space. The agent and the human's avatar can be located at any real-valued position on the playing field, and can be traveling at any velocity in the range [0.0, *MaxVelocity*] along a real-valued two-dimensional vector. The agent can change each component of its velocity by values in the range [−*MaxAcceleration*, *MaxAcceleration*] once per time step. The human controls his avatar

through a joystick, and has no software enforced acceleration limit. Therefore, the human has a distinct advantage in terms of maneuverability.

For this environment, the compact state definition used in learning is composed of: (1) the translation-invariant separation of the agent and human, and (2) the velocity vectors of the agent and human. Thus the compact state has six real-valued components. The action is stored as a two-dimensional real-valued acceleration vector.

As in the simplified rugby case study, there is no explicit communication between the agent and human, and the agent can fully perceive the current state of the virtual environment within the bounds of sensory honesty. The current state is presented to the human user in real time through computer graphics, as shown in figure 1. A fixed time step of 15 Hz is used in this case study, which is fast enough to make learning challenging and forces our learning technique to operate quickly.

We performed several experiments in this case study, varying the number of agents on each team, the initial state, and the human user's behavior. We gathered statistics on the accuracy of the agent's learning, the increase in its success rate with respect to the human user (i.e. ratio of tackles to scores), and the runtime performance of our adaptation system. These results are presented in figures 6 and 7, and table 3. Note that in the graphs, the experiments started with the agent having very incorrect information about the human user. We purposely did this to demonstrate how quickly our technique learns, especially when its current knowledge is invalid.

As seen in figure 6, our learning technique reaches high accuracy quickly, and then continues to learn such that it remains accurate for non-stationary human behavior. Also, as seen in figure 7, the agent eventually beats the human more than 50% of the time, even though the human's acceleration is not bounded but the agent's is. An interesting result from our experiments

is that our learning technique works well for multi-agent environments. For example, we performed an experiment where several agents (all on one team) were supposed to work together to tackle the human user. This experiment was successful, as the agents could predict what their teammates would do and thereby were able to cooperate effectively (see figure 8).

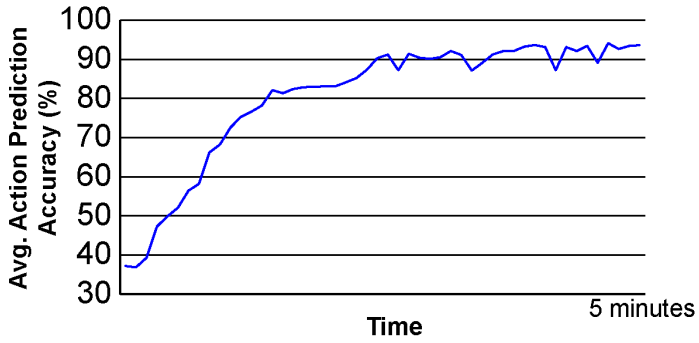

Figure 6. Accuracy of predicting the human's actions (L2-norm) in the complex rugby case study. This experiment started with the agent having very incorrect information about the human user.

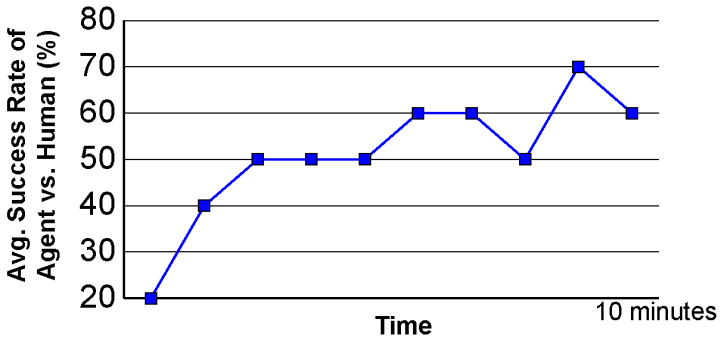

Figure 7. After a few minutes, the agent is able to outperform a human user in a complex continuous rugby environment, even though the human has more maneuverability. This experiment started with the agent having very incorrect information about the human user.

