Fast Multi-Level Adaptation for Interactive Autonomous Characters

JONATHAN DINERSTEIN and PARRIS K. EGBERT
Brigham Young University

Adaptation (online learning) by autonomous virtual characters, due to interaction with a human user in a virtual environment, is a difficult and important problem in computer animation. In this article we present a novel multi-level technique for fast character adaptation. We specifically target environments where there is a cooperative or competitive relationship between the character and the human that interacts with that character.

In our technique, a distinct learning method is applied to each layer of the character's behavioral or cognitive model. This allows us to efficiently leverage the character's observations and experiences in each layer. This also provides a convenient temporal distinction between what observations and experiences provide pertinent lessons for each layer. Thus the character can quickly and robustly learn how to better interact with any given unique human user, relying only on observations and natural performance feedback from the environment (no explicit feedback from the human). Our technique is designed to be general, and can be easily integrated into most existing behavioral animation systems. It is also fast and memory efficient.

Categories and Subject Descriptors: I.2.6 [**Artificial Intelligence**]: Learning; I.3.6 [**Computer Graphics**]: Methodology and Techniques—*Interaction techniques*; I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism—*Animation*

General Terms: Algorithms, Human Factors, Performance

Additional Key Words and Phrases: Computer animation, character animation, behavioral modeling, AI-based animation, machine learning

1. INTRODUCTION

Behavioral animation has become popular for creating autonomous self-animating characters. However, an important limitation of traditional behavioral animation systems (with respect to interactive environments) is that they are largely static. In other words, the character's behavior does not adapt online due to interaction with a human user and/or its environment. This may not only lead to a lack of variety and nonstimulating animation, but also makes it easy for a human user to predict a character's actions.

Intelligent, rapid online adaptation of a character's behavior would be extremely useful for many computer graphics applications. For example, consider a training simulator where a virtual character is an opponent to a human user (see Figure 1). By learning online 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.

In this article, we present a novel multi-level technique for fast character adaptation. Our technique is grounded in traditional machine learning, and is further inspired by insights into how humans

Authors' address: Brigham Young University, Computer Science Department, 3366 TCMB, Provo, UT 84602; email: J. Dinerstein, jondinerstein@yahoo.com; P.K. Egbert, egbert@cs.byu.edu.

Fig. 1. A virtual character (red skeleton) tries to catch the human's avatar (brown skeleton) in a rugby simulation. By adapting to the human's tactics, the character can become a more challenging, personalized opponent.

learn. We focus our discussion on environments where the character has a cooperative or competitive relationship with the human user. Note that our learning technique is applicable to both *behavioral* (reactive) and *cognitive* (deliberative) models of decision making for autonomous virtual characters. Also, our technique is designed to be general, and can be easily integrated into most existing behavioral animation systems.

Our adaptation technique contains a small set of distinct learning methods. A distinct learning method is applied to each layer of the character's behavioral (and/or cognitive) model. This allows us to efficiently leverage the character's observations and experiences in each layer. Thus the character can quickly and robustly learn how to better interact with any given unique human user, relying only on observations and natural performance feedback from the environment (no explicit feedback from the human).

Each learning method contained in our technique is specially designed with regards to the temporal constraints and degree of abstraction of the decision-making layer to which it is applied. For adaptation in low-level decision making (*action selection*), we take an observation-based approach, which operates through case-based reasoning. Specifically, state-action pairs of the human's observed behavior are recorded. These cases are then used to predict the human's future behavior. Through these predictions, a character can extrapolate the long-term utility of any activity it may engage in, and thereby can intelligently select its actions.

For adaptation in mid-level decision making (*task selection*), we take a combination experience- and planning-based approach. First, the character approximately learns in which regions of the state space each of its possible tasks are likely to help achieve its goals. Then, to select a task for the character's current state, the most promising tasks are found and further scrutinized by running internal simulations to more accurately measure their utility. To ensure that the simulations will be adequately accurate, we use the model of the human user's decision making constructed during low-level adaptation as described above.

Finally, for high-level decision making (*goal selection*), we monitor changes in the character's emotional state to adapt its personality. Specifically, the character learns in which situations selecting a given goal leads to increased happiness. This can be combined with existing interactive character training techniques so that a human user can explicitly adjust the character's personality.

In addition to these learning methods, we also provide the character with the ability to mimic novel, valuable behaviors that it observes the human user's avatar perform. Mimicking is executed in two steps. First, when the character determines that the human has achieved her goal, the human's behavior is tested to see if it is novel to the character; if it is novel, the corresponding state-action sequence is recorded. Second, the character can explicitly mimic this behavior when appropriate and desired.

Our multi-level adaptation technique allows an intelligent, complex character to adapt in the following ways:

—Learn to accurately predict the human's actions.
—Learn the best behavior for any given situation.
—Mimic the effective behaviors of the human.
—Learn through emotional feedback to maximize happiness.

We begin by surveying related work. We then give a brief introduction to the theoretical background of our technique, both in terms of behavioral animation and machine learning. Next we turn to a discussion of the key challenges in successfully performing fast and robust adaptation, providing a roadmap of what our technique must accomplish. We then present our technique for rapid adaptation of virtual characters. Finally, we conclude with our experience from three case studies. We use our first case study (a virtual sport like rugby or American football) for examples throughout this article. Our rugby case study is a challenging environment for character adaptation, because it is very fast-paced, involves synthetic humans, and the state/action spaces are continuous.

2. RELATED WORK

Our multi-level approach to adaptation is inspired by Stone [2000], where agents (characters) in a multi-agent environment learn offline in a layered fashion to perform their tasks and thereby learn better than with a direct, non-layered approach. What is new in our work is that we apply this layered-learning concept to achieve *faster* learning so that our characters can adapt online within the tolerances of human time and patience. Also, the per-layer learning methods used in Stone [2000] are standard machine learning techniques, whereas we have developed custom learning methods to fulfill our unique needs.

In some aspects, our custom per-layer learning methods resemble existing machine learning approaches. Our low-level learning method most closely resembles *agent modeling* or *user modeling* [Gmytrasiewicz and Durfee 2000; Kerkez and Cox 2003; Zhu et al. 2003]. Some notable differences from these existing techniques are that our method operates in continuous state/action spaces, can predict the human user's behavior several steps into the future, and can exercise caution or confidence when predicting actions. Our mid-level adaptation method is most closely related to deliberation-based approaches to learning [Yoon 2003], but leverages *reinforcement learning* [Kaelbling et al. 1996; Sutton and Barto 1998] concepts to expedite the deliberation process. Our high-level adaptation method is very similar to previous work in computer animation for allowing a user to interactively train a virtual character [Blumberg et al. 2002; Evans 2002], as well as methods for learning from change in emotional state [Tomlinson and Blumberg 2002; Gadanho 2003]. Our mimicking technique most resembles previous work in *learning from observation* or *programming by demonstration* [Price 2002; Kasper et al. 2001; van Lent and Laird 2001], but our approach is either significantly faster, more general, or more automatic than previous techniques.

As pointed out above, while our per-layer learning methods bear similarities to existing techniques in machine learning, several aspects of our learning methods are novel. We more thoroughly discuss these novel aspects later in this article after presenting each of our learning methods.

Our approach is also inspired by a study of how humans learn [Minsky 1985; Meltzoff and Moore 1992; Schyns et al. 1998]. Humans learn through a variety of stimuli, both observed and experienced,

and apply lessons learned in many distinct ways. Therefore, it is sensible that a robust character adaptation technique must also be multi-faceted, leveraging pertinent experience in several distinct ways. The optimal use of one experience to learn several lessons also makes learning faster.

A number of noteworthy architectures for behavioral animation of autonomous characters have been proposed [Reynolds 1987; Tu and Terzopoulos 1994; Blumberg and Galyean 1995; Perlin and Goldberg 1996; Funge et al. 1999; Faloutsos et al. 2001; Isla et al. 2001; Monzani et al. 2001]. While producing impressive results, most of these systems have not incorporated any form of learning and therefore cannot adapt in order to more skillfully interact with any given human user.

Offline behavioral learning has recently seen some attention, for example in our own work [Dinerstein et al. 2004; Dinerstein and Egbert 2004]. However, offline learning does not address the problem of interest in this article: interaction-based adaptation.

The multi-level adaptation approach we propose in this article has strong underpinnings in the Belief-Desire-Intention (BDI) agent model [Rao and Georgeff 1995]. In BDI, the agent's internal state is composed of *beliefs* (i.e. knowledge), *desires* (i.e. goals), and *intentions* (i.e. plans). The agent uses this internal state to select actions. Our approach is similar in that we collect knowledge through interaction, which the character leverages to more effectively make decisions and thereby fulfill its goals. However, our approach varies from traditional BDI because we acquire and utilize knowledge in a layered fashion.

A notable work performed in virtual character adaptation through prediction is Laird [2001]. In this technique, the character memorizes facts about the layout of the environment, and then uses this information to weakly predict the human's future behavior (assuming the human will behave exactly like the character would in the same situation). This prediction is then used to improve the virtual character's decision making. This interesting technique is one of the first working examples of useful adaptation in practice. However, the character only learns about the layout of the environment, not the human's behavior, so the character's ability to adapt is distinctly limited.

Online, interaction-based behavioral learning has only begun to be explored [Evans 2002; Tomlinson and Blumberg 2002]. A notable example is Blumberg et al. [2002], where a virtual dog can be interactively taught by the human to exhibit desired behavior. This technique is based on reinforcement learning with immediate explicit feedback from the human user, and has been shown to work extremely well. However, it has no support for long-term reasoning to accomplish complex tasks (i.e. it is reactionary). Also, the approach is based on continual and explicit feedback from the character's "master." Therefore these techniques, while interesting and useful, are best suited for a "master-slave" relationship between a character and a human user, not a competitive or cooperative relationship. It may be possible to alter Blumberg et al. [2002] such that the underlying learning algorithm utilizes feedback from a synthetic trainer rather than a human, but it is not clear at this time how this could be done.

