2007-06-08

Cognitive and Behavioral Model Ensembles for Autonomous Virtual Characters

Jeffrey S. Whiting
Brigham Young University - Provo

COGNITIVE AND BEHAVIORAL MODEL ENSEMBLES FOR AUTONOMOUS VIRTUAL CHARACTERS

by

Jeffrey S. Whiting

A thesis submitted to the faculty of

Brigham Young University

in partial fulfillment of the requirements for the degree of

Master of Science

Computer Science

Brigham Young University

August 2007

BRIGHAM YOUNG UNIVERSITY

GRADUATE COMMITTEE APPROVAL

of a thesis submitted by

Jeffrey S. Whiting

This thesis has been read by each member of the following graduate committee and by majority vote has been found to be satisfactory.

Date	Parris K. Egbert, Chair
Date	Dan Ventura
Date	Quinn Snell

BRIGHAM YOUNG UNIVERSITY

As chair of the candidate's graduate committee, I have read the thesis of Jeffrey S. Whiting in its final form and have found that (1) its format, citations, and bibliographical style are consistent and acceptable and fulfill university and department style requirements; (2) its illustrative materials including figures, tables, and charts are in place; and (3) the final manuscript is satisfactory to the graduate committee and is ready for submission to the university library.

Date

Parris K. Egbert
Chair, Graduate Committee

Accepted for the Department

Parris K. Egbert
Graduate Coordinator

Accepted for the College

Thomas W. Sederberg, Associate Dean
Physical and Mathematical Sciences

ABSTRACT

COGNITIVE AND BEHAVIORAL MODEL ENSEMBLES FOR AUTONOMOUS VIRTUAL CHARACTERS

Jeffrey S. Whiting

Computer Science

Master of Science

Cognitive and behavioral models have become popular methods to create 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 of a programmer, to create new characters in less time and would be able to leverage each model's strengths to increase the character's performance, and to create new behaviors and animations. This thesis provides a framework that can aggregate together existing behavioral and cognitive models into an ensemble. An animator only has to rate how appropriately a character performed and through machine learning the system is able to determine how the character should act given the current situation. Empirical results from multiple case studies validate the approach taken.

ACKNOWLEDGMENTS

I would like to thank my wife Laurie for her never ending expressions of love, patience, and encouragement. I would also like to acknowledge Dr. Egbert and my committee members for their time and help on this thesis.

Table of Contents

List of Tables

List of Figures

Chapter 1

Overview

1.1 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 modeling hierarchy, as shown in Figure 1.1. 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.

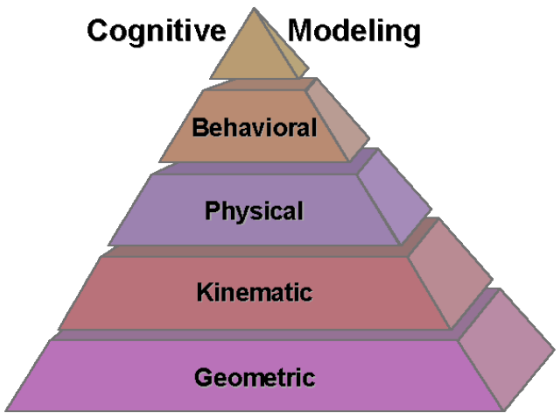


Figure 1.1: The computer graphics modeling pyramid.

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.

1.2 Overview of Thesis

This thesis 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 1997]. Although any one model may be insufficient to produce the desired result, the ensemble of models helps 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 2 shows the additions this thesis makes to the CG modeling pyramid. As can be seen, 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.

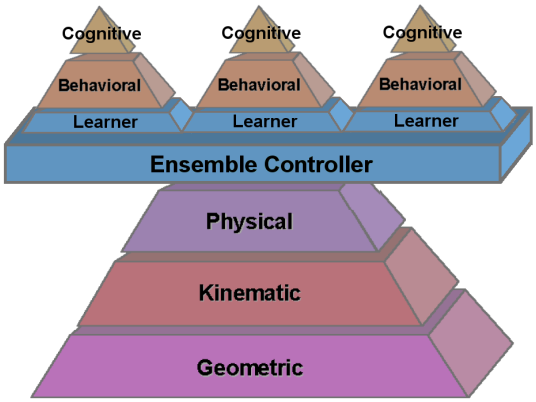


Figure 1.2: Additions made to the computer graphics modeling pyramid by this thesis (shown in blue).