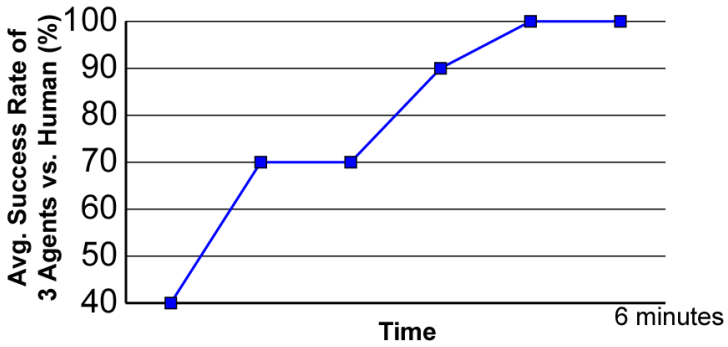

Figure 8. Effectiveness of three agents against the human user in the complex rugby environment. The agents learn to predict the actions of each other as well as a human, and thereby learn to cooperate effectively in the rugby environment.

A frame-by-frame example of our learning technique in action is given in figure 9. Further details and examples are given in a supplementary digital video, available from http://www.npl.com/~jon/video.html.

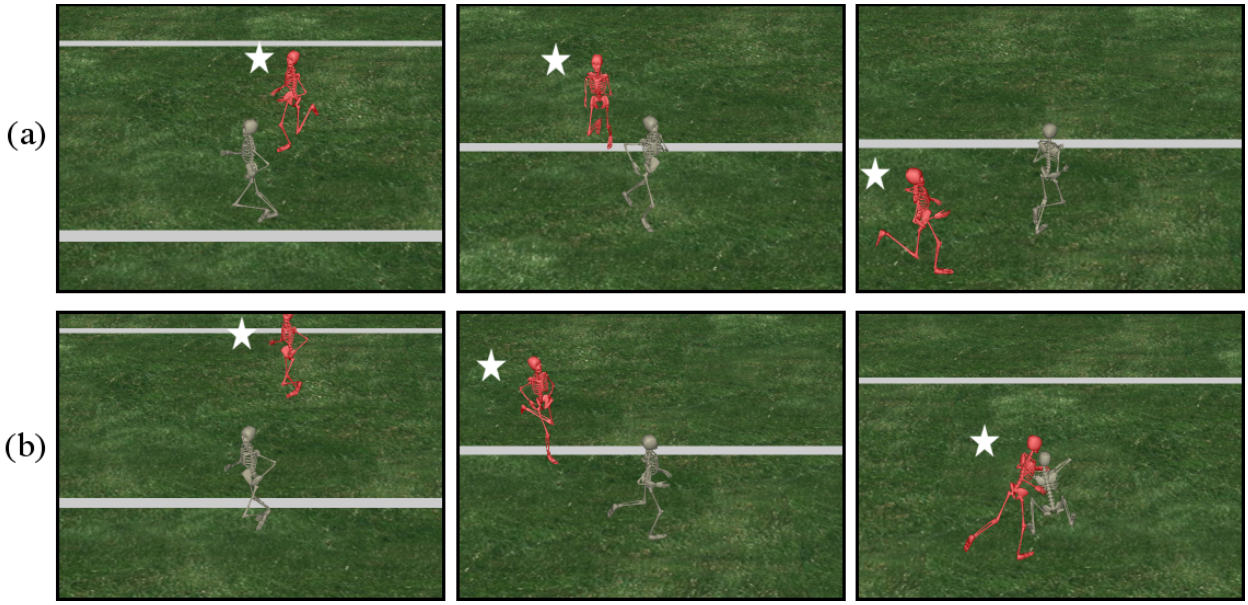

Figure 9. (a) The human user performs a loop, which succeeds in getting past the tackler agent (skeleton marked with a star). As a result, the human can score. (b) Now that the agent has adapted, the next time the human attempts a loop it predicts the human's actions and tackles him. Further examples of our technique in action are given in a supplementary digital video, available from http://www.npl.com/~jon/video.html.

