Cognitive and Behavioral Model Ensembles for Autonomous Virtual Characters

Jeffrey S. Whiting, Jonathan Dinerstein, Parris K. Egbert, and Dan Ventura
Computer Science Department, Brigham Young University, Provo Utah, USA

February 24, 2010

Abstract

Cognitive and behavioral models have become popular methods for creating autonomous self-animating characters. Creating these models presents the following challenges: (1) Creating a cognitive or behavioral model is a time intensive and complex process that must be done by an expert programmer, (2) The models are created to solve a specific problem in a given environment and because of their specific nature cannot be easily reused. Combining existing models together would allow an animator, without the need for a programmer, to create new characters in less time and to leverage each model's strengths, resulting in an increase in the character's performance and in the creation of new behaviors and animations. This paper provides a framework that can aggregate existing behavioral and cognitive models into an ensemble. An animator has only to rate how appropriately a character performs in a set of scenarios and the system then uses machine learning to determine how the character should act given the current situation. Empirical results from multiple case studies validate the approach.

Keywords: *Behavioral animation, cognitive modeling, ensembles, autonomous agents, AI-based animation.*

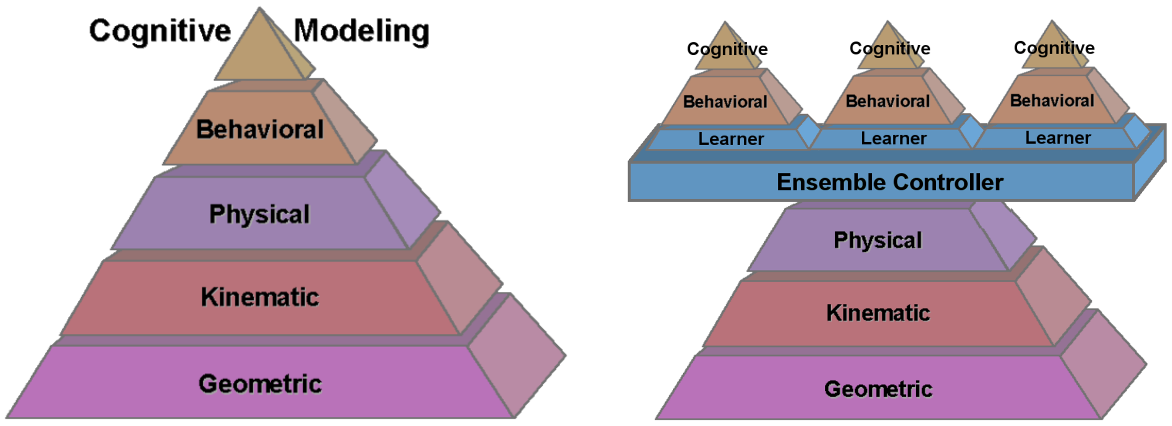
1 Introduction

Research in virtual character animation has focused recently on autonomous self-animating characters. Several different approaches to this problem have been developed and although these approaches differ greatly in their methodology, commonalities exist between them. Funge et al. [1999] have attempted to extract these commonalities into their computer graphics (CG) modeling hierarchy, as shown in Figure 1(a). Geometric and kinematic modeling involves the low level animation of geometric models, physical modeling applies real world physics to the character's motion, and behavioral and cognitive modeling attempt to create an executable model of the character's thought process. Cognitive Modeling seeks to accomplish long-term goals whereas behavioral models are reactive and seek to fulfill immediate goals.

The lower levels of the CG pyramid are generally considered to be better-understood problems that are simpler to implement. The higher levels are less well understood problems. Cognitive modeling, for example, has recently gained the attention of a few research groups but still remains a relatively undeveloped area.

Creating cognitive and behavioral models presents several challenges. First, creating a cognitive or behavioral model is a time intensive and complex process that is done by an expert programmer. The models are created to solve a specific problem in a given environment and because of their specific nature cannot be easily reused. This presents an interesting dilemma for animators and designers that are constantly creating new environments and problems but wish to incorporate autonomous characters. If existing models could be combined together the animator could create a new character in less time and without the need of a programmer. The new character would essentially be a composite of the existing models and under the direction of the animator each model's strengths would be leveraged to increase the character's performance, and to create new behaviors and animations.

This paper proposes a framework that can use existing cognitive and behavioral models and combine them to produce a character that acts according to the animator's desire. Each individual behavioral or cognitive model is combined in an ensemble similar to many machine learning ensembles [Dietterich, 2000]. Although any one model may be insufficient to produce the desired result, the ensemble of models helps



(a) Computer Graphics Modeling Hierarchy (b) Additions Made to the Hierarchy

Figure 1: (a) The computer graphics modeling hierarchy [Funge *et al.*, 1999]. (b) The addition made by this paper to allow for cognitive and behavioral model ensembles (shown in blue).

make up for the individual deficiencies. An *ensemble controller* is responsible for coordinating and choosing which model to use given the current state of the environment. When the character needs to make a decision, the ensemble controller polls each model for its suggested action. Each action is weighted according to the model's *context sensitive weight* and the best model is chosen to control the character. The context sensitive weights are learned through feedback given by the animator on how appropriately the character performed in a given situation. Through feedback given on specific examples, the framework is able to generalize to new situations allowing the character to act well without the need for additional feedback. Figure 1(b) shows the additions this paper makes to the CG modeling pyramid. As shown, the ensemble controller is able to manage multiple cognitive and behavioral models for a single character, allowing for new behaviors, new animations, and increased performance of the character.

2 Related Work

An agent is an entity that is able to perceive its environment, make choices, and then perform actions that carry out those choices [Weiss, 1999]. An agent may be a real physical agent (e.g. robots, humans, animals, etc.) or a virtual agent existing in computer-generated environments such as cinematic special effects, simulators, video games, etc. Of special interest are autonomous virtual characters. These virtual characters are independent, self-directed, and self-governing. They are able to make decisions without direction.

Many different systems have been created to control autonomous virtual agents [Tu and Terzopoulos, 1994; Blumberg and Galyean, 1995; Burke *et al.*, 2001; Isla and Blumberg, 2002]. The results of these techniques are exceptional but somewhat limited. They are primarily reactive (i.e. behavioral) systems unable to deliberate, they have no ability to learn or adapt their behavior, and they must be explicitly programmed.

Funge *et al.* [1999] have attempted to extract commonalities among the different systems into their computer graphics modeling hierarchy (see Figure 1(a)) and have added cognitive modeling. Cognitive models select goals and make deliberative decisions, in contrast to behavioral models which make only reactive decisions, and both modeling techniques are powerful tools for building autonomous characters. Though considered difficult to create, many models exist that perform specific tasks [Burke *et al.*, 2001], reach designated goals [Funge *et al.*, 1999], or behave in a stylistic manner [Blumberg *et al.*, 2002].