1.3 Thesis Statement

Adding ensemble controllers to the current CG modeling hierarchy allows for the use of multiple cognitive and behavioral models in combination to create new behaviors and animations, and to increase overall performance.

Chapter 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 [Stone and Veloso 2000, Weiss 1999]. An agent may be a real physical agent (e.g. robots, humans, animals, etc.) or 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, Perlin and Goldberg 1996, Blumberg and Galyean 1995, Monzani et al. 2001, 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 which is difficult.

Funge et al. [1999] have attempted to extract commonalities between the different systems into their computer graphics modeling hierarchy (see Figure 1.1) and have added cognitive modeling. Cognitive models select goals and make deliberative decisions whereas behavioral models only make reactive decisions. Cognitive and behavioral modeling techniques are powerful tools to build 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] employed Support Vector Machines (SVM) to

learn the preconditions of specialized physical controllers allowing them to be used in a general purpose motor control system. Their work represents one of the first systems to combine multiple models together to improve the character's performance. However, their technique is not suitable for behavioral and cognitive models and is thus limited to physics-based motor controllers due to the relative simplicity of the motor controllers, reliance of the machine learning technique on domain knowledge, and the intrusive nature of the framework that places strict requirements on the motor controllers. Dinerstein [2004, 2005a, 2005b] used emotional feedback during goal selection, simulation-based learning and mimicking in task selection, and adaptive action prediction and mimicking during action selection. Recent work has focused 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 execution of the cognitive models. Blumberg [2002] has done an excellent job of incorporating machine learning within the models. Dinerstein [2004] used a machine learning algorithm to approximate the cognitive model which increased the scalability and decreased the creation time.

Little effort has been directed into how to combine multiple cognitive and behavioral models together. However, it would be advantageous to leverage the strengths of each model to improve the overall performance of the character according to the animator's desire. This thesis 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 thesis comes in part from current research into ensemble learning [Dietterich 1997]. 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 results in the ensemble being more accurate than the individual classifiers.

Methods for constructing good ensembles of classifiers is one of the most active areas of research. Bagging and boosting rely on subsampling the data or feature sets

to create multiple classifiers [Freund and Schapire 1996, Breiman 1996]. In bagging, multiple classifiers are created based on subsets of the data and predicts the class through unweighted voting. In boosting, the later classifiers focus on data that is misclassified and weighted voting, based on the error rate of the classifier, is used to predict the class. Stacking takes a different approach from bagging and boosting. It uses a set of classifiers and learns a function that combines the predictions of the individual classifiers. Wolpert [1992] proposes a stacking scheme using a leave-one-out cross validation.

Gating networks, in addition to learning the weights for a model that uses learned weights, learn a gating function that produces as output the weights for such a model [Jordan and Jacobs 1993]. Thereby the gating function allows the model to change its set of weights based on the input making weights become context sensitive. In theory the gating function can be arbitrarily complex and can be applied to any weighted model (function). The weighted model can be a traditional machine learning technique, an ensemble of classifiers, a stacking scheme, or any other weighted method.

Chapter 3

Proposed Solution

3.1 Methods

The goal of this thesis is to create a framework that can 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 layering specific controllers, starting with low level geometric controllers and moving up the abstract layers to cognitive models, to construct autonomous characters [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.

The following sections describe the details of this framework. Section 3.2 explains what the ensemble controller is and its essential role in the framework. The methods used by the ensemble controller to choose an action are described in Section 3.3. Section 3.4 discusses the context sensitive weighting used for each cognitive

or behavioral model. We investigate how the context sensitive ratings are learned from the animator and how the current state of the environment is represented in Section 3.5. Section 3.6 concludes this chapter explaining how the different parts of the framework work together.

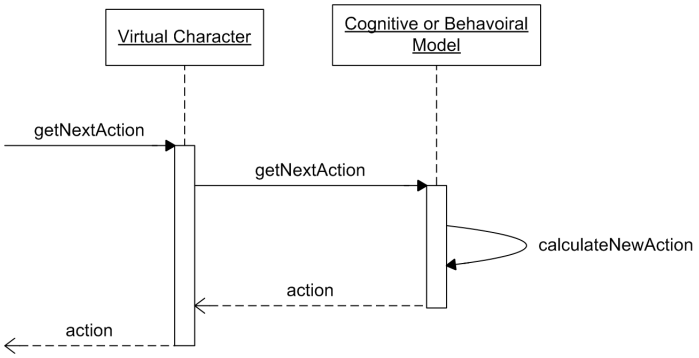
3.2 Ensemble Controller

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 1.2). 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) [Dietterich 1997], the ensemble controller combines the individual cognitive or behavioral models C_1 ... C_n actions 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 come from a combination of models that perform different actions given the same state allowing the animator to choose the best one.