5.3 Capture The Flag

This case study is based on a well-known multi-agent research test bed called *Gamebots* (Kaminka 2002). This test bed modifies the popular computer game *Unreal Tournament 2003*, allowing a programmer to replace the built-in agent AI. In Unreal Tournament, the virtual world is a complex 3D environment of rooms, hallways, stairs, etc. It is populated with two or more players (virtual humans) organized into two teams. Each player is controlled by a human user or software agent. The players are armed with "tag guns"; once a player is tagged, he is out for a period of time. The objective is to reach the other team's flag.

We have modified the Gamebots test bed so that, rather than overriding the entirety of an agent's standard AI, we only override the choice of combat movements (e.g. dodging and/or moving into an advantageous position to tag an opponent). Goal selection, environment navigation, etc, are performed in the standard manner by the agent. In our experiments, there was one human user and multiple software agents (approximately half of the agents on the human's team, the others on the opposing team).

Each agent uses action prediction to predict the human's 2D movement to better cooperate with or compete against him. Actions such as aiming the tag gun, firing, and jumping were not predicted.

Due to the complexity of this environment, it is not plausible to use a complete state space for action prediction. Thus we use an approximate state space. The state space is composed of: (1) the translation-invariant separation between the human user and the opponent closest to him; (2) the 2D velocities of the human and his closest opponent; and (3) an angle θ repre-

senting the average direction toward nearby obstacles. Thus the compact state definition is: (Δx, Δy, $\mathbf{V}^H$, $\mathbf{V}^A$, θ), where $\{\Delta x, \Delta y\}$ is the translation-invariant separation between the human and his closest opponent, and $\{\mathbf{V}^H, \mathbf{V}^A\}$ are the 2D velocities of the human and closest opponent respectively. All nearby obstacles are represented by a single mean angle, θ (oriented around the "up" direction), representing the average direction toward the obstacles according to the human's frame of reference. Assuming the user's avatar will never be in a very narrow hallway or room, this angle will be valid since the avatar will not be surrounded by obstacles in the environment.

The results of this case study are presented in figure 10. Action prediction is of a lower accuracy than in our rugby case studies, largely because we use such a crude approximation of the current state. Nevertheless, our results are still promising, suggesting that our adaptation technique scales sufficiently to be useful for complex environments and agents. A slide show of this case study is given in figure 11.

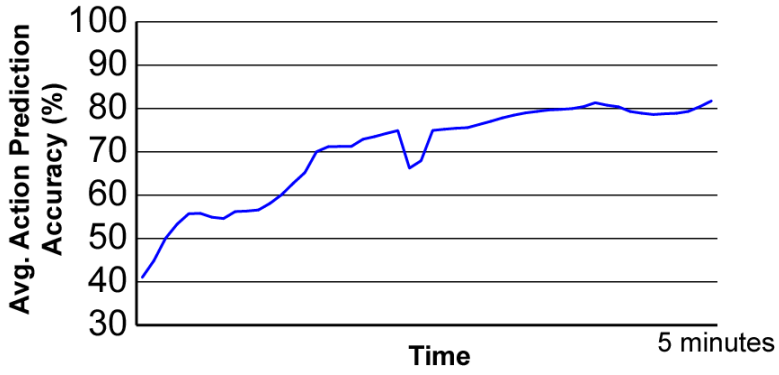

Figure 10. Accuracy of predicting the human's actions (L2-norm) in the Capture The Flag case study. This experiment started with the agent having very incorrect information about the human user.

6 Summary and Discussion

We have presented a novel method that enables autonomous cooperate/competitive agents to quickly adapt on-line to better interact with a unique human user (or another synthetic agent). Our technique is very general, simply providing extra information to an agent, and thus can be used with many existing types of agents. Our system supports both reactive and deliberative decision making, in discrete or continuous state/action spaces. Our contribution in this paper is important because we present a solution for a previously unsolved problem: fast on-line learning for agents interacting with humans. Learning is an important problem for agents in many interactive software applications, such as training simulators, computer games, etc. For further examples than those given in this paper, see the supplementary digital video at http://www.npl.com/~jon/video.html.