Machine learning algorithms have been used within almost every level of the CG hierarchy. Faloutsos *et al.* [2001] employ Support Vector Machines (SVM) to learn the preconditions of specialized physical controllers, allowing them to be used in a general purpose motor control system. Blumberg [2002] does an excellent job of incorporating machine learning within the models. Dinerstein [2004; 2005; 2005] uses various approaches (including emotional feedback during goal selection, simulation-based learning and mimicking in task selection, adaptive action prediction and mimicking during action selection) to quickly learn useful behavioral/cognitive models. Recent work extends this idea to learning and synthesizing new models from those demonstrated by a human, allowing non-technical users to easily create models that produce effective, natural-looking behavior [Dinerstein *et al.*, 2008]. On-line or real-time machine learning techniques are used to modify the character's actions immediately upon receiving feedback. This technique is frequently used when the character will be directly interacting with a human controlled agent because it allows the character to adapt its behavior quickly to the human's actions and is an area of ongoing research in both academia [Yoon *et al.*, 2000; Tomlinson and Blumberg, 2003; Burke *et al.*, 2001] and industry [Funge *et al.*, 2007]. These advances focus on using machine learning techniques *within* the cognitive and behavioral models to allow the virtual characters to learn from their experience, perform better over time, learn new behaviors, and speedup the creation and execution of the cognitive models.

However, little effort has been directed towards *combining* multiple cognitive and behavioral models together, and that is the focus of this work—to synergistically leverage the strengths of multiple models to improve the overall performance of the character. This paper presents a novel framework for combining multiple cognitive and behavioral models together to increase the character's performance and to create new behaviors and animations. This frees the animator from having to create cognitive and/or behavioral models from scratch each time one is needed.

Inspiration for this paper comes in part from research in ensemble learning [Dietterich, 2000]. Ensemble learning consists of a set of classifiers whose individual decisions are combined in some manner, usually by weighted or unweighted voting, to classify new examples. Combining the classifiers together usually results in the ensemble being more accurate than the individual classifiers.

Traditional methods for constructing good ensembles of classifiers include Bagging and Boosting, which rely on subsampling the data or feature sets to create multiple classifiers [Freund and Schapire, 1996; Breiman, 1996]. In bagging the class is predicted through unweighted voting whereas boosting uses weighted voting, based on the error rate of the classifier, to predict the class. Stacking [Wolpert, 1992] and gating networks [Jordan and Jacobs, 1993] take a different approach from bagging and boosting. They attempt to learn a function that best combines the output of the classifiers. Gating networks use the output of the learned gating function as the weights for the weighted model, thereby allowing the model to change its set of weights based on the input, making the weights context sensitive.

3 Ensemble Framework

The goal of the framework is to combine existing cognitive and behavioral models to allow for new characters and increased performance with only a small amount of time required for setup. The framework must be minimally intrusive, easy to implement, and in general not limit the types of cognitive and behavioral models that can be used. For this work, cognitive and behavioral models are considered black boxes. The only requirement is that they produce an action that can be carried out by the lower level motor controller(s). This requirement utilizes the well established and widely used method of constructing autonomous charactors by layering specific controllers, starting with low-level geometric controllers and moving up the abstract layers to cognitive models [Funge *et al.*, 1999].

For example, an animator has two cognitive models available, C_1 which herds a flock of sheep and C_2 that is able to navigate complex obstacles. It would be beneficial to use the models in combination enabling a virtual German Shepherd dog to excel at both tasks. To permit multiple cognitive and behavioral models for a single character, a framework must be created to coordinate and manage these models. The framework must include a controller to coordinate the models and a simple and easy-to-use mechanism for receiving feedback on which models perform the best in the current state of the environment.

In what follows, rather than consider the efficacy of the ensemble approach at the atomic behavior level, we instead focus on its results at the behavior trajectory level. That is, rather than ask whether an agent chose a good action at instantaneous time t, we ask whether the agent produced a good sequence of actions over time that results in accomplishing its high-level goal(s). These two questions are not unrelated, but we will concentrate on the macro-level here. The case studies in Section 4 demonstrate that agents employing

Ensemble Controller Algorithm

```
function E::getNextAction
    for $i = 1$ to $n$ do
        $a_i \leftarrow C_i$.getNextAction()
        $\lambda_i \leftarrow L_i$.getContextSensitiveRating($s$)          ▷ $s$ is state
    end for
    for $i = 1$ to $n$ do
        $w_i \leftarrow$ calcContextSenstiveWeight($i, \vec{a}, \vec{\lambda}, s, t$)          ▷ $t$ is time index
    end for
    $a \leftarrow$ selectAction($\{a_i, w_i\}_{i=1}^{n}$)          ▷ weighted voting
end function
```

Figure 2: The algorithm used by the ensemble controller to choose the next action. Because the ensemble controller mimics a cognitive model, the motor controller is unaffected by the change. Similarly, the cognitive model is unaware of and unaffected by the change. The variables $\vec{a}$ and $\vec{\lambda}$ represent n-vectors of all actions and ratings respectively (i.e., $\vec{a} = [a_1, a_2 \ldots a_n]^T$).

ensemble models produce better high-level behavior trajectories than do agents employing a single model.

3.1 Ensemble Controller

The ensemble controller is the central element of the system. It sits in between the cognitive and behavioral models and the lower level controller, usually the physical controller (refer to Figure 1). This controller is responsible for coordinating and managing the models and replicates their output—an action to be taken. Similar to an ensemble classifier that takes a set of classifiers L_1 ... L_n and combines their individual decisions (typically through unweighted or weighted voting), the ensemble controller combines the actions of individual cognitive or behavioral models C_1 ... C_n using weighted voting. Ensemble classifiers are most effective when they disagree with each other and have uncorrelated errors [Hansen and Salamon, 1990]. Likewise, the most effective ensembles consist of a combination of models that perform different actions given the same state allowing the animator to choose the best one.

When the character needs to make a decision, each model C_i in the ensemble is polled and returns its suggested action a_i. Each action, along with its weighting w_i (based on the current state s of the environment), is considered and an appropriate action or combination of actions is chosen. Therefore the ensemble controller E takes as input a set of weighted actions along with the current state and outputs an action:

$$E(\{a_1, w_1\}, \{a_2, w_2\}, ...\{a_i, w_i\}, s) \rightarrow a \tag{1}$$

It is important to note that the cognitive model is unaffected by and need not even know about the ensemble controller. The same applies to the lower level controller. Figure 2 shows the algorithm used to determine the next action.