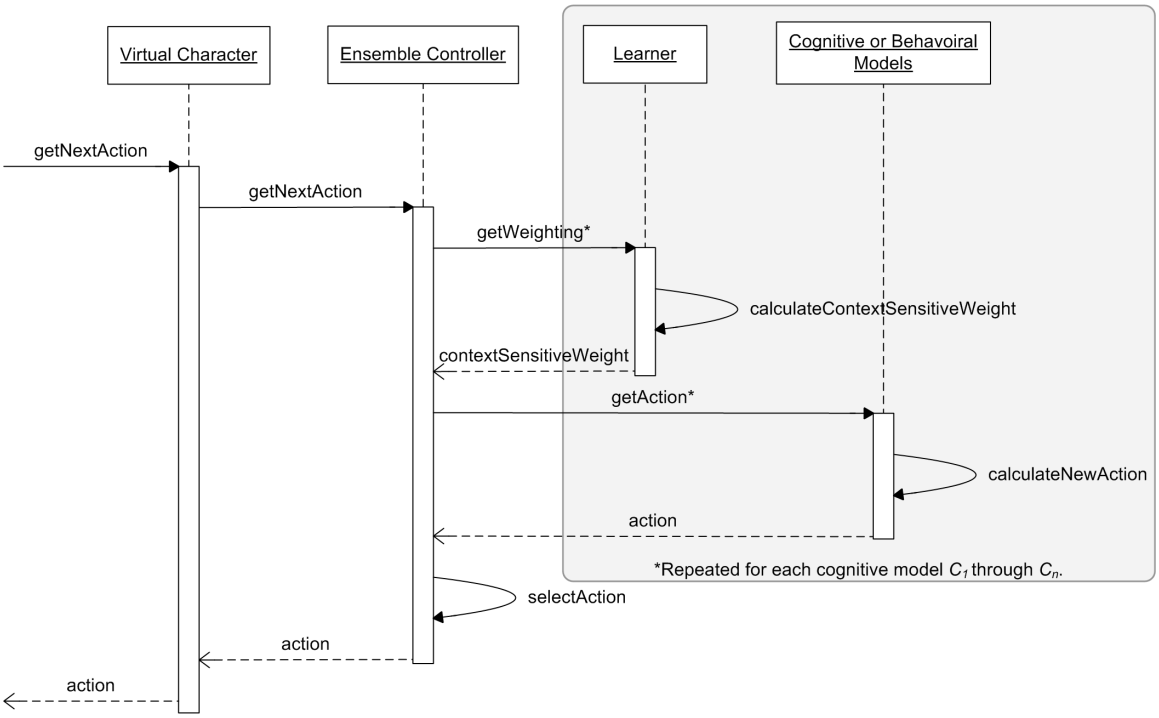
When the character needs to make a decision, each model C_i in the ensemble is polled and returns its suggested action a_i. Each action, along with its weighting w_i (based on the current state of the environment S), is considered and an appropriate action or combination of actions is chosen. Therefore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

$$E(\{a_1, w_1\}, \{a_2, w_2\}, ...\{a_i, w_i\}, S) \rightarrow a \tag{3.1}$$

Figure 3.1 (b) shows the call sequence of the ensemble controller to return the next action and Figure 3.1 (a) shows the normal call sequence. 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.



(a) Normal Call Sequence



(b) Call Sequence with Ensemble Controller

Figure 3.1: Subfigure (a) shows the normal call sequence to retrieve the next action, subfigure (b) shows the modification made to this call sequence by the framework. Notice that the cognitive model is unaffected by this change.

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 to learn the rating based on the current state of the environment and later generalize to unseen states. A tuple is created containing the cognitive or behavioral model C and its associated learner L (Figure 3.2). These tuples are used to create the inputs $(a_1, w_1), ...(a_i, w_i)$ for the ensemble controller.