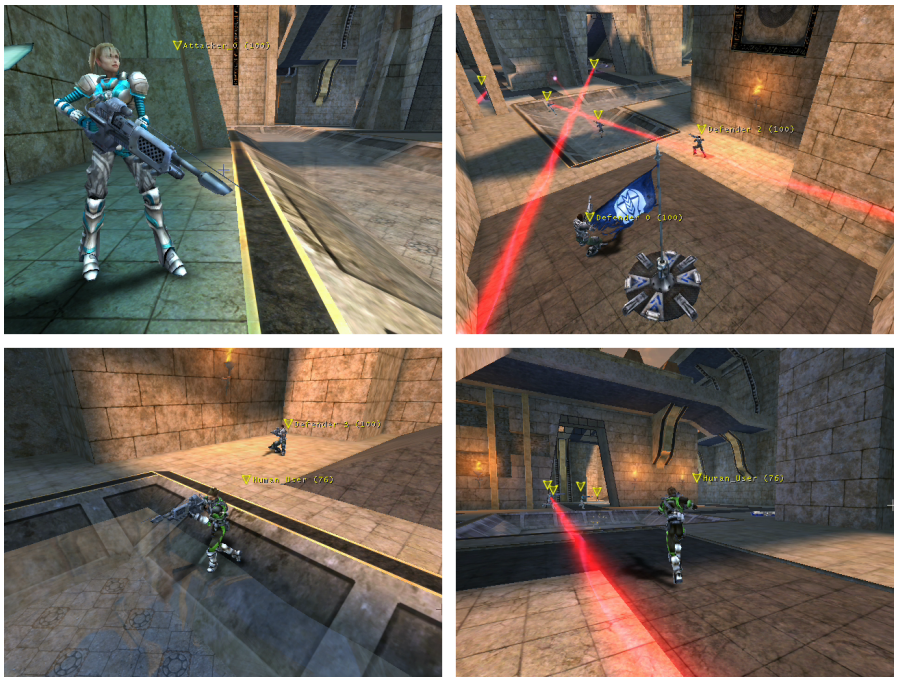

Figure 11. Slide show of our Capture The Flag case study.

The reason why our technique is sufficient for on-line agent learning to better interact with a unique human user is because the agent learns all the non-stationary knowledge it needs to more optimally select actions. For example, in our rugby case studies, the agent already has adequate perception, motor control, and decision-making skills. To optimally choose what actions to perform, it simply needs to know which of its candidate actions is most valuable for any given situation. By accurately predicting the behavior of the human, the agent can predict the results of its actions and thereby can more optimally select what to do. Thus our knowledge-gathering approach to interactive on-line learning can be seen as hitting a "sweet spot" between nature vs. nurture.

It is because our technique forgets old cases that it is able to learn non-stationary behavior. Forgetting also helps keep memory requirements stable. However, forgetting can cause an agent to make a mistake if the human waits sufficiently long before repeating a behavior.

An interesting benefit of our technique is that, since an agent can adapt on-line, it can fill "gaps" in its decision-making skills. In other words, a programmer does not have to carefully construct the agent such that it will immediately handle every possible situation properly. This can also make an agent more robust. Further, in environments where there is no pareto-optimal Nash equilibrium (i.e. no universally best strategy), adaptation may be necessary to achieve and maintain good behavior.

However, while our technique has been shown empirically to work well, there are some weaknesses that are important to recognize. First, while knowledge gathering is very fast, using that knowledge does require a nontrivial amount of CPU time. As a result, it may not be plausi-

ble to have many adapting agents in the same interactive environment. Second, when generalizing cases, we make assumptions: most notably that the human will only perform an action he has previously performed in/near the current state. Thus the agent may ignore worst-case scenarios.