The weighting for each cognitive or behavioral model is determined by the animator. The animator rates each model based on how well it performs in the current situation. A machine learning algorithm is then responsible for learning the rating based on the current state of the environment and later generalizing to unseen states. A tuple is created containing the cognitive or behavioral model C and its associated learner L (Figure 3). These tuples are used to create the inputs $(a_1, w_1), ...(a_i, w_i)$ for the ensemble controller.

3.2 Action Selection

Once the ensemble controller has the action and weight tuple (a_i, w_i) for each model it chooses an action based on the weighting. There are several ways this can be done. The particular action selection method that is used depends on the environment. Methods that could be used are:

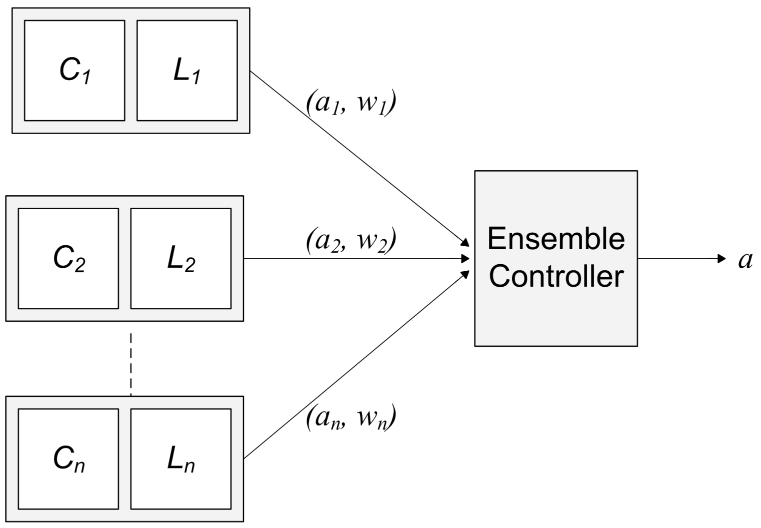


Figure 3: The framework contains tuples of a cognitive model C and the machine learning algorithm L. The ensemble controller polls C_1 through C_n and uses the context sensitive weights (w_i) to choose the action a that will be performed by the virtual character.

- **The top action is chosen.** This method chooses the action with the highest weighting.
- **The action is probabilistically chosen.** The top m actions are considered and their weights are normalized to form a probability distribution function (PDF). The action is then stochastically chosen based on that PDF.
- **The actions are blended.** The top m actions are considered and the resulting action is a linear combination based on the weights. This method is only suitable for cognitive and behavioral models that output real valued actions that can be linearly combined.

Each method has its relative advantages. Choosing the top action maximizes the performance of the character with respect to the feedback given by the animator. When interacting with a human controlled avatar it is not always desirable to choose the top action as it makes the agent deterministic—given a state, the same action is always chosen—allowing the human to guess future actions. Regardless of which of the above methods are used, the “combined” models create a behavior trajectory that is unique to the ensemble that none of the individual models can produce. The capture the flag (Section 4.1) and crowd simulation (Section 4.3) case studies choose the top action to maximize performance and demonstrate not only the character’s unique behavior trajectory but also the character’s increased efficacy to complete the required task.

Finally the creation of entirely new actions can be accomplished through blending the possible actions together. These actions are created based on a linear or quadratic combination of the suggested actions thus limiting the technique to a smaller number of environments where this is possible. Blending the actions together requires real valued action vectors such as velocities or accelerations. Categorical actions such as “capture the red flag”, “intercept the red agent”, etc., cannot be combined together in this way. In the flocking case study (section 4.2) the virtual birds blend their actions together allowing simultaneously actions such as maintaining a V-formation while avoiding a tree—see Figure 7(a).

3.3 Context Sensitive Weights

Each model has its own set of context sensitive weights learned from the feedback given by the animator. The weights are not static but vary based on the current state of the environment. The weight for model i at time t is given by

$$w_i(s,t) = \lambda_i(s) + \sum_{j=1, j \neq i}^{n} \lambda_j(s) * \delta(a_i, a_j) + \tau(t) \tag{2}$$

Where:

- $\lambda(s)$ is the context sensitive rating. This value is learned from the feedback given by the animator and is in the range [0..1]. The machine learning algorithm uses the current state of the environment to determine the appropriate weight.
- $\delta(a_i, a_j)$ is the action similarity metric. This computes the similarity between actions a_i and a_j and is provided by an expert. The same action would have a value of 1.0 and completely different actions would have a value of 0.0. The sum over the action similarities $\sum \lambda_j * \delta(a_i, a_j)$ increases the probability that models proposing similar actions will be chosen.
- $\tau(t)$ is the temporal boost. This term reduces *temporal aliasing*, i.e. rapidly switching between models. When a model is first chosen it is given a boost that decays over time. This helps prevent the character from rapidly switching between models in a short amount of time and is comparable to the momentum term used in some neural network weight updates. The boost at time t with a decay rate of r and an initial boost b is represented by $\tau(t) = r^t * b$.

3.4 Learning the Context Sensitive Rating

The central variable in the weight calculation is the context sensitive rating, and this value is learned through the feedback given by the animator. It is desirable that the learning process be as easy and as unobtrusive as possible to reduce the burden placed on the animator and to keep the time to train the system small.

The framework records the actions taken by the character, the current model, and the current state information. Offline the animator is able to review the character's actions and easily rate how appropriately the character performed using a basic slider on a scale from 0.0 to 1.0, with 0.0 being *inappropriate*, 0.5 *neutral*, and 1.0 *appropriate*. The rating given is then associated with the active model and the current state. Each model has an associated learner that attempts to learn a mapping of the animator's preference onto the state (refer to Figure 3). Because the preference is mapped onto the environment state the weights become context sensitive, i.e. weights are different based on the character's current state.

The learner can be any machine learning technique that can do regression, and the algorithm used in this work is k-nearest neighbors (k-NN) [Cover and Hart, 1967] because of its rapid training time and robustness to noise. Since the weights are learned quantities, the framework is able to generalize from the animator provided examples to unseen states. Giving feedback does not have to be a one-time event as the animator has the ability to give additional feedback at a later time to help improve the character's performance.

3.4.1 State Representation

Providing an adequate representation of the state for the learning algorithm is a challenging problem. When the system is introduced into a specific environment it is not known ahead of time how the ensemble will be used and what features are most important to adequately learn the animator's preference. This implies that as many features as possible should be included in the state representation, giving the learner the opportunity to use the feature(s) that are best suited for learning the preference. Unfortunately, extraneous and potentially unimportant features can prohibit learning. To satisfy these two constraints a very large number of features are included in the state representation, and once the animator has rated each model, a feature selection algorithm can be run on the resulting data to reduce the feature set or the animator can choose which specific features are most correlated with the feedback. The reduced feature set and data are given to the learner, providing a representation to the framework that uses only the most salient features.