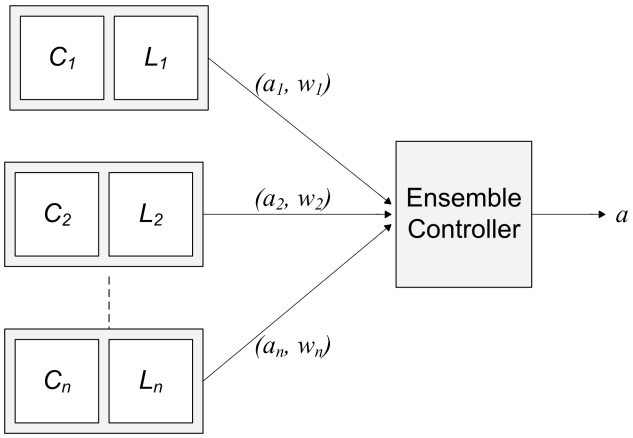


Figure 3.2: The framework contains tuples of cognitive models C and the machine learning algorithm L. The ensemble controller polls C_1 through C_n and uses the context sensitive weights (w_i) to choose the model C_i that will control the virtual character.

An alternative technique to the one presented above involves replacing each individual learning algorithm for models and the ensemble controller with one learning algorithm. The learning algorithm would then be entirely in charge of selecting the appropriate action based on the current state and the animator's feedback. The current technique is chosen because it simplifies the feedback mechanism, which reduces the burden on the animator, and allows for various model selection methods. The system doesn't present the animator with the actions of each model for every state but instead begins with an initial state from which all models are evaluated. Even though subsequent states for each model are different, the individual learning

algorithms associated with each model are able to learn animator's preference. Thus, the animator is able to rate each model individually and doesn't have to evaluate all models for all given states as would be required in the alternative technique.

3.3 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:

- **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. Because of this it is the method used in 2 of the 3 case studies (sections 4.2, 4.4) to highlight the performance gains possible by implementing the framework. 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. Finally, blending the actions together can only be used in a small number of environments where it is possible. Completely new actions are created based on a linear or quadratic combination of the suggested actions. 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 4.3) the actions are linearly combined together allowing the virtual birds to avoid obstacles while staying in formation.

3.4 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

$$w_i(s,t) = \lambda_i(s) + \sum_{j=1, j\neq i}^{n} \lambda_j(s) * \delta(a_i, a_j) + \tau(t) \tag{3.2}$$

Where:

- $\lambda(s)$ is the context sensitive rating. This value is learned from the feedback given by the animator and is in the range of [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 momentum used in neural networks. The boost at time t with a decay rate of r and an initial boost b is represented by $\tau(t) = r^t * b$

3.5 Learning the Context Sensitive Rating

The central variable in the weight calculation is the context sensitive rating. 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 dislike, 0.5 neutral, and 1.0 like (see Figure 3.3). 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 Figures 1.2 and 3.2). 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.

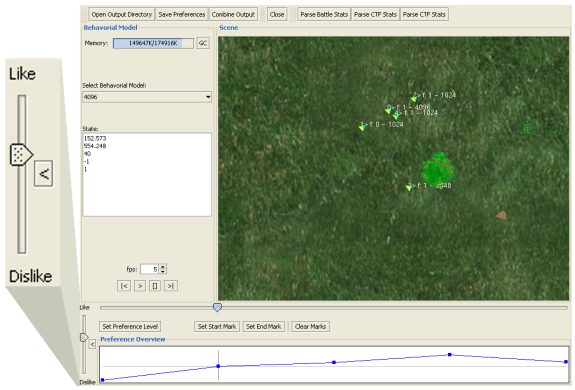


Figure 3.3: The simple slider used by the animator to give feedback to the framework. A learning algorithm uses the current state and the animator preference to learn the context sensitive ratings.

The learner can be any machine learning technique that can do regression. Algorithms we will explore in this work include k-nearest neighbors (K-NN) [Cover and Hart 1967] and regression trees [Breiman and Breiman 1984]. K-NN has

the advantage of extremely quick training time and robustness. In addition to their reasonable training times and fast execution times, regression trees have the ability to ignore extraneous features. Since the weights are learned quantities, the framework is able to generalize from the animator provided examples to unseen states. Giving feedback doesn't 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.

3.5.1 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. This gives 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. 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. This should help provide sufficient information to the framework using only the most salient features.