Although our technique has proven effective in our case studies, there is no guarantee that it will be effective for every imaginable agent and environment. However, as long as our assumptions in section 4 are met, we believe that our technique will work well for many categories of agents and environments. In summary, we assume that a compact state representation and (if minimax is used for generalization) an effective gradient fitness function have been provided. Recall that organizing the action space such that actions can be blended is not strictly necessary because case generalization through minimax performs no blending. An example of an agent for which our technique will perform poorly is a chess player, because the state space is too large and complex.

Of course, our learning technique is not limited to learning about a human; it can be used for one synthetic agent to learn about another synthetic agent. The reason we have focused on agent-human interactions in this paper is because this type of learning is generally more difficult than in agent-agent interactions. Another possible use of our technique is the creation of entirely new agent decision-making modules in an on-line fashion. We can use the state-action model of the human's behavior to perform decision making for an agent. This is fundamentally similar to *behavior capture* (van Lent and Laird 1998) and *behavior cloning* (Kasper et al. 2001). One drawback to this proposed use of our method is that the decision making is somewhat shallow, since it only predicts actions (not task or goal selection).

For future work, we are interested in developing a more complete on-line agent adaptation system. Ever since the pioneering work of Brooks (1986), agent AI has usually been im-

plemented in a layered fashion. Our action prediction technique is most pertinent for low-level decision making (e.g. action selection). However, higher-level decision making (e.g. task and goal selection) is also important. We want to develop new learning techniques for these higher layers. We envision these new learning techniques, along with our action prediction technique, composing a complete and effective on-line agent adaptation system. Another interesting avenue for future work may be to replace the case-based reasoning in our action prediction method with a more powerful and modern approach such as (Wilson and Martinez 2000). Yet another direction for future work may be to apply our method for use in adaptive user interfaces (Zhu, Greiner, and Haubl 2003). A further direction is to explicitly address partial observability of the state when recording state-action pairs of a human's behavior.

References

Blumberg, B., Downie, M., Ivanov, Y., Berlin, M., Johnson, M.P., and Tomlinson, B. 2002. Integrated learning for interactive synthetic characters. *In* Proceedings of SIGGRAPH 2002, pp. 417-426.

Brooks, R. 1986. A robust layered control system for a mobile robot. IEEE Journal of Robotics and Automation, **2**: 14-23.

Bui, H., Kieronska, D., and Venkatesh, S. 1996. Learning other agents' preferences in multi-agent negotiation. *In* Thirteenth National Conference on Artificial Intelligence, pp. 114-119.

Carberry, S. 2001. Techniques for plan recognition. User Modeling and User-Adapted Interaction, **11**: 31-48.

Carmel, D., and Markovitch, S. 1996. Incorporating opponent models into adversary search. *In* Thirteenth National Conference on Artificial Intelligence, pp. 120-125.

Carmel, D., and Markovitch, S. 1996. Learning models of intelligent agents. *In* Thirteenth National Conference on Artificial intelligence, pp. 62-67.

Evans, R. 2002. Varieties in learning. *In* AI Game Programming Wisdom, pp. 567-578. *Edited by* E. Rabin. Charles River Media, Hingham MA.

Gmytrasiewicz, P., and Durfee, E. 2000. Rational coordination in multi-agent environments. Autonomous Agents and Multi-Agent Systems, **3**(4): 319-350.

Gmytrasiewicz, P., and Durfee, E. 2001. Rational communication in multi-agent environments. Autonomous Agents and Multi-Agent Systems, **4**: 233-272.

Hu, J., and Wellman, M.P. 1998. Online learning about other agents in a dynamic multiagent system. *In* Second International Conference on Autonomous Agents, pp. 239-246.

Isbell, C. Jr., Shelton, C., Kearns, M., Singh, S., and Stone, P. 2001. A social reinforcement learning agent. *In* Fifth International Conference on Autonomous Agents, pp. 377-384.