3.5 Putting It All Together

For the framework to successfully increase the performance of the character and utilize multiple cognitive and behavioral models, the system must be trained. Typically the training process is done in several stages

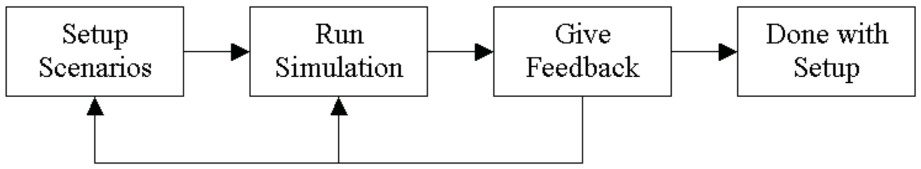


Figure 4: The process the animator uses to give feedback to the system. Scenarios representative of the animator's goals are selected and the base models are each evaluated under these scenarios. The animator provides feedback on each base model's performance and that feedback is used to train an ensemble controller. The process is then iterated using the ensemble controller to select models for each situation until the ensemble has assimilated the animator's feedback well enough to choose the "correct" base model under each scenario.

(Figure 4): the animator selects multiple scenarios that are representative of what he/she wishes to accomplish; each model is run in those scenarios and feedback is given on how well each one did; the simulation is run again, this time with the ensemble controller active and using the feedback just given; the performance of the character is once again evaluated and additional feedback is given on how well the character performed. At this point if the animator is satisfied with the character's performance no additional training is necessary. Otherwise, additional feedback can be given or additional scenarios set up to improve the character's performance until the animator is satisfied. Although this process can take several iterations, typically it is not very time consuming (usually under an hour), is significantly faster than creating a new cognitive model from scratch, and does not require an expert programmer.

4 Case Studies and Results

Three case studies were created with varied environments to stress the framework and measure its effectiveness. In each case study behavioral and cognitive models were created using "traditional" techniques with creation time ranging from several days to several weeks per model depending upon their complexity. After the individual models are created, they are combined into ensembles using the ensemble framework. Table 1 shows the approximate creation time for each ensemble. Simulations were then run to measure the performance of the virtual characters with and without the model ensembles. The performance of the characters with the individual models are used as a baseline to measure any performance gains made by characters using the model ensembles.

The studies were implemented using OpenGL on a 2.0Ghz PC with a commodity graphics card and run in realtime. The following case studies were implemented:

- Capture the Flag
- Flocking
- Crowd Simulation Using a Combat Situation

4.1 Capture the Flag

The capture the flag (CTF) case study consists of a discrete world with two opposing teams (red and blue), one blue flag, and one red flag. The goal is for an agent to retrieve the competing team's flag and return the flag to its base while preventing its flag from being taken. This case study incorporates two different environments: simple CTF and multi-agent CTF.

The simple CTF environment consists of a 16x16 map, one blue agent using naïve behaviors, and both flags (see Figure 5(a)). The four behavioral models (B_1, B_2, B_3, B_4) are unintelligent and always move the agent in the same direction regardless of the current state of the world ($B_1 \Rightarrow$ left, $B_2 \Rightarrow$ right, $B_3 \Rightarrow$ up, $B_4 \Rightarrow$ down). Because each model can only move in one direction it is not possible for any single model to successfully capture the flag.

The multi-agent CTF environment consists of a 32x32 map, two blue agents, two red agents, and obstacles (see Figure 5(b)). Rather than the naïve behavioral models the following three behavioral models are used.

A* Model. The A* model is based on the popular A* search algorithm and will plan a path through the obstacles to capture the flag [Hart *et al.*, 1968].

Potential Fields Model. The potential fields model uses imaginary forces to direct the agent [Andrews and Hogan, 1983]. Obstacles and enemies exert repulsive forces, while the goal applies an attractive force. The agent simply follows these forces to capture the flag. No path planning is done, but the agent is better able to avoid enemies than an agent using the A* model.

Arc Defense Model. This model attempts to put the agent between the enemy and the agent's own flag, thereby preventing the enemy from making a capture. This is the only model that does not actively seek to capture the enemy's flag.

The state representation for both environments consists of the following variables:

$\vec{P}$: The x and y position of the agent in the world.
$\vec{R}$: The translation vector from agent's current position to enemy's flag.
$\vec{B}$: The translation vector from agent's current position to its own flag.
H_r: If the red flag is currently being held by an agent.
H_b: If the blue flag is currently being held by an agent.
D_o: The distance to the agent's own flag.
D_e: The distance of the agent to the enemies flag.

4.1.1 Results

To measure the performance of the framework, the agent(s) and the two flags are randomly placed on their side of map. The multi-agent CTF environment also randomly generates obstacles to cover 10% of the board. The simulation is allowed to run until a team captures the flag or 500 turns have passed and then a new map is given. A minimum of 500 rounds are simulated.

In the simple CTF environment the ensemble of naïve behavioral models is able to capture the flag despite the fact that neither the ensemble controller, the behavioral models, nor any single part of the system has enough information to know what model should be chosen to successfully capture the flag. The success of the agent is entirely dependent upon a transfer of knowledge from the animator to the framework through the context sensitive weights. The ensemble controller must produce a unique behavior trajectory that allows the character to capture the flag—a trajectory that is not possible with any individual model. The results in Table 2 show that the framework is able to correctly learn the context sensitive weights and successfully capture the flag. A single iteration of the ensemble training process described in Figure 4 produced an initial capture rate of 84.1% (already a significant improvement over the baseline). In the cases in which the agent did not capture the flag, it would get stuck in a "rut" moving back and forth between several squares repeatedly until the 500 turn limit was reached. Giving additional feedback to the framework helped alleviate this problem and raised the success rate to 98.2%.

With the multi-agent CTF environment the red team and blue team attempt to capture each other's flags. To test the framework, one of the blue agents was given an ensemble of the three models while the

Case Study	**Ensemble Training Time**
Capture the Flag - Simple Environment	20 minutes
Capture the Flag - Multi-agent Environment	30 minutes
Flocking	50 minutes
Crowd Simulation - Character Ensemble	40 minutes
Crowd Simulation - Commander Ensemble	25 minutes

Table 1: The approximate amount of time the animator spent to train each model ensemble using the process described in Figure 4.