The chosen algorithm for this framework is the k-nearest neighbor (K-NN) algorithm [Cover and Hart 1967]. K-NN is robust to noise and has a quick training time. Feature selection is important for the K-NN learner as extra dimensions will significantly reduce its effectiveness. In addition, the results will be rerun using a regression tree algorithm [Breiman and Breiman 1984] which will then be compared to K-NN. The regression tree algorithm does automatic feature selection when building the decision trees, making it a good contrast to the K-NN algorithm.

3.6 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 3.4). 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 give on how well each one did. Next, the simulation is run again but 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 needs to be done. Otherwise additional feedback can be given or additional scenarios setup 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.

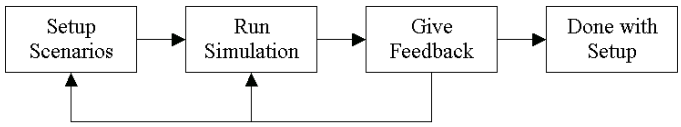


Figure 3.4: The process the animator uses to give feedback to the system.

To give an example of this process we will look at the flocking case study in section 4.3. The goal of this case study is to allow the virtual birds to flock together in a V-formation while avoiding the obstacles and following the flock leader who will be performing various stunts. There are three behavioral models $\{B_1, B_2, B_3\}$ that excel at each individual task.

B_1 A "boid" model which avoids obstacles and flocks towards other virtual birds [Reynolds 1987].

B_2 A V-formation model that dynamically forms flocks in the shape of a V [Gervasi and Prencipe 2004].

B_3 A stunt model that can perform a number of different scripted stunts.

We first setup three scenarios that are most representative of what we are trying to accomplish. In the first scenario, S_1, the virtual bird is placed on a collision course with an obstacle. In the second scenario, S_2, the virtual bird is placed at the head of the flock. And in the third scenario, S_3, the virtual bird is placed around other birds as a follower. Once the scenarios are setup we run the simulation to determine how well each individual behavioral model works in the different scenarios.

In scenario S_1, model B_1 receives higher ratings the closer it gets to the obstacle because it was successfully avoiding it, while the other models (B_2, B_3) receive neutral ratings because they would make no attempt to avoid the obstacle. In scenario S_2, model B_3 is rated the highest since we want the leader to perform stunts. Finally in scenario S_3 model B_2 is rated the most favorable due to our desire to have them form a V-formation.

After the first round of training, the simulation is run and the results observed. The birds generally performed quite well by avoiding obstacles, flocking in a V-formation and performing stunts. However the birds following in the V-formation would begin to avoid an obstacle but stop too soon, ignore the obstacle, and get back into a perfect V, hitting the obstacle. After running the initial simulation, additional feedback is given and model B_2 is given a lower rating, and model B_1 is given a much higher rating in the problem areas. After giving the feedback the birds performed as desired—they avoided the obstacle while maintaining a V-formation the best they could. Adjusting the simulation also brought out another situation that had not been handled to that point by our scenarios—the case of a single bird. In this situation it was decided to have the bird do stunts until it found other nearby birds. That scenario was created and feedback was given favoring B_3.

After the additional scenario was created and feedback given the results were found to be satisfactory and we were done training the system. With the framework

trained, we could then test the system in a variety of settings to see how well the ensemble controller responded. We were able to achieve satisfactory results (see section 4.3.3) and the process took a little over 30 minutes.

The following chapter explores how the framework performed in a variety of circumstances. To create the model ensembles, the system had to be trained, as outlined above, for each case study. All together the case studies provide an array of difficult challenges to stress the system in different ways allowing us to gauge its performance and to test its validity.

Chapter 4

Feasibility Study, Case Studies, and Comparisons

This chapter contains the results of using the framework in a number of different scenarios. First a feasibility study is carried out to test the merit of the idea. The particular application used for the feasibility study is the game of capture the flag, using a single agent. Following the feasibility study several case studies are described which use the framework. The following case studies were implemented:

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

Each case study describes the unique challenges it provides, relates the cognitive or behavioral models used, and presents the results. In addition, a comparison between K-NN and the regression tree algorithm is provided which evaluates the impact the machine learning algorithm has on the framework.

4.1 Feasibility Study: Simple Capture the Flag

4.1.1 Introduction & Environment

The goal of this study is to verify the feasibility of our proposed framework and explore the different parameters within the framework. With this goal in mind the study is designed to be simple. The environment consists of a discrete 16x16 world (Figure 4.1) with one blue agent, one blue flag, and one red flag. The goal is for the agent to retrieve the red flag and return it to where the blue flag is. Feedback is only given on one instance of the map (i.e. both flags are not moved for the duration of the system training). The ensemble controller then attempts to use the context sensitive weights to choose the best behavioral model, which should lead the character to capture the red flag and return it to the blue base.