Isla, D., Burke, R., Downie, M., and Blumberg, B. 2001. A layered brain architecture for synthetic creatures. *In* Proceedings of IJCAI, pp. 1051-1058.

Kaminka, G., Veleso, M., Schaffer, S., Sollitto, C., Adobbati, R., Marshall, A., Scholer, A., and Tejada, S. 2002. Gamebots: a flexible test bed for multiagent team research. Communications of the ACM, **45**(1): 43-45.

Kasper, M., Fricke, G., Steuernagel, K., and von Puttkamer, E. 2001. A behavior-based mobile robot architecture for learning from demonstration. Robotics and Autonomous Systems, **34**: 153-164.

Kerkez, B., and Cox, M. 2003. Incremental case-based plan recognition with local predictions.

International Journal of Artificial Intelligence Tools, **12(**4): 413-463.

Laird, J. 2001. It knows what you're going to do: Adding anticipation to the quakebot. *In* Proceedings of the Fifth International Conference on Autonomous Agents, pp. 385-392.

Littman, M. 1994. Markov games as a framework for multi-agent reinforcement learning. *In* Proceedings of ML94, pp. 157-163.

Mataric, M. 2000. Getting humanoids to move and imitate. IEEE Intelligent Systems (July), 18-24.

Nadel, L. 2003. Encyclopedia of Cognitive Science, Nature Pub. Group, London.

Nicolescu, M. 2003. A framework for learning from demonstration, generalization and practice in human-robot domains. PhD Thesis, University of Southern California.

Price, R. 2002. Accelerating reinforcement learning through imitation. PhD Thesis, University of British Columbia.

Reynolds, C. 1987. Flocks, herds, and schools: A distributed behavioral model. *In* Proceedings of SIGGRAPH 1987, pp. 25-34.

Rovatsos, M., Weib, G., and Wolf, B. 2003. Multiagent learning for open systems: A study on opponent classification. *In* Lecture Notes on AI 2636, pp. 66-87. *Edited by* E. Alonso, D. Kudenko, and D. Kazakov. Springer.

Russell, S., and Norvig, P. 2003. Artificial Intelligence: A Modern Approach (second edition). Prentice-Hall, Englewood Cliffs, NJ.

Sen, S., and Arora, N. 1997. Learning to take risks. *In* Collected papers from American Association for Artificial Intelligence (AAAI-97) workshop on multiagent learning, pp. 59-64.

Stone, P., and Veloso, M. 1997. Multiagent systems: A survey from a machine learning perspective. Autonomous Robots, **8**(3): 345-383.

Stone, P. 2000. Layered Learning in Multi-Agent Systems: A Winning Approach to Robotic Soccer. MIT Press, Cambridge, MA.

Sutton, R., and Barto, A. 1998. Reinforcement Learning: An Introduction. MIT Press, Cambridge, Massachusetts.

Tomlinson, B., and Blumberg, B. 2002. Alphawolf: Social learning, emotion and development in autonomous virtual agents. *In* First NASA Goddard Space Flight Center / NASA Jet Propulsion Laboratory Workshop on Radical Agent Concepts.

Tran, T., and Cohen, R. 2002. A reputation-oriented reinforcement learning strategy for agents in electronic marketplaces. Computational Intelligence, **18**(4): 550-565.

van Lent, M., and Laird, J. 1998. Behavior capture: Motion is only skin deep. *In* Proceedings of Lifelike Computer Characters.

Vidal, J., and Durfee, E. 1997. Agents learning about agents: A framework and analysis. *In* Collected papers from American Association for Artificial Intelligence (AAAI-97) workshop on multiagent learning, pp. 71-76.

Weiss, G. 1999. Multiagent Systems: A Modern Approach to Distributed Artificial Intelligence. MIT Press.

Wilson, D.R., and Martinez, T. 2000. An integrated instance-based learning algorithm. Computational Intelligence, **16**(1): 1-28.

Zhu, T., Greiner, R., and Haubl, G. 2003. Learning a model of a web user's interests. *In* Ninth International Conference on User Modeling.