There is need for a technique that allows autonomous virtual characters with complex behavioral and/or cognitive models (in a competitive or cooperative relationship with the human) to rapidly adapt online to better fulfill their goals in an interactive environment. This adaptation should be effective, believable, fast enough to not tax human patience, and appear similar to the way a real human peer or competitor would learn through interaction. Our contribution is a technique that fulfills these needs.

3. BACKGROUND

3.1 A Common Behavioral Animation Framework

For our online character adaptation technique to be broadly useful, it must be applicable in most, if not all existing behavioral animation systems. As discussed in Millar et al. [1999], behavioral animation

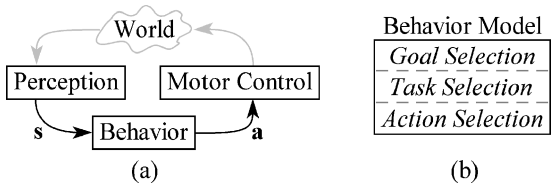

Fig. 2. (a) A simplified common behavioral animation framework. **s** is the current perceived state, and **a** is the action selected to be performed. (b) Most behavioral models are layered to break up the decision-making process into tractable pieces.

systems can be cast into a common framework. While the details of these systems can vary significantly, they are nevertheless quite similar from a framework-level perspective. We develop our technique within this common framework so that it can be usable in current and future behavioral animation systems.

A simplified common behavioral animation framework (as seen from a high level) is shown in Figure 2. Behavioral animation systems in general have three primary modules: perception, behavior, and motor control. We are most interested in the behavior module, as this is where decision making takes place and therefore is key to adaptation. This module is often called the *behavioral model*. Further details of this module are shown in Figure 2b. The decision-making process is usually split into layers, decomposing it into manageable pieces. This layered approach to decision-making has been standard for control of autonomous agents since the pioneering work of Brooks [1986]. The upper layers perform higher-level, coarse-grain decision making, while the lower layers perform lower-level, fine-grain decision making. This provides a natural and powerful breakup of the decision-making process.

It is most common, both in behavioral animation and overlapping fields, for there to be three layers in the behavioral model. However, there are no universal names for these layers. Throughout this article, we refer to these as *action selection*, *task selection*, and *goal selection* (in order of fine to coarse temporal granularity).

Our character adaptation technique integrates with a behavioral animation system through the layers of the behavioral model. A distinct learning method is applied to each layer. This allows us to efficiently leverage the character's observations and experiences for each layer, based on that layer's unique degree of abstraction and temporal constraints. However, note that while our technique is specially designed for three-layer behavioral models, it can be used effectively with models of any number of layers.

There are two types of decision-making models that are popular for use in behavioral animation. The traditional *behavioral model* [Reynolds 1987] performs reactive decision making. *Cognitive modeling* [Funge et al. 1999] was introduced to provide virtual characters with deliberative decision making. Our adaptation technique works well for either approach. However, for simplicity, and without loss of generality, we will use the term "behavioral models" to represent both behavioral and cognitive models.

3.2 Machine Learning

Our multi-level character adaptation technique is composed of four custom learning methods. These methods are grounded in traditional machine learning approaches, but are novel in several aspects. We now briefly review pertinent issues in machine learning, and give some motivation for our need to develop custom learning methods to achieve practical online character adaptation.

Formally, machine learning is the process of a computer program learning an unknown mapping, often formulated as $f : \mathbb{R}^n \rightarrow \mathbb{R}$. There are numerous machine learning techniques presented in the literature [Mitchell 1997]. Each approach has its own strengths and weaknesses, and each is a good

fit for some category of applications. Nevertheless, there is no machine learning technique that works well for all applications (thus the existence of so many alternative approaches).

In general, there are four main categories of machine learning techniques:

(1) *Example-based learning,*
(2) *Experience-based learning,*
(3) *Planning-based learning,*
(4) *Observation-based learning*.

Most existing machine learning techniques are example-based. They learn using a large set of explicit input-output examples provided by the user. These techniques include artificial neural networks, decision trees, and so forth [Mitchell 1997]. Many of these techniques learn very slowly and/or require a large number of input-output examples, and therefore are not appropriate for interactive, online learning. Another challenge is that it may be inappropriate or prohibitively difficult for an end user to provide the necessary input-output examples. Thus these techniques (in their traditional form) are not applicable for our problem of interest.

Reinforcement learning [Kaelbling et al. 1996; Sutton and Barto 1998] is an effective approach to experience-based learning. In reinforcement learning, the world in which the agent lives is assumed to be in one of a set of perceivable states. The objective is to learn the long-term value (i.e. *fitness*) of each state-action pair. The main approach taken is to probabilistically explore states, actions, and their outcomes to learn how to act in any given situation. Unfortunately, while this approach learns *well*, it does not learn *fast*. For example, the system TD-Gammon [Tesauro 1995] taught itself to play Backgammon at a master's level. However, to learn this, it had to play two million games against itself! While it is practical for a software program to play against itself this many times offline, it is impractical to expect a human to spend so much time interacting with it. Indeed, human patience is the most critically scarce resource in interaction-based adaptation. There are further challenges in using traditional reinforcement learning for interactive adaptation, such as requiring discrete state and action spaces, and so on.

Thus far we have discussed example- and experience-based machine learning, the most common approaches. There are other machine learning approaches, for example *planning-* and *observation-based*, which could be considered for use in character adaptation. For example, planning through a tree search [Sutton and Barto 1998] to compute the long-term value of each state-action directly, rather than waiting for the value to backup using local updates. This places the burden of learning more on processing power than continually repeating experiences, so learning can occur far more quickly with respect to human interaction. However, planning requires a complete model of the character's environment, including all agents therein—but we do not have a model of the human user's behavior.

Observation-based learning appears especially applicable to our needs in interactive character adaptation. These techniques include agent/user modeling [Gmytrasiewicz and Durfee 2000; Kerkez and Cox 2003] and imitation [Price 2002]. In agent/user modeling, the agent learns to predict the actions of someone else and can thereby predict the utility of any behavior it may engage in. In imitation, the agent memorizes the behavior of another agent or user and mimics it. These observation-based approaches to machine learning are interesting because they are natural for online use. However, existing techniques are limited to discrete state/action spaces, or do not learn quickly from few observations and so on.

Therefore, while current approaches in machine learning provide us with a solid theoretical foundation to build on, existing techniques do not solve our problem of interactive adaptation for virtual characters.

4. MAKING ADAPTATION PRACTICAL

4.1 Requirements

While machine learning provides a theoretically sound basis for building systems that learn, there are a number of issues that make existing techniques problematic in the context of interactive adaptation for autonomous animated characters. We have identified the following characteristics that are necessary for an adaptation algorithm (for cooperative or competitive autonomous characters) to be practical and useful, listed in order of importance:

1. Fast learning. Human time and patience is the resource we will certainly have the least of. Further, slow learning will not make a character keep up with a fast-learning human. Therefore, adaptation must occur quickly based on few experiences.

2. No explicit human feedback. To achieve adaptation for truly autonomous cooperative or competitive relationships, we cannot ask the human to supply detailed feedback on the character's every action (this denotes a master-slave relationship, and could interrupt the flow of an animation). Therefore, the adaptation must occur without any explicit human feedback, just natural feedback from the environment.

3. Believable adaptation. To be convincing, adaptation must appear intelligent. That is, when the character learns, it should usually (or always) become better at its task.

4. Must perform at interactive rates on a PC. For adaptation to be as widely useful as possible, it must be practical for interactive use on current PC CPUs.

4.2 Assumptions

In order to achieve the above listed goals, we make the following assumptions. These assumptions, however, are valid within our theoretical foundation and do not impose any important limitations:

1. Knowledge acquisition is sufficient for adaptation. We assume that our character already possesses adequate motor control, perception, and decision-making skills. Therefore, to optimally interact with a unique human user, all that it needs to do is gather some key knowledge with which to guide its decision making.

2. Natural-looking learning is sufficient. Our adaptation technique does not need to be as powerful as traditional reinforcement learning. We know from ethology that Nature places a premium on learning adequate solutions quickly.

3. Insight is used to accelerate adaptation. It is well known that insight by the programmer into what the character must learn can greatly simplify the learning process. We assume that the programmer provides such domain knowledge in the form of a good compact state definition, and a *gradient fitness function* [Kaelbling et al. 1996] (i.e. reward is given for approaching goal states).

4. The current state and the user's action are observable. All of our learning methods utilize the current state of the environment, and some utilize the human user's actions. Thus the current state and user's actions must be observable. This assumption is reasonable in our problem domain because the low-level actions of the human user are explicitly input through a device such as a joystick. Also, the virtual environment exists in its entirety in software and therefore is perfectly observable (with the exception of the user's internal state which we infer as discussed later in the article).

5. SYSTEM DESCRIPTION

We now give a detailed description of our multi-level technique for fast character adaptation. First, note that we start with a non-user-specific behavioral model that is constructed offline. This model provides the baseline behavior for the character. It is this model that we adapt online so that the character will better interact with a unique human user.

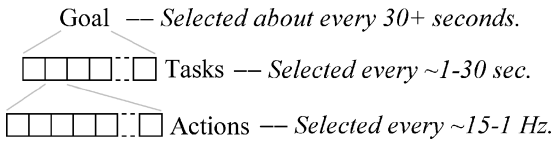

Fig. 3. Relationship between *goals*, *tasks*, and *actions*. These are the names we apply to these temporal levels of decision-making.

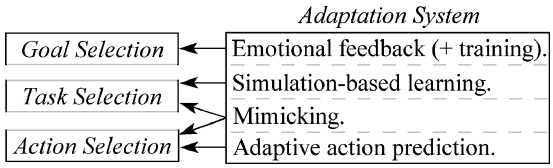

Fig. 4. Overview of our adaptation system. Individual learning methods are applied to the layers of a behavioral model.