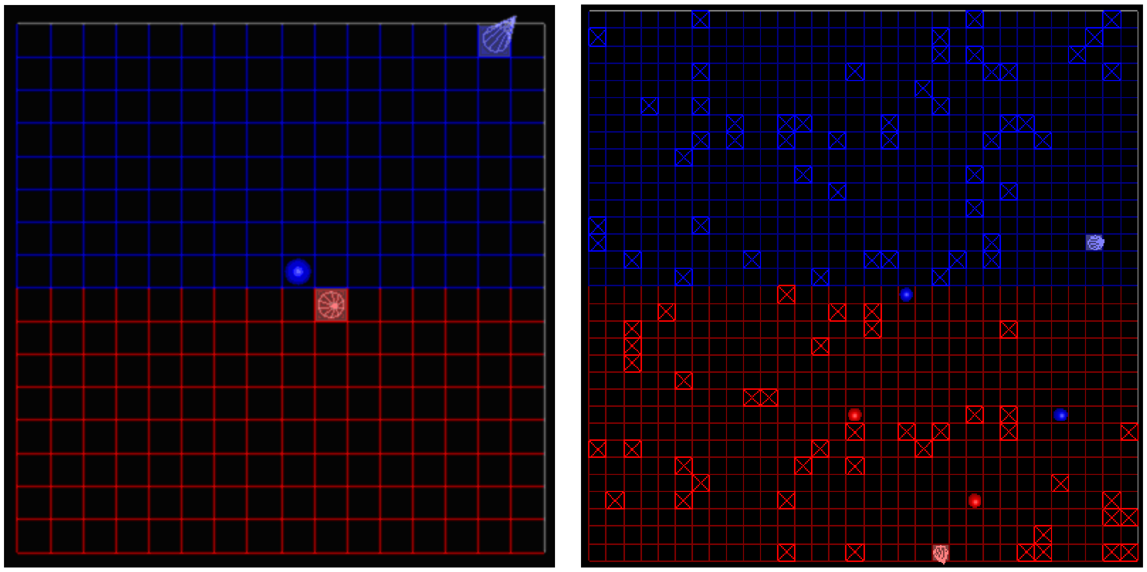

(a) Simple CTF (b) Multi-Agent CTF

Figure 5: (a) The simple capture the flag environment consists of two flags and a single agent. (b) The multi-agent CTF environment adds complexity – the world is significantly larger, includes obstacles, and has multiple agents.

other agents were restricted to one of the base models. The blue agent with the ensemble controller had a notable impact on the results (see Table 2). When the other agents employ the A* model, the result is an 8.4% spread between the teams' success rates—when the simulation did not end in a draw, the blue team captured the flag 54.2% of the time compared to 45.8% of the time for the red team. With the potential fields model the spread is not as large but is still in blue's favor at 4.0%. Finally, with the arc defense model there is a 100% spread between the teams because the red team never attempted to capture the blue team's flag, while the blue team captured the red team's flag 19.7% of the time (that is, all winning games were won by the blue team).

In both environments the framework is able to successfully combine the strengths of each individual model and improve the performance of the agent. In the simple CTF environment there is a transfer of knowledge from the animator to the system enabling the agent to perform a task that none of its naïve behavioral models could possibly do alone—successfully capture the flag. In the multi-agent environment the one blue agent with the ensemble had a positive impact on the outcome of the simulation for its team. This study demonstrates that the framework is able to learn the animator's preferences, improve the performance of the character, and create new behavior.

4.2 Flocking

The flocking simulation consists of a large continuous area with various trees, plants, mountains and virtual birds—see Figure 6. The birds attempt to flock together while avoiding the obstacles in the environment. In this case study, we augment the action of the ensemble controller to allow blending of actions for the creation of novel actions. This is possible because the models output real number acceleration vectors which can be combined in a natural way.

Rather than creating custom behavioral models for this study, existing models are used with the ensemble controller. The following models are included:

The boid model created by Reynolds [1987]. The boid model attempts to flock towards other birds, match their velocity, avoid crowding, and dodge obstacles. The boids flock based completely on their local perception of the world. Platt's [2004] boid implementation is used as a base for this work.

Capture Rate ⇒	**Baseline**	**Ensemble**
Simple CTF	0.0%	98.2%
Multi-Agent CTF A* model vs Ensemble	49.8%	54.2%
Multi-Agent CTF Potential Fields vs Ensemble	50.8%	52.0%
Multi-Agent CTF Arc Defense vs Ensemble	0%	100%

Table 2: Summary of CTF success rate for the blue team with and without the agent using behavioral model ensembles. Results in the *Baseline* column reflect the percentage of wins for the blue team without an ensemble agent. For simple CTF, the blue team never wins because none of the base models are powerful enough to accomplish the task of stealing the flag. In the more complex game, the baseline results are random for two of the scenarios, as the composition of both teams is the same. For the the baseline arc defense model, neither team ever attempts to steal the opponents flag. The *Ensemble* column demonstrates an advantage for the blue team when it includes an ensemble player (while the red team has only base model players).

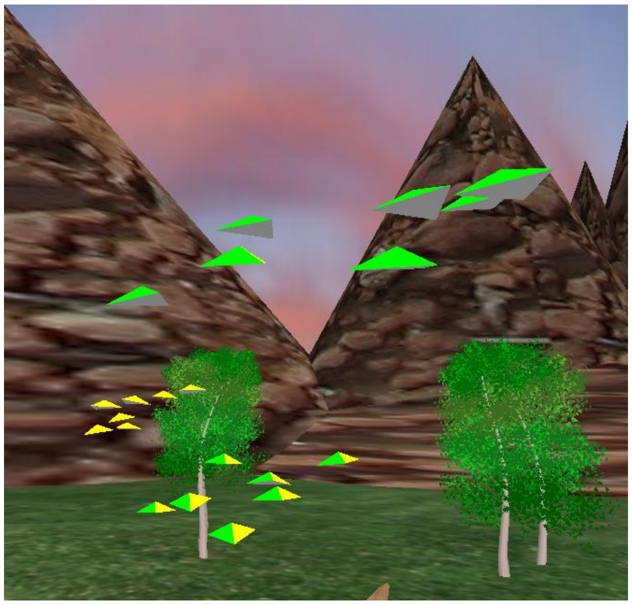
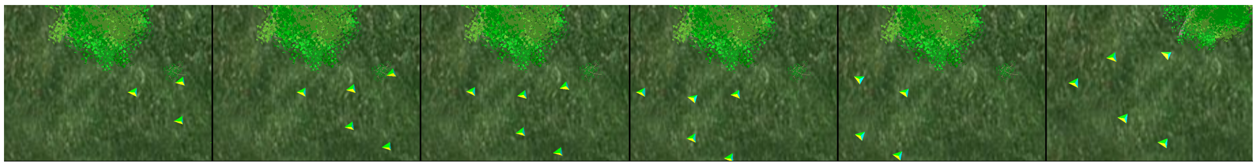
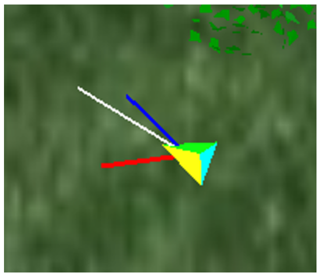


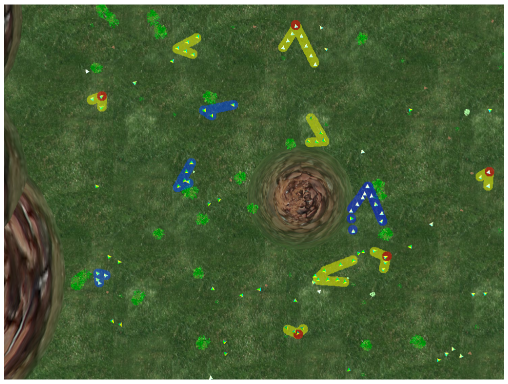
Figure 6: Three groups of flocking birds can be seen here. They attempt to stay together while avoiding obstacles such as trees and mountains.



(a) Obstacle Avoidance While in Formation



(b) Action Blending



(c) Highlighted Field

Figure 7: Subfigure (a) The birds are able to keep in formation while avoiding the obstacle. Subfigure (b) shows a bird and the suggested actions of the models as well as the chosen action. The blue, red, and green (not visible) lines represent the different suggested actions by the behavioral models and the white line represents the actual action taken. As can be seen, the final action is a weighted combination of the different suggested actions. Subfigure (c) shows the field with met visual goals highlighted. The red highlighted birds are performing a stunt, the yellow ones are flocking in a V-formation while following their leader, and the blue ones are avoiding the obstacles while staying as much as possible in V-formation.

Model	**Collisions per Second**
Boid	3.00
V-formation	16.25
Stunt	8.43
Ensemble	3.23

Table 3: Flocking Results. The ensemble agents successfully combine stunt and formation flying with obstacle avoidance, reducing collisions to nearly the baseline (*Boid*) rate.

A V-formation model [Gervasi and Prencipe, 2004]. Gervasi and Prencipe's work involves the distributed coordination and control of anonymous, memoryless virtual birds in a formation with a designated leader in a 2D environment. Their algorithm is used as the basis for a V-formation model without any predesignated leader in a 3D environment.

Scripted stunt model. A scripted stunt model allows an agent to perform a number of different stunts. The possible stunts are a figure 8, an S shaped path, a loop left and a loop right.

The state representation for this study consists of the following variables:

$\vec{P}$: The x, y, and z position of the agent in the world.
D_o: The distance of the agent to the nearest obstacle.
A_o: The direction of the obstacle relative to the agent's current direction.
H: If agent is currently at the head of a flock of birds in the V-formation.

4.2.1 Results

This case study is evaluated primarily on a statistical level but is also evaluated on a visual level. Statistically the number of collisions the birds have with the obstacles is compared. Visually we expect the birds to attempt to flock in a V-formation while avoiding the obstacles. Additionally, when a boid is leading the formation it should begin performing stunts, inducing the other birds to follow. For each simulation, 100 birds start off at random locations on the map with random horizontal and vertical directions.

To gather the results, 3 different simulations were done for 3 minutes each, and their outputs were averaged. Table 3 shows the number of collisions per second with 100 virtual birds in the field. Of the base models, only the boid model features the collision avoidance algorithm and can be considered the baseline for the results. The V-formation and stunt models hit the obstacles at a much higher rate. When the birds are using an ensemble of the models they achieve a rate of 3.23 collisions per second, which is very close to our baseline, even though they are using all three models simultaneously. The ensemble is able to successfully learn the animator's preference to avoid the obstacles, thus meeting our goals statistically.

In addition to avoiding collisions, the ensemble method needs to meet our visual goals to be considered a success. The goals are to flock in a V-formation while avoiding the obstacles and to performs stunts when leading the formation. Admittedly the the visual goals are subjective in nature, but these goals need to be met or little has been gained by using the ensemble. In Figure 7(a) we can see how the model ensemble reacts as the formation approaches the tree. As the birds get closer to the obstacle they begin to avoid it while at the same time attempting to stay in formation. The actions are linearly blended to produce a completely new action (see Figure 7(b)). The new action allows the birds to successfully avoid the tree when close while staying in formation as best as possible throughout the avoidance maneuver. Figure 7(c) is a snapshot of the simulation after it had been running for approximately 1 minute showing the birds and the different visual goals they are meeting. To get to this point in the simulation, the birds had to find nearby birds, form a V-formation, and avoid obstacles.

The continuous environment in this case study presents an important challenge that the framework is able to overcome—the ensemble characters successfully meet the statistical goals and the subjective visual goals. The linear blending of actions not only produces completely new actions not suggested by any single model but also increases the performance of the birds—they are able to avoid obstacles while staying in formation.

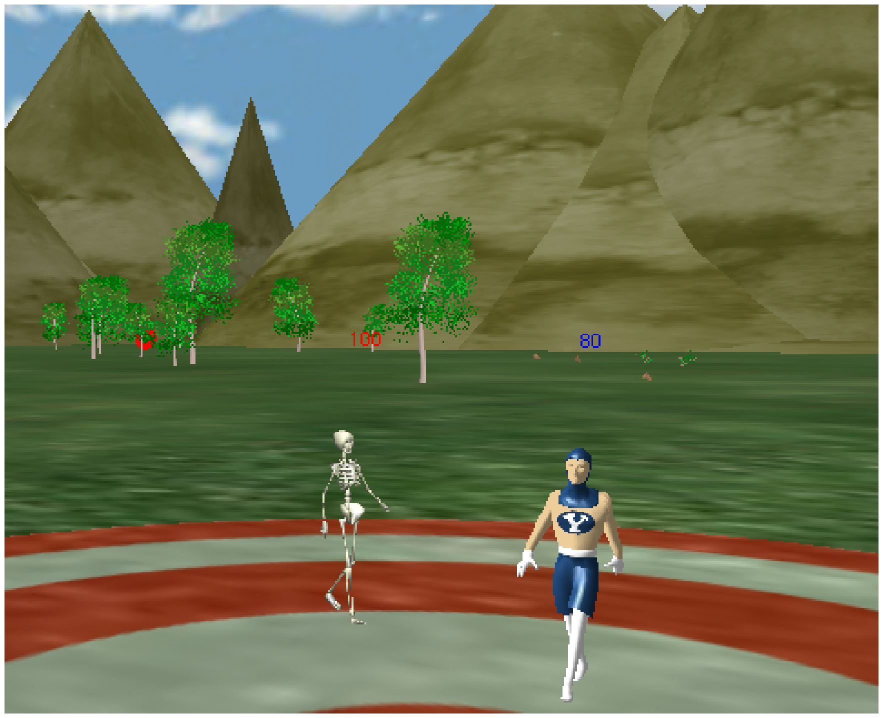


Figure 8: Shown here is the combat situation environment. The super character is attempting to secure the location specified by the target.