In our technique, we fulfill the requirements listed in Section 4.1 by using a combination observation-, experience-, and planning-based approach to machine learning. Moreover, each layer of the behavioral model is treated differently; the specific approach to learning used in each layer is unique to the needs and temporal constraints of that layer. In fact, it is the temporal granularity of decision-making in each layer that primarily dictates what approach to learning will be most effective. We will discuss this in more detail later.

There are no universal names for the decision-making layers of behavioral models. For clarity, we use the following names (see Figure 2): *action selection*, *task selection*, and *goal selection* (in order of fine to coarse temporal granularity). This relationship is demonstrated in Figure 3. An *action* is a very low-level decision, which can be directly translated into motor control. A *task* is somewhat more high level, requiring several actions to perform. A *goal* is the highest level, representing the character's strongest current desire, and can be broken up into several tasks. The current goal loosely determines the tasks to perform, and so on.

An overview of our adaptation system is given in Figure 4. As mentioned previously, it is composed of a set of discrete learning methods that are applied to individual layers in an existing behavioral (reactive) or cognitive (deliberative) model. Note that if a given model has fewer than three layers, an appropriate subset of our learning methods can be used.

5.1 State and Action Representations

For any given character and virtual environment, our state space may have to be continuous. This is because 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 smooth and aesthetically pleasing character animation. Therefore, our technique uses a continuous internal representation of states and actions. Since this is more general than a discrete space, both representations are naturally supported.

We represent the state space as a real-valued, n-dimensional feature vector. Thus a state $\mathbf{s} \in \mathbb{R}^n$ is a point within this feature space.

A *feature* is some salient aspect that characterizes the state of the world. Usually, even complex worlds can be effectively represented with a compact set of features. In fact, there are known techniques to automatically discover salient features, for example principal component analysis [Mitchell 1997]. Alternatively, human intuition can be applied to this problem. As was stated in Section 4.2, we assume that the programmer has provided a good, compact state representation for the given character/world. This is important because a compact state space will help the character adapt quickly and better generalize what it has learned. For more information on selecting a compact set of features, see Reynolds

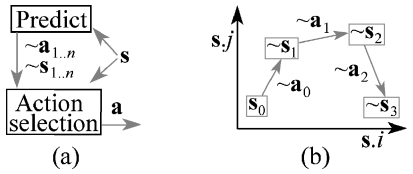

Fig. 5. (a) Structure of our low-level adaptation method. An n-step sequence of states and actions into the future is predicted, then action selection uses this information to make wiser choices. (b) To predict the human's actions n steps into the future, we predict/compute the actions of all agents in the environment, determining a new state, and then repeat. The tilde notation signifies that the predictions are approximate.

[1987] or Grzeszczuk et al. [1998]. As an example, in our 1-on-1 rugby case study, the state is composed of the translation-invariant separation of the two characters, and their velocities.

Like states, actions are real-valued vectors, $\mathbf{a} \in \mathbb{R}^m$, so that both discrete and continuous action spaces are supported. Actions should be organized in such a way that they can be combined into some sort of "intermediate" action (e.g. 'left' and 'forward' become 'diagonal').

It is important, for the sake of generalization, that our real-vector-valued states and actions be organized such that similar states usually map to similar actions. More formally:

$$(\|\mathbf{s}_1, \mathbf{s}_2\| < \varepsilon_a) \Rightarrow (\|\mathbf{a}_1, \mathbf{a}_2\| < \varepsilon_b),$$

where $\|\ \|$ is the l2-norm, and ε_a and ε_b are small scalar thresholds. Certainly, this constraint need not always hold, but the smoother the relationship the simpler it will be to learn.

If our adaptation technique is to be integrated into an existing behavioral animation system, it may be necessary to transform states and actions into our internal real-vector-valued representation. This can be performed through a simple, custom transformation:

$$T_{\mathbf{s}} : \texttt{state} \rightarrow \mathbf{s} \in \mathbb{R}^n, \quad T_{\mathbf{a}} : \texttt{action} \rightarrow \mathbf{a} \in \mathbb{R}^m, \quad T_{\mathbf{a}}^{-1} : \mathbf{a} \rightarrow \texttt{action}$$

where teletype signifies the external format of states and actions.

While we assume in this article that the programmer has provided an effective and compact state space, there are several techniques available for automatic state space discovery (for example Blumberg et al. [2002]; Guyon and Elisseeff [2003]). Our motivation for using explicitly designed state spaces is that they have often proven superior in machine learning experiments reported in the literature. Nevertheless, better results may be achieved through automatic techniques when the programmer is inexperienced with designing compact state spaces.

5.2 Low-Level Learning (for Action Selection)

The primary challenge faced in performing low-level adaptation (i.e. learning in the action selection layer) is that, with the decision-making being so temporally fine-grain, it is impossible to quickly learn long-term values of state-action pairs. However, due to this fine temporal granularity, it is easy to observe the human's behavior in the form of state-action pairs. This is possible because the human's actions are explicitly input using a device such as a joystick, and the state is entirely observable because the virtual world exists in software (with the exception of the human's internal state which we infer as discussed below).

Therefore, we take an observation-based approach to adaptation for action selection. Specifically, through observation, we can construct a Markovian model of the human's behavior. We can then use this model to predict the human's future behavior, and thereby our character can more wisely select actions to perform (see Figure 5).

In our action selection adaptation method, the behavior of the human user is recorded online in the form of state-action pairs. 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 state-action pairs. For example, in our rugby case study, the time step matches the frame rate of the animation (15 Hz).

The model of the human's behavior is constructed through case-based learning. Each recorded state-action pair is treated as a case in a case-based reasoning engine. A library is maintained of useful cases. Since the state space is continuous, the library is organized as a hierarchal partitioning of the state space. Partitioning is important so that fast lookup of cases can be performed. Automatic partitioning techniques (e.g. a kd-tree) can be used to great effect. Alternatively, partitioning can be performed by the programmer so that human insight may be applied.

To predict the human's future behavior, the library of cases must be generalized so that, for any given query state, an associated action is computed. To fully exploit our knowledge of the human, we generalize through the *continuous* k-*nearest neighbor* algorithm. 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:

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

The tilde notation signifies that the answer is approximate. 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. Note that it is helpful to normalize the axes of the state space, so that they will contribute equivalently in computing distance.

Alternatively, to focus on caution rather than exploitation, we generalize using a custom modified minimax search. The k cases closest to the query state are found. Then, the k actions associated with the retrieved cases are tested with the character's own fitness function. Finally, the action that results in the minimum fitness is assumed to be the one the human will select. More formally:

$$\tilde{\mathbf{a}} = \arg \min_{\mathbf{a}_i \in k \text{ cases}} (\text{fitness}(\mathbf{s}, \mathbf{a}_i)).$$

For this cautious generalization, we prefer $3 \leq k \leq 16$. The greater the value of k, the more cautious the generalization will be.

It is important that we predict the human's actions several steps into the future, so that our character can make wise decisions. To do this, we first predict the human's action for the current time step, then we either compute, predict, or assume the actions of all other agents in the virtual world. This allows us to predict the future state, which in turn allows us to predict the next action taken by the human, and so forth. We have found that predicting between 5 to 15 steps into the future works well, is accurate enough to be useful, and usually requires little CPU time.

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 $(\mathbf{s}, \mathbf{a})_i$ are allowed for each region r_j of the state space. 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 to be added, it is likely to be removed. Thus the character has the ability to "forget," which is very important in learning something as nonstationary as human

behavior. We formally define the case replacement as:

$$\text{replace} \quad \arg\min_{(\mathbf{s},\mathbf{a})_i \in r_j} (M(\mathbf{s},\mathbf{a})_i) \quad \text{with the currently observed case} \quad (\mathbf{s}_t, \mathbf{a}_t),$$

using a metric M:

$$M((\mathbf{s},\mathbf{a})_i) = -\alpha \cdot \text{age} + \beta \cdot \|(\mathbf{s},\mathbf{a})_i, (\mathbf{s}_t, \mathbf{a}_t)\|.$$

Human decision-making and behavior is nondeterministic. Therefore, it is critical that our action prediction technique properly handle nondeterminism. Because we explicitly store discrete state-action cases, nondeterminism can be represented in our case library. Both of our approaches to case generalization properly handle nondeterminism, but in different ways. k-nearest neighbor is less tolerant of nondeterminism than minimax, since conflicting cases can average out to be a null action. However, in situations of greater case homogeneity, k-nearest neighbor produces accurate predictions of average behavior. In contrast, our custom minimax always properly handles nondeterminism, since it merely searches for the action that will most damage the character's fitness.

The reason our observation-based approach to low-level adaptation is sufficient is because the character learns all the nonstationary knowledge it needs to wisely select actions. By accurately predicting the human's behavior, the character can predict the results (i.e. utility) of its actions and can thereby wisely select what to do.

An alternative way to use action prediction (rather than predicting an entire sequence of actions given an initial state, as in Figure 5) is for the behavioral/cognitive model to request individual predictions for specific states. This can be especially useful for a cognitive model, as it can request information specific to any state it encounters while deliberating. However, this requires more CPU than a single linear prediction.

Of course, the human user's decision making will vary based on her current goal. As a result, it can be useful to employ multiple state-action case libraries, one for each goal. As discussed in works such as Blumberg et al. [2002] and Evans [2002], the human's goal can be easily inferred because the virtual environment constrains what she needs to accomplish.

5.2.1 *Relationship To Previous Work.* Our action prediction method is related to other agent/user modeling techniques, such as *Case-Based Plan Recognition* (CBPR) [Kerkez and Cox 2003]. CBPR fundamentally operates like our method, using state-action cases. However, CBPR is limited to discrete state/action spaces, does not fully generalize cases, and can only predict an agent's behavior one action into the future. Moreover, CBPR always assumes the closest case in the state space is correct, whereas our technique can exercise caution or confidence when predicting actions. In fact, our technique can even predict the Nash equilibrium strategy if the proper information is available in the local cases. Another related technique is *Maximum Expected Utility* (MEU) [Sen and Arora 1997], which is a modification of minimax. However, MEU is limited to discrete state/action spaces, and can require significant CPU (exponentially increasing) to predict behavior several steps into the future.