4.3 Crowd Simulation Using a Combat Situation

The environment is a large continuous area where one army attempts to secure two target locations on the map defended by a second army (see Figure 8). The attacking team has three different types of characters which will all be combined in an ensemble to create a "super character". Although the characters act independently, a commander directs where they go and when they attack. Different commanding strategies are created—a rush strategy and an end-around strategy. An ensemble of cognitive models using these strategies is also created to improve the commander's performance.

There are four different cognitive models (characters) created for this case study, each with 100 health points and different abilities:

- A hand-to-hand combat character that actively seeks out nearby enemies and has the strongest attack.
- A character that can attack the enemy from a distance.
- A character that can heal other friendly characters around it but is unable to attack or heal itself.
- A "super character" is created that uses an ensemble of the above cognitive models to perform the actions of all three.

In addition to the troop cognitive models there are two commander cognitive models that attempt different strategies. The first model, rush commander, attempts to secure the points as quickly as possible by charging the enemy. It does this by following the shortest path to the target locations and cutting through the defending team. The second model, the end-around commander, takes a more defensive approach and attempts to use an end-around strategy by avoiding the enemy all together. When the enemy gets close, a defensive formation is taken and the enemy is allowed to attack.

The commander controlling the defending characters patrols the area around the two target locations and attacks when the other team gets close.

	Rush Commander			**End-Around Commander**		
Statistic	No Ensembles	Ensembles	t-Test	No Ensembles	Ensembles	t-Test
Sim. Time	170.6s	181.7s	0.123	222.9s	232.6s	0.42
Win %	58.8%	78.4%	**0.033**	80.4%	98.0%	**0.0038**

Table 4: Simulation Results for Ensemble Troops. Incorporating ensemble troops into the game impacts both simulation time and success rate. While training time is slightly longer with ensemble troops, this is not statistically significant; in contrast, including ensemble troops drastically increases winning percentage and is statistically significant at $\alpha = 0.95$.

Ensemble Commander but no Ensemble Troops					
	Ensemble Commander	Rush Commander	t-Test	End-Around Commander	t-Test
Simulation Time	193.7s	170.6s	**0.0021**	222.9s	**0.0019**
Winning Percentage	88.2%	58.8%	**0.00062**	80.4%	0.28

Ensemble Commander and Ensemble Troops					
	Ensemble Commander	Rush Commander	t-Test	End-Around Commander	t-Test
Simulation Time	191.2s	181.7s	0.19	232.6s	**0.00018**
Winning Percentage	94.1%	78.4%	**0.021**	98.0%	0.31

Table 5: Ensemble Commander Simulation Results. Incorporating an ensemble commander has a positive effect on both simulation time and winning percentage. The teams led by an ensemble commander successfully combine the winning percentage of troops led by the *End-Around Commander* with the speed of troops led by the *Rush Commander*, independent of whether or not those troops include ensemble agents themselves.

The state representation for this study consists of the following variables:

$\vec{P}$: The x, and y position of the agent in the world.
T: The team the agent is on.
D_f: The distance of the agent to the closest friendly agent.
H_f: The health of the closest friendly agent.
D_e: The distance of the agent to the closest enemy.
H_e: The health of the closest enemy.

4.3.1 Results

The attacking team is made up of 12 units—4 hand-to-hand characters, 4 long range characters, and 4 healing characters. The defending team is made up of 8 hand-to-hand characters. This setup was chosen because it allows each character to exhibit their unique skills, each team has the same number of offensive characters giving them similar attacking abilities, and although the attacking team has a slight edge the results are not lopsided (see winning percentage without ensembles in Table 4).

For the ensemble simulations, two hand-to-hand combat characters and 2 long range attack characters were replaced by super characters. Each simulation involved the random placement of the characters on their side of the field and was run until the two target locations were secured or until every member of the attacking team was killed. For each simulation the *simulation time*—the amount of time it took before an ending condition was reached—and the *winning percentage*—the overall winning percentage of the attacking team—was recorded. A two sided t-test was performed on each variable and is only considered significant above the 95th percentile. Fifty battles were simulated for each commander with and without the super characters.

Table 4 shows the results with the rush and end-around commanders, and reveals that the super characters have a large impact on battles. The winning percentage rose from 58.8% to 78.4% for the rush commander and 80.4% to 98.0% for the end-around commander. Note that the end-around commander has a much higher winning percentage but is significantly slower than the rush commander in accomplishing the capturing task.

An ensemble of the commander cognitive models was also made. The enhanced commander should secure the locations more quickly while at the same time avoiding the enemy and taking a defensive posture when the enemy gets close. Table 5 shows that with and without the super characters the ensemble of the commander cognitive models captures more quickly than the end-around commander and has a better winning percentage overall. Although the winning percentage is less with the ensemble commander and ensemble troops than with the end around commander ($94.1\% < 98.0\%$) the t-test shows the result is not significant.

Overall, incorporating the framework into this case study works well. Performance improves in nearly every aspect and layering model ensembles (commander and troops), results in even better performance. In addition, the super character performed the unique actions of all three behavioral models, allowing for new behaviors and animations.

5 Conclusion

Limitations of current techniques for creating cognitive and behavioral models include the need for a programmer to create the models through a time intensive and complex process and an inability to reuse or combine existing models together. This paper has presented a novel framework that is able to overcome these limitations.

The framework consists of an ensemble of cognitive and behavioral models coordinated by an ensemble controller (Section 3.1, Figure 3) that is able to synergistically combine the individual models (leveraging the strengths of each one) and improve the character's performance. The manner in which the models are combined is determined by the animator's feedback given to the system and varies depending upon the current environment state (Sections 3.2, 3.4). In each of the case studies the framework is empirically shown to increase the character's performance. Furthermore, the characters exhibit new behaviors and animations not present in the individual models.

The framework successfully leverages multiple cognitive models, and the average time taken to train the system can be measured in minutes as opposed to the days or weeks it takes a programmer to create new cognitive and behavioral models. The animator is able to control how the models work together through feedback given to the system.