In Gmytrasiewicz and Durfee [2000], a hierarchical approach to agent modeling is used to produce coordination between software agents. This technique assumes that all agents desire to cooperate, and that payoff matrices are sufficient to model behavior. Thus this technique is limited to producing only cooperative behavior, and is limited to discrete state/action spaces. Moreover, this technique is likely not plausible for interactive use, because it creates a tree-like nesting of models, which can require significant storage and processing power. Our action prediction method contrasts with hierarchical techniques like Gmytrasiewicz and Durfee [2000] because we use a single "flat" model, which is updated to represent recently observed behavior and forget older behavior. Our method has theoretical underpinnings in *Fictitious Play* theory [Stone and Veloso 1997].

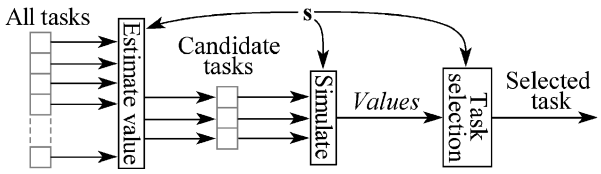

Fig. 6. Structure of our mid-level learning method. The values of all tasks are estimated for the current state, then promising tasks are evaluated more accurately through simulation.

Another related technique is Zhu et al. [2003], an example of user modeling. In this technique, the system learns to predict the activities of a user interacting with a web browser. Like this technique, most other user modeling techniques are focused on interaction through GUI interfaces. The fundamental assumption is that the agent-user interaction takes place through a constrained interface. Thus they are not appropriate for our problem domain of graphical characters interacting directly with a human user in a virtual world.

We believe that our action prediction method may be very applicable to problems outside of interactive virtual character adaptation, since it has unique strengths as compared to existing techniques. However, an examination of these uses is outside the scope of this article, and therefore left to future work.

5.3 Mid-Level Learning (for Task Selection)

The challenges faced in performing mid-level adaptation are different than those in low-level adaptation. This is primarily due to the fact that the temporal granularity of decision-making is now more coarse. Specifically, action selection is performed quite often (about once every 15 to 1 Hz), while task selection is more seldom (about once every 1 to 30 seconds).

The primary challenge faced in mid-level learning is that we cannot with certainty observe the human's behavior in the form of state-task pairs. This is because task selection is at a conceptually high enough level that it is likely to involve a large amount of hidden state information inside the human's brain (i.e. it is non-Markovian). As a result, it is impossible to determine exactly what motivated the human to select tasks as she did. Therefore, we cannot take a direct observation-based approach here like we did for action selection (low-level learning). Moreover, we also cannot easily take an experience-based approach for task selection, as this can take a long time to learn.

Our approach to mid-level adaptation is primarily planning-based, but also involves experience- and observation-based reinforcement learning. The key is to be able to run simulations (i.e. plan) to determine which candidate task will most likely perform best for the current state. As mentioned in our review of machine learning, we can simulate the outcome of any decision if and only if we have a complete model of the environment, including a model of the human's behavior. Luckily, through our approach to adaptation we already have a model of the human's low-level behavior, constructed during low-level adaptation. We reuse this model here, to run simulations.

Our mid-level adaptation technique involves the following steps, as shown in Figure 6:

(1) **Estimate value.** Compile a small set of candidate tasks that have the highest estimated values (utilities) for the current state.

(2) **Simulate.** Run internal simulations for each of the candidate tasks, to more accurately measure their utility.

(3) **Select.** Rank candidate tasks according to their utility. Once this is done, the behavioral model then selects one.

The reason we use Step #1 rather than proceeding straight to Step #2 is because simulating all possible tasks may be implausible. Therefore, to limit the set of candidate tasks to a practical number, we need

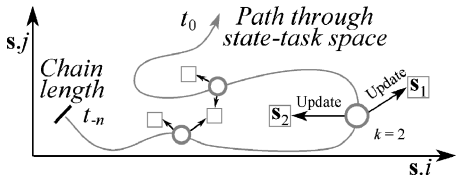

Fig. 7. To know which state-task values to update, we store the path traversed by the character. Then, after receiving feedback, we update the k cases closest to each state-task visited.

rough approximations of each task's value so that poor candidates can be eliminated early. To do this, we use case-based libraries of state-task values, one library for each task. These libraries perform a state $\rightarrow$ value mapping: determining for any given state the utility of selecting a given task. The libraries are structured and evaluated similar to those in low-level adaptation, using k-nearest neighbor. However, the case population of the state-task libraries is much sparser (maximizing learning speed) because we only need rough approximations; we will run simulations to provide more accuracy as needed. The state space can be uniquely defined and/or partitioned for each state-task library, if desired. Also, the libraries are originally initialized with regards to generic human behavior.

Credit assignment is performed to update the state-task values both after running simulations (according to their predicted utility) and after selecting a task (according to feedback from the environment). Updating after a simulation is simple, as we know exactly what state-task value should be updated. However, updating due to feedback is more difficult, as feedback is usually delayed. We could use a traditional local update rule from reinforcement learning, but this is too slow. Instead, as shown in Figure 7, we maintain a chain of the previous n state-tasks visited by the character. The value of every state-task in the chain is updated every time new feedback is received. The longer the chain, the more accurate to long-term utility the state-task values will be. The actual update of the case-based library of values is performed by updating the existing cases closest to the actual case we wish to update; no new cases are added nor are any removed. More formally:

$$\text{value}(\mathbf{s}_j,\ task)' = \text{value}(\mathbf{s}_j,\ task) + \alpha \cdot (\tilde{}value - \text{value}(\mathbf{s}_j,\ task)),\quad \forall\, (\mathbf{s}_j,\ task) \in k \text{ neighbors},$$

where $\alpha \approx 0.5$ is the scalar learning rate, and $\tilde{}value$ is the apparent state-task utility. The maximum aggregate change of a case is bounded for each time it is visited (e.g. 40% of its possible range).

The internal simulations are run for a reasonable amount of time into the future. We have found that a few seconds in the character's time frame usually works well. During the simulations, the human's actions are predicted using the case-based model constructed during low-level learning. If there are other computer-controlled characters besides the one currently performing task selection, their actions can either be computed, predicted, or assumed. Surprisingly, we have found that it usually is sufficient to assume constant action (e.g. the character's last performed action). It can also work well to compute only one new action for every several time steps. Such assumptions can help speed up the simulation for more computationally complex behavioral or cognitive models.

The apparent utility of the simulated and/or executed task is computed as the average of all feedback received during the simulation or execution:

$$\tilde{}value = \sum_{t=t_0+1}^{t_0+n} \text{fitness}(\tilde{}\mathbf{s}_t),$$

where t_0 is the current time step, and n is the number of steps to simulate into the future. No emphasis is given to early or latter time steps. As mentioned in Section 4.2, a *gradient fitness function* is used, meaning that feedback should be received for most/all states visited. While strong feedback is given for reaching a goal or terminating state, weak feedback is given for intermediate states. This gradient

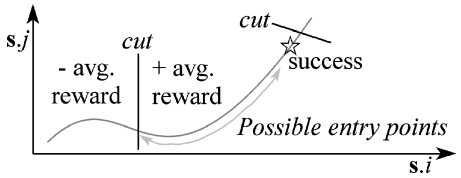

Fig. 8. Overview of our mimicking technique. The character is interested in sequences of actions where the human's situation improved overall (i.e. positive average reward), culminating in achieving her goal. This action sequence is recorded and later mimicked.

fitness function, if designed properly by the programmer, can help lead even short simulations to optimal long-term utility.

Our mid-level learning technique can take time to compute, especially if the character's behavioral/cognitive model is computationally complex. When necessary to maintain interactive performance, it is possible to simply select a task using the case-based state-task values. Note that task selection does not need to be performed between animation frames—the simulation process can be spread out over several frames, as long as the current state does not vary too greatly.

5.3.1 *Relationship To Previous Work.* Our case-based approach to learning state-task values has a solid foundation, as local function approximation is currently considered to be the best approach for learning value functions [Sutton and Barto 1998]. The aspect of our task-selection adaptation method that is most novel is the combination of experience- and simulation-based machine learning concepts. In fact, we are not aware of any existing techniques that use a similar method. By taking a combination approach, we are able to achieve sufficiently accurate results with reasonable CPU use and storage requirements (see the results section). In comparison, traditional reinforcement learning forces an agent to experience all state-task pairs multiple times to learn their values.

Another interesting aspect of our approach is that it is a practical realization of using deliberation to generalize sparse learning, a relatively unexplored direction in machine learning that has recently seen a great deal of interest [Yoon 2003]. In fact, it is currently hypothesized by many cognitive scientists and engineers that deliberation is a necessary ingredient to achieve human-like learning (see the proceedings of DARPA 2003 Cognitive Systems Conference for more information on this topic).

5.4 Mimicking (for Action and Task Selection)

In our system, *mimicking* (i.e. imitation) is another learning method for low- and mid-level decision making. It compliments the learning methods already presented by gathering knowledge in a different fashion. While less general than our other learning methods, it is very fast and can easily encapsulate very complex behaviors and decision-making.

Mimicking provides the character with the ability to quickly learn novel behaviors in a clever and natural, yet indirect way. Moreover, creating truly novel behaviors is difficult, whereas mimicking is far simpler to perform and is highly likely to improve the performance of a character. Another interesting aspect of mimicking with regards to interactive virtual environments is that it leverages human intelligence in a non-intrusive, intuitive way. While the human user interacts with a challenging, adapting environment, she will be forced to adjust her behavior and tactics. Thus she will attempt possibly novel behaviors. The character can observe these new behaviors and, based on their apparent success, decide to remember them for later imitation when faced with the same situation as the human was.

Our mimicking technique is summarized in Figures 8 and 9. In our system, our mimicking method affects both the task and action selection layers. This is because the observed novel behavior, which the

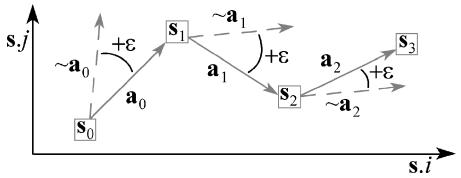

Fig. 9. To determine if an observed action sequence is novel to the character, we compute the average difference, ε, between the recorded actions and those that the character would normally select.

character wishes to mimic in the future, is stored as a new task. If this new task is selected, it overrides action selection by forcing the recorded chain of actions to be performed as they were observed. Each recorded observed behavior is stored as its own task, and has individual candidacy to be selected for execution by the behavioral model.

Since the novel observed behaviors are treated as tasks, we must associate state-task values with them to determine when they should be simulated during task selection (see Section 5.3). We do this by creating a new case library for each recorded behavior, with state-task value cases for the states visited by the human while executing this behavior, plus a sparse set of additional cases that are regularly positioned throughout the remainder of the state space. Those states that were visited during the human's execution of the behavior are initialized with "good" values, while all other regions of the state space are given "poor" values. These state-task values are thereafter updated just like those of standard tasks. If the behavior fails too often (e.g. 3 times consecutively), it is assumed to be entirely ineffective and is deleted.

An important question is how to determine the beginning and ending of the novel observed behavior, since there is a continuous stream of observed state-actions. As shown in Figure 8, we do this by keeping all state-actions where an *overall* improvement toward achieving the human's goal was observed (as measured using the character's own fitness function). In other words, we keep the subsequence of state-action pairs starting with the global fitness minimum in the entire sequence, and ending with the pair where the human achieved her goal. Thus our retained subsequence has the property:

$$\text{fitness}(\mathbf{s}_0) \leq \text{fitness}(\mathbf{s}_i), \quad \forall i,$$

where $\mathbf{s}_0$ is the global minimum (i.e. first state in the retained state-action subsequence). Later, when performing simulations during task selection, we allow the entry point into the recorded sequence to vary depending on the current state of the virtual world $\mathbf{s}_c$:

$$\text{entry point} = \arg \min_{\mathbf{s}_i \in \text{sequence}} \|\mathbf{s}_c - \mathbf{s}_i\|.$$

To determine whether an observed behavior is novel to the character, and therefore of interest for mimicking, the recorded state-action sequence must be compared to the decision making in the action selection layer of the character's behavioral model. As shown in Figure 9, we do this by performing action selection for a random subset of the states in the recorded state-action sequence. We then compute the average component-wise difference, ε, between the recorded actions and those selected by the behavioral model:

$$\boldsymbol{\varepsilon} = \left(\sum_{j=0}^{n-1} (\mathbf{a}_j - \tilde{\mathbf{a}}_j) \right) / n.$$

$$\text{if } \|\boldsymbol{\varepsilon}\| \geq \xi, \text{ then } \texttt{Novel}, \text{ else } \texttt{Known}.$$

We have found that $\xi \leq 20\%$ of the max possible error is effective.

Note that for mimicking to work, the observing character must be able to infer the human's goal. In interactive virtual environments, this is usually trivial, since the goal is largely determined by the high-level situation of the virtual world. For example, in our rugby case study, the general goal can be determined by which team has the ball. Obviously, the team with the ball will attempt to score, while the other team will try to stop them. This general, high level understanding of the human's goal is sufficient. For example, if the human's avatar succeeds in running around the opposing team's characters and scoring, then the human's behavior in that situation is a candidate for mimicking by the character when it has the ball.

The character can generalize observed behavior if similar enactments are observed and recorded. This generalization can be done either by performing a weighted blending of two or more recorded action sequences, or by unexpectedly switching from one sequence to another. Similarity between recorded behaviors is determined by attempting to overlap some portion of the recorded state-action sequences by computing the mean component-wise difference between the actions.

Since we assume that behaviors begin with fitness minima, our mimicking method cannot learn all interesting behaviors. For example, there are effective strategies where the human could purposefully do something of poor fitness to misdirect the character. In such situations, it is likely the character will only learn the portion of the behavior after the misdirection. Nevertheless, in our experience most valuable novel behaviors do not directly violate the character's fitness function, and thus can be learned effectively by our mimicking method.

5.4.1 *Relationship To Previous Work.* There has long been interest in teaching synthetic agents through observation or demonstration. This is because learning through demonstration is a natural form of instruction used in the real world by humans [Meltzoff and Moore 1992; Yoon 2003]. It is widely believed that demonstration is one of the most natural and effective human-computer interfaces possible, especially for a nontechnical user. However, developing an effective and general demonstration interface has proven elusive.

Blumberg et al. [2002] allows a human user to interactively train a virtual character (e.g. a graphical dog). This technique allows the user to lure the character into a certain position, indirectly demonstrating desirable poses. However, while effective at teaching some aspects of motor control, this technique does not allow teaching of decision-making to achieve tasks and goals. Therefore, this technique does not fulfill our needs. van Lent and Laird [2001] present a programmer-oriented technique for learning through demonstration. A programmer must define and implement all *operators* (i.e. actions or tasks) that a character can perform. The post- and pre-conditions of these operators are then learned through demonstration (with explicit annotations provided by the programmer after the demonstration). This technique has proven effective but is not automatic, and therefore cannot be used for online adaptation. Kasper et al. [2001] presents a technique for teaching a robot simple navigational behaviors. However, this is too limited for our needs. In Price [2002], one agent guesses state-action values by observing the behavior of another agent. While this allows for fine-grain learning, it is not significantly faster than traditional reinforcement learning and is limited to discrete state/action spaces.

In contrast to these previous techniques, our mimicking method is less general, but is fully automatic, learns quickly, and can easily encapsulate very complex behaviors. The most novel aspects of our mimicking method include automatic detection of novel behaviors, and the use of a fitness function to automatically determine which novel behaviors may be valuable to imitate.

5.5 High-Level Learning (for Goal Selection)

It is at the level of goal selection that some previous master-slave interactive behavioral learning techniques have been applied. This is because the decision-making is temporally coarse grain and

```
fitness_tackler(P^U, P^C)
{
     const float PLAYER_SIZE = 0.4;
     float dist = ||P^U − P^C||;
     if (P^U_y > P^C_y + PLAYER_SIZE) {
         /* User has passed character, so she can score easily. */
         return (-100 - dist);
     }
     else if (dist < PLAYER_SIZE) {
         /* Close enough to tackle. */
         return (500);
     }
     else {
         /* Character is in user's way (where it should be). */
         return (100 - dist);
     }
}
```

Fig. 10. Pseudo-code of our gradient fitness function for a rugby character that will rush and attempt to tackle the human user. P = "position", U = "user", and C = "character". This fitness function specifies that the character should get as close to the user as possible without falling behind him (and thereby allowing the user to score). The character's cognitive model automatically determines how to behave to maximize long-term fitness. This function can also be used to detect valuable behaviors to mimic by measuring the fitness of the human user's own tackling behaviors.

thus it is plausible for the human to provide timely feedback. Unfortunately, in many cooperative and competitive situations, it is unnatural for the human to provide explicit feedback to the character. We leverage a different type of feedback: *emotion*.

In nature, emotional feedback plays a heavy role in the formation of personality [Matthews 1997]. For example, it is well known that humans will more often engage in activities that make them happy. In algorithmic terms, this means that the desirability (value) of each goal is updated based on the happiness of the character as a result of performing it.

Emotion is often implemented in a behavioral model (as in Tu and Terzopoulos [1994]; Tomlinson and Blumberg [2002]; Egges et al. [2004]) as a small set of variables, for example: `Happiness`, `Fear`, and so on. The character's current emotional state is the combination of all these variables. In our method for high-level adaptation for goal selection, we are only concerned with the change in happiness. There are many possible events that can affect a character's happiness. One of the most well known is success or failure. Note that emotion-changing triggers are part of the behavioral model, and have been examined in previous works, so we do not dwell on them further here. The fact that the character's emotional state changes is sufficient for our needs.

In our emotion-driven adaptation method, we use a highly abstract representation of the character's state: e.g. "`Hungry` + `FoodNearby` → `Eat`". We determine the value (i.e. predicted resulting happiness, ˜`Happiness`) of a goal by maintaining a set of weights which define how important each high-level feature of the abstract state is for a certain goal. The value of the goal for the current state is then computed as the weighted sum of the state features using a linear perceptron:

$$\tilde{}\texttt{Happiness} = \sum_i (\mathbf{s}.i \cdot w_i).$$

In other words, if features `Hungry` and `FoodNearby` are high, and the associated weights are high, then the goal `Eat` will have a high value. The weights are updated once, either when a new goal is about to

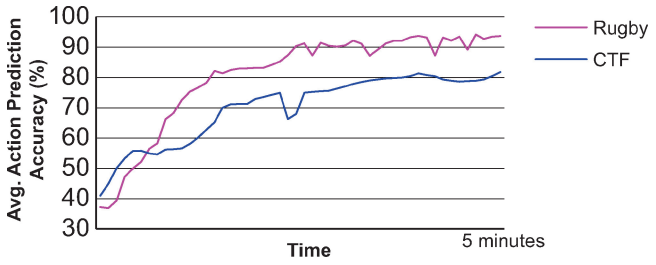

Fig. 11. Accuracy of predicting the human's actions (L2-norm). This demonstrates the accuracy of both low- and mid-level learning in our system. This experiment started with the character having very incorrect information about the human user.

be selected or after the current goal has been active for a sufficient amount of time (e.g. 30 seconds). We update the weights using gradient descent, based on the state under which the goal was selected, to more accurately be able to predict the resulting happiness in future iterations:

$$w_i{}' = w_i + (\gamma \cdot \mathbf{s}.i \cdot (\texttt{Happiness} - \tilde{}\texttt{Happiness})), \quad \forall\, i,$$

where $\gamma \approx 0.5$ is the scalar learning rate, $\tilde{}$`Happiness` was the predicted happiness, and `Happiness` is the actual resulting happiness. This allows the character to not only adjust under what circumstances a goal is desirable, but also to adjust the magnitude of the desire.