This paper shows that it is possible to successfully create cognitive and behavioral model ensembles that improve performance of the character. This type of framework allows for reuse of models already created, the potential creation of specialized cognitive models that excel at a specific task that can be shared with others, and rapid creation of new characters through combination of existing models.

The framework is most useful when it is able to combine the strengths of each base model, allowing the character to overcome the deficiencies of any single model. This implies that the cognitive or behavioral models demonstrate diversity in their suggested actions so that the system can use feedback from the animator to select the most appropriate action. When selecting the next action to perform, the ensemble controller takes as input the suggested actions from each cognitive model, requiring every model to execute each time an action is taken. This may require significant processing resources; fortunately the models can be executed in parallel, bounding the action selection time by the execution time of the slowest model.

Future work includes the exploration of different learning algorithms and their impact on performance; improved model weighting schemes, possibly leveraging techniques like boosting; and improved animator control over the character. Additionally, the animator could be replaced with a fitness function such that the system receives feedback from the function and automatically learns the best combinations of models to maximize the fitness function. In its current instantiation, the ensemble controller resembles a perceptron [Rosenblatt, 1958], in essence computing a linear combination of the base cognitive models. An obvious avenue to explore would be to implement the ensemble controller as a higher order machine learning technique. Additional case studies such as a conversional agent that converses with a human, training an animal using real-time learning techniques, or creating an ensemble agent to compete in a video game could be performed to help identify the viability of the framework in such environments. Finally, this framework will not work for all models given the enormous variability in how cognitive models are integrated into the CG heirarchy. Additional research could be done in those situations to determine how to implement an ensemble framework.

References

[Andrews and Hogan, 1983] J. R. Andrews and N. Hogan. Impedance control as a framework for implementing obstacle avoidance in a manipulator. *Control of Manufacturing Processes and Robotic Systems*, pages 243–251, 1983.

[Blumberg and Galyean, 1995] B.M. Blumberg and T.A. Galyean. Multi-level direction of autonomous creatures for real-time virtual environments. *Proceedings of SIGGRAPH*, 95:47–54, 1995.

[Blumberg *et al.*, 2002] B. Blumberg, M. Downie, Y. Ivanov, M. Berlin, M.P. Johnson, and B. Tomlinson. Integrated learning for interactive synthetic characters. *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*, pages 417–426, 2002.

[Breiman, 1996] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[Burke *et al.*, 2001] R. Burke, D. Isla, M. Downie, Y. Ivanov, and B. Blumberg. Creaturesmarts: The art and architecture of a virtual brain. *Proceedings of the Game Developers Conference*, pages 147–166, 2001.

[Cover and Hart, 1967] T. Cover and P. Hart. Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1):21–27, 1967.

[Dietterich, 2000] T.G. Dietterich. Ensemble methods in machine learning. *First International Workshop on Multiple Classifier Systems*, LNCS 1857(4):1–15, 2000.

[Dinerstein and Egbert, 2005] J. Dinerstein and P.K. Egbert. Fast multi-level adaptation for interactive autonomous characters. *ACM Transactions on Graphics (TOG)*, 24(2):262–288, 2005.

[Dinerstein *et al.*, 2004] J. Dinerstein, P.K. Egbert, H. Garis, and N. Dinerstein. Fast and learnable behavioral and cognitive modeling for virtual character animation. *Computer Animation and Virtual Worlds*, 15(2):95–108, 2004.

[Dinerstein *et al.*, 2005] J. Dinerstein, D. Ventura, and P.K. Egbert. Fast and robust incremental action prediction for interactive agents. *Computational Intelligence*, 21(1):90–110, 2005.

[Dinerstein *et al.*, 2008] J. Dinerstein, P.K. Egbert, D. Ventura, and M.A. Goodrich. Demonstration-based behavior programming for embodied virtual agents. *Computational Intelligence*, 24(4):235–256, 2008.

[Faloutsos *et al.*, 2001] Petros Faloutsos, Michiel van de Panne, and Demetri Terzopoulos. Composable controllers for physics-based character animation. In Eugene Fiume, editor, *Computer Graphics Proceedings (SIGGRAPH)*, pages 251–260. ACM Press / ACM SIGGRAPH, 2001.

[Freund and Schapire, 1996] Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. *Proceedings of the Thirteenth International Conference on Machine Learning*, 148:156, 1996.

[Funge *et al.*, 1999] J. Funge, X. Tu, and D. Terzopoulos. Cognitive modeling: Knowledge, reasoning and planning for intelligent characters. *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*, pages 29–38, 1999.

[Funge *et al.*, 2007] J. Funge, R. Musick, D. Dobson, N. Duffy, M. McNally, X. Tu, I. Wright, W. Yen, and B. Cabral. Real time context learning by software agents. United States Patent No. 7,296,007, November 2007.

[Gervasi and Prencipe, 2004] V. Gervasi and G. Prencipe. Coordination without communication: The case of the flocking problem. *Discrete Applied Mathematics*, 144(3):324–344, 2004.

[Hansen and Salamon, 1990] L.K. Hansen and P. Salamon. Neural network ensembles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10):993–1001, 1990.

[Hart *et al.*, 1968] P.E. Hart, N.J. Nilsson, and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics*, 4(2):100–107, 1968.

[Isla and Blumberg, 2002] D. Isla and B. Blumberg. New challenges for character-based AI for games. *AAAI Spring Symposium on AI and Interactive Entertainment, Palo Alto, CA, March*, 2002.

[Jordan and Jacobs, 1993] M.I. Jordan and R.A. Jacobs. Hierarchical mixtures of experts and the em algorithm. *Proceedings of the International Joint Conference on Neural Networks*, 2, 1993.

[Platt, 2004] Robert Platt. 3D boids simulation. http://www.navgen.com/3d_boids/, 2004.

[Reynolds, 1987] C.W. Reynolds. Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics*, 21(4):25–34, 1987.

[Rosenblatt, 1958] F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386–408, 1958.

[Tomlinson and Blumberg, 2003] B. Tomlinson and B. Blumberg. Alphawolf: Social learning, emotion and development in autonomous virtual agents. *Innovative Concepts for Agent-based Systems*, LNCS 2564:35–45, 2003.

[Tu and Terzopoulos, 1994] X. Tu and D. Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, pages 43–50, 1994.

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

[Wolpert, 1992] D.H. Wolpert. Stacked generalization. *Neural Networks*, 5(2):241–259, 1992.

[Yoon *et al.*, 2000] S.Y. Yoon, R.C. Burke, B.M. Blumberg, and G.E. Schneider. Interactive training for synthetic characters. In *Proceedings of the National Conference on Artificial Intelligence*, pages 249–255, 2000.