An interesting result of our approach is that similar characters can develop widely varied personalities, depending on their experiences. In fact, a character's personality may diverge from the "optimal" personality with respect to its assigned role in a virtual world. This is because an unlucky character may repeatedly fail at a goal that usually would be accomplished and thereby learn to avoid that goal. This type of learning has an interesting parallel to phobias in real humans. If desired, such divergence can be avoided by either not using our goal-level adaptation method or placing range limits on the weights in the linear perceptron.

5.5.1 *Relationship To Previous Work.* Our goal-level adaptation method is very similar to previous work in computer animation for character training [Blumberg et al. 2002; Evans 2002], as well as methods for learning from change in emotional state [Tomlinson and Blumberg 2002; Gadanho 2003]. The novel aspect of our approach is the use of emotion to update personality within a multi-level framework that also provides learning for nonreactive behavior. We are not aware of such a combination in the literature.

Our use of a linear perceptron to predict the value of a goal is similar to Evans [2002]. This approach has proven effective, both in our own experiments and in Evans' work. However, since the function approximation is linear, there can only be one contiguous region of the state space where a certain goal is likely to be selected. However, since we use a highly abstract representation of the state space, this should usually not be a problem.

Note that our adaptation technique for cooperative/competitive relationships can coexist nicely with previous techniques for master-slave relationships. This is because a character may have a cooperative/competitive relationship with some humans and/or characters, and a master-slave relationship with others. For example, to integrate our work and Evans [2002], both emotional and human-user feedback could be used to train the perceptron.

5.6 Using Adaptation in Practice

The accuracy of the learning in our system has proven to be very promising (see Figures 11 and 12 in the results section). Also, the performance is well within interactive speeds (see Table I), and it has a small memory footprint (usually $\leq$ 2 MB).

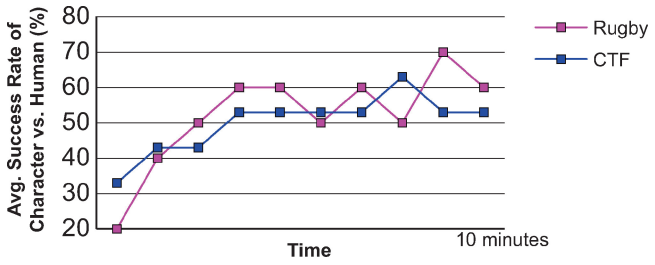

Fig. 12. This graph demonstrates the effectiveness of our adaptation system as a whole. For example, in the rugby case study, how often the character scores compared to how often the human user scores.

Table I. Typical Performance Results of our Adaptation System in our Three Case Studies (for One Given Adapting Character). We Used a 1.7 GHz PC with 512 MB RAM

	Rugby	*CTF*	*Camera*
Action prediction time	30 μ sec	33 μ sec	24 μ sec
Simulation time	28 ms	42 ms	N/A
Total avg. CPU usage	7%	12%	35%

Recall that the function of our adaptation system is to supply a behavioral/cognitive model with supplementary information. Therefore, the way this information is used can be unique for any given behavioral/cognitive model. For example, given an n-step prediction of the human's actions into the future, the character 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 model, even a black box implementation. The information can also be used to cooperate with or compete against the human, as the character wishes.

Our adaptation technique is not limited to small environments with only one human. Indeed, it can operate in 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 is because this circumvents the curse of dimensionality, thereby allowing our adaptation technique to be used for interesting, difficult problems. If necessary, it can even be useful to aggressively approximate the current state—even though this limits the accuracy of state information, and thereby limits the potential accuracy of the character's learning, it makes the dimensionality tractable. By keeping the state space small, we have successfully applied our adaptation technique to very complex characters/environments.

To further counteract the curse of dimensionality, we have found it useful to modularize the adaptation when possible. For example, consider a character 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. Modularization is especially useful for our action prediction method, as it is the most sensitive to the curse of dimensionality of all our learning methods.

In some circumstances, we have found it useful to share acquired knowledge between all adapting characters in a given animation. In other words, we only use a single repository for acquired knowledge, which the adapting characters share. This is useful for reducing storage requirements, as well as allowing every character to behave optimally according to what has been learned.

It is important to point out that, while our adaptation technique does change the behavior of a character, it only does so within bounds set by the behavioral/cognitive model. That is, since our adaptation

technique only supplies supplementary information, the behavioral/cognitive model is still in full control of decision making. This feature of our technique is important for stability, and maintaining animator goals.

Each of our learning methods provides a unique degree of adaptation for a character. We have determined the individual usefulness of each learning method by applying only one learning method at a time in our rugby case study. Action prediction (low-level learning) is actually the most powerful of all our methods. This is because low-level decision making is most critical in achieving overall life-like and effective behavior since actions are the only decisions directly performed by the character. The second most powerful of our learning methods is mimicking, as it allows a character to rapidly learn novel, complex behaviors that affect both the action and task levels. The third most powerful is mid-level learning for task selection and the least powerful is high-level learning for goal selection.

In some circumstances, it may be desirable to only use a subset of the learning methods presented in this article (i.e. not apply learning to all layers of a behavioral/cognitive model). This is because, as detailed in the previous paragraph, each learning method provides a different degree of benefit. Also, each method can require a notable amount of CPU to execute (except goal-level adaptation). Moreover, each learning method must be integrated separately into a behavioral/cognitive model, which can be a time-consuming undertaking. If a subset of methods is to be used, we recommend choosing methods according to the usefulness order given in the previous paragraph.

Recall that we utilize a gradient fitness function in our system to guide the character's behavior. It is beyond the scope of this article to detail methods for creating fitness functions—but this problem has been thoroughly studied, thus we point the interested reader to the literature. Techniques to (semi)automatically create gradient fitness functions include *potential fields* [Reif and Wang 1999], *value iteration* [Sutton and Barto 1998], and interpolation of discrete state fitness labels. As reported in the literature, even complex problems can often be adequately represented with gradient fitness functions if they are properly abstracted (e.g. in motion synthesis for character animation [Sims 1994; Arikan et al. 2003]). In our case studies reported in this article (see Section 6), we have used explicitly programmed gradient fitness functions, one for each task.

As discussed previously, our learning methods for both action selection and task selection use case-based reasoning, where the case libraries are initialized with regards to "generic" human behavior. This initialization is worth discussing in additional detail here because early character behavior is crucially dependent on this data. These initial cases are created by training the character "offline" through interaction with one or more human users. In other words, the character gains its initial knowledge through learning to interact with a small set of users that is considered to be representative of the set of all possible users. Thereafter, the character need only adjust its knowledge to more effectively interact with a specific human user.

6. EXPERIMENTAL RESULTS

6.1 Virtual Rugby

Our first case study is of synthetic human players engaging in a sport such as rugby or American football (see Figure 13). This application of our adaptation system to virtual athletics is an interesting challenge; sports in general is known to be a very difficult environment for autonomous synthetic characters [Stone 2000].

In our case study, there is no explicit communication between the characters nor with the human user. Like the human user, the characters must rely on "visual" perceptions to ascertain the current state of the virtual world. Specifically, a character senses physical features such as its location, the

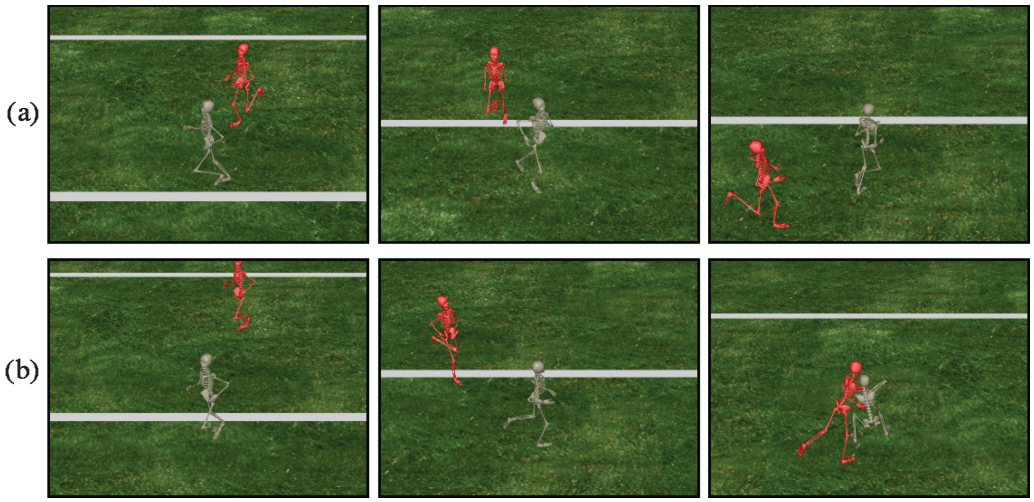

Fig. 13. (a) The human user (brown skeleton) performs a loop, which succeeds in getting past the character (red skeleton). As a result, the human can score. (b) Now that the character has adapted, the next time the human attempts a loop it predicts the human's actions and tackles him.

distance from it to other characters, velocities, and so on. Perception is performed by each character individually, and semi-realistic sensory honesty is enforced (i.e. a character can't see through the back of its head, etc).

The characters' and user avatar's motor control is performed through skeletal animation, based on a library of motion capture data. Specifically, there is a motion capture clip associated with each action a character may perform. These clips are blended together when necessary (using quaternion interpolation) to avoid jittering or discontinuities in the motion.

Action selection is performed at 15 Hz (once per frame), task selection once every 1 to 8 seconds, and goal selection once after every time a tackle or score occurs. The action space is composed of a continuous range of acceleration vectors, which represent the change in running velocity of a player. The human user controls his avatar through a joystick. The characters have several candidate tasks, such as to charge the user, cautiously wait for the user to approach, and so on. The only two goals are to score (if the character's team has the ball) or stop the user from scoring.

As detailed in Section 4.2, our adaptation technique assumes that the programmer provides compact state space definitions as necessary for each learning method. We now present the state space used for action prediction in this case study. For one-on-one rugby (one character against one human user), the compact state is defined as the relative (i.e. translation invariant) positions of the character and user, and their velocities. Thus the compact action prediction state space is six-dimensional:

$$\mathbf{s}_t = \left(\Delta x,\ \Delta y,\ V_x^U,\ V_y^U,\ V_x^C,\ V_y^C\right),$$

where U = "user" and C = "character". We ignore noncritical features, such as closeness to going out-of-bounds, to keep the space dimensionality low. This one-on-one rugby state space can be extended to support many-on-one rugby by including salient information about more than one character. We have found that fully representing the closest character, partially representing the second-closest character, and ignoring all more-distant characters works well.

We use one state space for all mid-level adaptation and mimicking. The state space is analogous to the one used for action prediction but is smaller, and is from the adapting character's frame of reference. It

Table II. Average Ratio of Tackles to Scores for the Ball-Runner Performing all Behaviors of Length 7 in a Simplified, Discrete Rugby Environment. Note that this is a Different Environment than the Continuous World Used in the Rest of our Rugby Case Study (as Shown in Figures 1 and 13). With no Learning for the Tackler Character, 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

is composed of the translation-invariant separation of the adapting character and user, and the user's velocity. Thus the task-level compact state space is four-dimensional:

$$\mathbf{s}_t = \left(\Delta x,\ \Delta y,\ V_x^U,\ V_y^U\right).$$

We purposefully made this space small so that approximate state-task value learning would occur quickly. When performing task selection, the simulations we run on the most promising tasks/behaviors provide us with sufficient accuracy. Since the simulations involve all characters in the virtual world, both one-on-one and many-on-one rugby work well with the same task-level state space.

The characters are controlled with a cognitive model, which performs decision making through a tree search (using A*). To make searching tractable, a discrete version of the action space is used for deliberation. We support varying goals and tasks by allowing a character's fitness function to vary. One of our fitness functions (for a character that is attempting to tackle the user) is given in pseudo-code in Figure 10. This is a gradient fitness function because a fitness is produced for every state, with fitness increasing toward the goal state (tackling the user). Rather than implementing gradient fitness functions algorithmically, a popular approach in the literature is to specify fitnesses for a subset of states and then interpolate. This alternative approach is especially pertinent for use with our adaptation technique, since our state spaces are real-vector-valued and organized so that Euclidean-similar state vectors represent similar physical states.

We performed several experiments in this case study, varying the number of characters on each team. We tested both cooperative relationships between teammates, and competitive relationships between non-teammates. We also performed experiments in which we varied the initial state and the human user's behavior. We gathered statistics on the accuracy of the character's learning, the increase in its success rate with respect to the human user, and the runtime performance of our adaptation system. These results are presented in Figures 11 and 12, and Table I. Demonstrations are given in the supplementary video accompanying this article (available from http://rivit.cs.byu.edu/a3dg/publications.php).

6.1.1 *Additional Action Prediction Experiments.* We also ran some focused experiments on action prediction, to determine when k-nearest neighbor or minimax should be used to generalize cases. Recall that action prediction provides more benefit than any of our other per-layer learning methods (see Section 5.6)—therefore we thought it worthwhile to delve deeper into this specific learning method. In these experiments, we used a simplified, discrete version of our rugby environment. There was no human user, just two characters. The tackler adapted, whereas the ball-runner exhaustively tested all possible behaviors of 7 actions in length. The results of these experiments are presented in Tables II and III. While using k-nearest neighbor lets the tackler keep the ball-runner to negative forward progress on average, we found that critical mistakes were sometimes made. Alternatively, using minimax allowed the ball-runner to achieve positive forward progress on average, but very few critical mistakes were made.

Table III. Average Forward Progress Made by Ball Runner Before End of Game for all Behaviors of Length 7 (in the Same Simplified, Discrete Rugby Environment Used in Table II)

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

6.2 Capture The Flag (CTF)

This case study is based on a well-known research test bed called *Gamebots* [Kaminka et al. 2002]. This test bed modifies the popular computer game *Unreal Tournament 2003*, allowing a programmer to replace the built-in behavioral model. In Unreal Tournament, the virtual world is a complex 3D environment of rooms, hallways, stairs, and so on. It is populated with two or more players (virtual humans) organized into two teams. 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. A slide show of this case study is given in Figure 14.

We have modified the Gamebots test bed so that, rather than overriding the characters' standard behavioral model, we can simply provide the characters with auxiliary information and suggestions. It is important to note that what we have done in this case study is to add our adaptation system to an existing, professional behavioral model. Integration, while not trivial, proved to be straightforward in most aspects. This provides some additional validation for our claim that our adaptation technique can be integrated into most existing behavioral animation systems.

The Unreal Tournament behavioral model is composed of three layers, named: *Team* (goal selection), *Squad* (task selection), and *Bot* (action selection). A team is composed of one or more squads, while a "bot" is an individual character. What is unique about this behavioral model is that all members of a team or squad share the same *Team* or *Squad* layer instance, respectively. Thus there is unified group decision making. We applied our adaptation technique to all three of these layers. However, the behavioral model is complex enough that many small "component" decisions are made, and we chose to not apply adaptation to a number of these because the possible utility was deemed to be too small in comparison to the workload of integration. If the behavioral model were implemented with adaptation in mind, integration would likely be easier and more complete.

The most difficult portion of integration was the task-level learning method. Specifically, it proved challenging to run internal simulations to determine the utility of candidate tasks. This was difficult because the Unreal Tournament behavioral model is tightly coupled with the rest of the software, and could not easily be decoupled to allow "hidden" executions of the behavioral model which would not be reflected visually to the human user. Due to these complexities, we implemented a very simple version of the environment for use in running simulations. This proved sufficient in our experiments, although the characters occasionally made critical mistakes in their decision making.

The compact state spaces we use for adaptation in this case study are quite similar to those in our rugby case study. The notable differences are that only the nearest opponent and no teammates are represented in the current state, and we supply approximate information about nearby obstacles in the virtual environment. 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 character's or user's frame of reference. Assuming the character will never be in a very narrow hallway or room, this angle will be valid since all nearby obstacles will have surface normals pointing in the same general direction. Thus, for action prediction, the compact state space is defined as:

$$\mathbf{s}_t = \left(\Delta x,\ \Delta y,\ V_x^U,\ V_y^U,\ V_x^C,\ V_y^C,\ \theta\right).$$

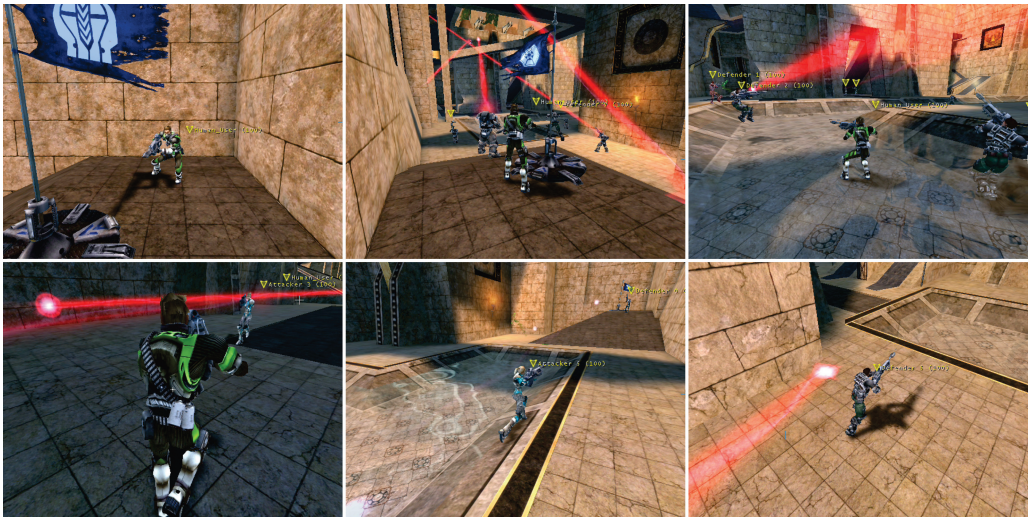

Fig. 14. Snapshots of a single Capture The Flag (CTF) animation. The human user (in green and black) is on the defending team. They are defeated by the attacking team (characters in blue and gray), who have adapted to the human's tactics and learned to work well together as a team. The last remaining defender, who has adapted to become a coward, runs away.

The dimensionality of this state space is greater than the space used for action prediction in our rugby case study. As a result, adaptation is somewhat slower than in our rugby case study. Moreover, because we use such a crude approximation of the complex virtual environment, action prediction is of a lower accuracy than in rugby. Nevertheless, our results are still promising, suggesting that our adaptation technique scales sufficiently to be useful for complex environments and characters.

The results of this case study are presented in Figures 11 and 12, and Table I. A slide show of one contiguous animation is given in Figure 14. Additional examples are given in the supplementary video accompanying this article (available from http://rivit.cs.byu.edu/a3dg/publications.php).

6.3 Automated Cinematography and Attention Selection

Usually, the actions taken in behavioral animation are movement. However, our adaptation technique is not limited to applications in movement or navigation. In this case study, we examine a very different use for adaptation: automatic selection of where a virtual camera or character's attention should be directed. Autonomous camera control and attention selection have been topics of interest in computer graphics for years (e.g. He et al. [1996]; Gillies and Dodgson [2002]).

There is a notable challenge in automatic camera and attention control: the system must be able to accurately predict the future actions of all agents in the environment. For example, in cinematography, this is necessary to achieve natural and aesthetically pleasing camera cuts between views (e.g. cut *before* two participants begin to interact). For attention selection, this is helpful in achieving intelligent-looking eye movement and in ensuring that no critical sensory information is missed. Therefore it is important that user and/or character behavior be predicted accurately.

In this case study, we performed an experiment where the camera automatically placed itself within a dynamic scene of many characters which either milled about or stopped to talk to each other. We only used the action prediction portion of our adaptation technique, and achieved good results. Note that, as shown in Table I, total CPU usage was higher in this case study than the others because there were many characters for which to predict actions.

7. SUMMARY AND DISCUSSION

We have presented a novel technique that enables autonomous cooperative/competitive virtual characters to quickly adapt online due to interaction with a human user. Our system is composed of a small set of independent learning methods that are applied individually to the layers of a behavioral/cognitive model. Our system is designed around a common behavioral animation framework, and thus can be used with most existing behavioral animation systems. Our system fully supports both reactive (behavioral) and deliberative (cognitive) decision making, in discrete or continuous state/action spaces. Our contribution in this article is important because we present a solution for a previously unsolved problem: fast adaptation for cooperative/competitive virtual characters. Adaptation is an important problem for many interactive graphical applications, such as training simulators, computer games, and so on.

As discussed throughout this article, the layered approach we have taken to interactive adaptation is logical with regards to current trends in diverse but related fields (e.g. behavioral animation, machine learning, multi-agent systems, etc). There is also interesting validation from psychology, where researchers postulate that the best approach to model human cognition is in computational layers [Newell 1990].

Our knowledge-gathering approach to adaptation can be seen as hitting a "sweet spot" between nature vs. nurture. This is because the character begins with a fundamental skill set (motor control, perception, and decision making), and then gathers knowledge in an online fashion. The character then uses this knowledge to more optimally interact with the human user.

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

However, while our multi-level adaptation technique has proved 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 notable amount of CPU. As a result, it may not be plausible to have many adapting characters in the same interactive animation. Second, integrating our adaptation technique into an existing behavioral/cognitive model is not trivial, but appears to usually be straightforward. Of course, integration will be easier with new behavioral/cognitive models that are implemented with adaptation in mind. Third, the success of our adaptation technique for a given environment/character depends on the programmer supplying an adequate compact state space representation and fitness function. Therefore, to effectively use our adaptation technique, the programmer may have to develop some new skills. Finally, since we apply discrete learning methods to different layers, there is no smooth "inbetweening" for additional intermediate layers. However, note that any layer of a behavioral/cognitive model can use one of our learning methods, based on the temporal granularity of its decision making (see Figures 3 and 4). Therefore, our adaptation technique can be applied to most existing behavioral animation systems and autonomous characters.

Although our technique has proven effective in our case studies, there is no guarantee that it will be effective for every imaginable character and environment. However, as long as our assumptions in Section 4.2 are met, we believe that our technique will work well for nearly all characters and environments. This is because the requirements of the underlying learning methods (and the system as a whole) will be met.

Another possible application of our adaptation technique is for one character to learn to adapt to another virtual character. This is interesting because it can result in very natural-looking behavioral animation, as realistic learning is reflected as the animation proceeds. Note that this use of adaptation is also applicable to offline animation, as human interaction is not required. Another possible use of

our technique is the creation of entirely new behavioral models in an online fashion by leveraging our work in this article to perform learning through demonstration. We can use the state-action model of the human's action selection to determine the decision-making of a character. One drawback to this approach is that the decision-making is somewhat shallow.

REFERENCES

ARIKAN, O., FORSYTH, D., AND O'BRIEN, J. 2003. Motion synthesis from annotations. *ACM Trans. Graph. 22*, 3, 402–408.

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*. ACM Press/ACM SIGGRAPH, 417–426.

BLUMBERG, B. AND GALYEAN, T. 1995. Multi-level direction of autonomous creatures for real-time virtual environments. In *Proceedings of SIGGRAPH 1995*. ACM Press/ACM SIGGRAPH, 47–54.

BROOKS, R. 1986. A robust layered control system for a mobile robot. *IEEE J. Robot. Autom. 2*, 14–23.

DINERSTEIN, J. AND EGBERT, P. 2004. Improved behavioral animation through regression. In *Proceedings of Computer Animation and Social Agents (CASA '04)*. Computer Graphics Society, 231–238.

DINERSTEIN, J., EGBERT, P., DE GARIS, H., AND DINERSTEIN, N. 2004. Fast and learnable behavioral and cognitive modeling for virtual character animation. *J. Comput. Anim. Virtual Worlds 15*, 2, 95–108.

EGGES, A., KSHIRSAGAR, S., AND MAGNENAT-THALMANN, N. 2004. Generic personality and emotion simulation for conversational agents. *J. Comput. Anim. Virtual Worlds 15*, 1–13.

EVANS, R. 2002. Varieties in learning. In *AI Game Programming Wisdom*, E. Rabin, Ed. Charles River Media. Hingham MA, 567–578.

FALOUTSOS, P., VAN DE PANNE, M., AND TERZOPOULOS, D. 2001. Composible controllers for physics-based character animation. In *Proceedings of SIGGRAPH 2001*. ACM Press/ACM SIGGRAPH, 39–48.

FUNGE, J., TU, X., AND TERZOPOULOS, D. 1999. Cognitive modeling: Knowledge, reasoning, and planning for intelligent characters. In *Proceedings of SIGGRAPH 1999*. ACM Press/ACM SIGGRAPH, 29–38.

GADANHO, S. 2003. Learning behavior-selection by emotions and cognition in a multi-goal robot task. *J. Mach. Learn. Res. 4*, 385–412.

GILLIES, M. AND DODGSON, N. 2002. Eye movements and attention for behavioural animation. *J. Visual. Comput. Anim. 13*, 287–300.

GMYTRASIEWICZ, P. AND DURFEE, E. 2000. Rational coordination in multi-agent environments. *Autonomous Agents and Multi-Agent Systems 3*, 4, 319–350.

GRZESZCZUK, R., TERZOPOULOS, D., AND HINTON, G. 1998. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of SIGGRAPH 1998*. ACM Press/ACM SIGGRAPH, 9–20.

GUYON, I. AND ELISSEEFF, A. 2003. An introduction to variable and feature selection. *J. Mach. Learn. Res. 3*, 1157–1182.

HE, L. W., COHEN, M., AND SALESIN, D. 1996. The virtual cinematographer: A paradigm for automatic real-time camera control and directing. In *Proceedings of SIGGRAPH 1996*. ACM Press/ACM SIGGRAPH, 217–224.

ISLA, D., BURKE, R., DOWNIE, M., AND BLUMBERG, B. 2001. A layered brain architecture for synthetic creatures. In *Proceedings of IJCAI*. 1051–1058.

KAELBLING, L., LITTMAN, M., AND MOORE, A. 1996. Reinforcement learning: A survey. *J. Arti. Intell. Res. 4*, 237–285.

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. *Comm. 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. *Int. J. Artif. Intell. Tools: Architectures, languages, algorithms 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*. 385–392.

MATTHEWS, G. 1997. *Personality, Emotion, and Cognitive Science*. Elsevier, Amsterdam.

MELTZOFF, A. AND MOORE, M. 1992. Early imitation within a functional framework. *Infant Behavior and Development 15*, 479–505.

MILLAR, J., HANNA, J., AND KEALY, S. 1999. A review of behavioural animation. *Comput. Graph. J. 23*, 127–143.

MINSKY, M. 1985. *Society of Mind*. Simon & Schuster, New York.

MITCHELL, T. 1997. *Machine Learning*. WCB/McGraw-Hill.

MONZANI, J., CAICEDO, A., AND THALMANN, D. 2001. Integrating behavioural animation techniques. *Comput. Graph. Forum 20*, 3.

NEWELL, A. 1990. *Unified Theories of Cognition*. Harvard University Press, Cambridge, MA.

PERLIN, K. AND GOLDBERG, A. 1996. Improv: A system for scripting interactive actors in virtual worlds. In *Proceedings of SIGGRAPH 1996*. ACM Press/ACM SIGGRAPH, 205–216.

PRICE, R. 2002. Accelerating Reinforcement Learning through Imitation. Ph. D. thesis, University of British Columbia.

RAO, A. AND GEORGEFF, M. 1995. BDI agents: From theory to practice. In *Proceedings of the First International Conference on Multi-Agent Systems*.

REIF, J. H. AND WANG, H. 1999. Social potential fields: A distributed behavioral control for autonomous robots. *Robotics and Autonomous Systems 27*, 171–194.

REYNOLDS, C. 1987. Flocks, herds, and schools: A distributed behavioral model. In *Proceedings of SIGGRAPH 1987* (1987). ACM Press/ACM SIGGRAPH, 25–34.

SCHYNS, P., GOLDSTONE, R., AND THILBAUT, J. 1998. The development of features in object concepts. *Behavioral and Brain Sciences 21*, 1, 1–54.

SEN, S. AND ARORA, N. 1997. Learning to take risks. In *Collected papers from AAAI-97 workshop on multiagent learning*. 59–64.

SIMS, K. 1994. Evolving virtual creatures. In *Proceedings of SIGGRAPH 1994*. ACM Press/ACM SIGGRAPH, 15–22.

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

STONE, P. AND VELOSO, M. 1997. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots 8*, 3, 345–383.

SUTTON, R. AND BARTO, A. 1998. *Reinforcement Learning: An Introduction*. MIT Press, Cambridge, Massachusetts.

TESAURO, G. 1995. Temporal difference learning in TD-Gammon. *Comm. ACM 38*, 3, 58–68.

TOMLINSON, B. AND BLUMBERG, B. 2002. Alphawolf: Social learning, emotion and development in autonomous virtual agents. In *First GSFC/JPL Workshop on Radical Agent Concepts*.

TU, X. AND TERZOPOULOS, D. 1994. Artificial fishes: Physics, locomotion, perception, behavior. In *Proceedings of SIGGRAPH 1994*. ACM Press/ACM SIGGRAPH, 43–50.

VAN LENT, M. AND LAIRD, J. 2001. Learning procedural knowledge through observation. In *Proceedings of International Conference On Knowledge Capture*. ACM Press, 179–186.

YOON, B. 2003. Real world learning: exploratory efforts. In *Proceedings of DARPA Cognitive Systems Conference*.

ZHU, T., GREINER, R., AND HAUBL, G. 2003. Learning a model of a web user's interests. In *Proceedings of Ninth International Conference on User Modeling*.

Received April 2004; revised October 2004; accepted January 2005