The state representation for this study consists of the following variables:
$\vec{P}$: The x and y position of the agent in the world.
$\vec{R}$: The translation vector from the agent's current position to the enemy's flag.
$\vec{B}$: The translation vector from the 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.

4.1.2 Behaviorial Models

Four very simple behavioral models are used. The four models are labeled B_1, B_2, B_3, and B_4. Each model is unintelligent and always performs the same action regardless of the current state of the world. B_1 moves the agent to the left, B_2 moves the agent to the right, B_3 moves the agent up, and B_4 moves the agent down. Because each model will only move in one direction it is not possible for any one single model to successfully capture the flag.

4.1.3 Results

The above models are used to capture the flag despite the fact that neither the ensemble controller, the behavioral models, nor any single part of the system has

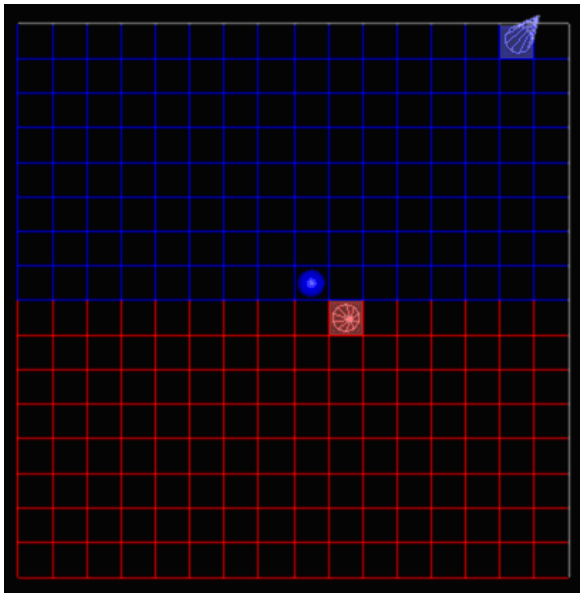


Figure 4.1: Shown here is the CTF environment.

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. Therefore feedback will be given to the system rating highly models that move the agent towards the red flag and then back towards the blue base after picking up the flag.

To measure the performance of the framework, the agent and the two flags are randomly placed in the world. The agent is then given the opportunity to capture the flag within 200 turns and the simulation is ended when the flag is captured or the 200 turn limit is reached. If the flag has not been captured the simulation is ended and it is recorded that no capture was made. At least 500 rounds of testing are done for each combination of parameters.

Running the Simulation with the Initial Feedback

The results in Table 4.1 show that the framework was able to correctly learn the context sensitive weights and successfully capture the flag. When choosing the top model the character was able to capture the flag 84.1% of the time. 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. Temporal boosting had a negative effect on performance because it caused the agent to stay with a particular model and overshoot its desired destination increasing the likelihood of finding a rut.

Probabilistically choosing the model resulted in better performance, a 99.2% capture rate. The capture rate is much higher because of the stochastic nature of using the PDF to select the model—the agent will eventually choose an action that gets it out of any ruts. Unfortunately this caused the agent to look unintelligent as its movements tended to be stuttered and jerky. Statistically this can be seen in the higher number of turns needed to capture the flag. When choosing the top model the agent was able to go to the flag, pick it up, and return it without hesitation unlike the stochastic method. Temporal boosting reduced the agent's stuttering and made it appear more intelligent but it still continued to overshoot its goals wasting turns. The only time the agent did not capture the flag was when the 200 turn limit was reached.

Table 4.1: Capture the Flag Results

Parameters	Capture Rate	Avg. Number of Turns
Choose top action without temporal boosting	84.1%	27.1
Choose top action with temporal boosting	58.2%	36.2
Probabilistically choose the action without temporal boosting	99.2%	85.2
Probabilistically choose the action with temporal boosting	96.4%	97.0

Running the Simulation with an Additional Round of Feedback

The framework can be given feedback at anytime to improve performance. The CTF simulation runs were repeated, but this time additional feedback was given

whenever the agent got stuck in a rut. Table 4.2 shows that the agent did improve its performance particularly when choosing the top model without temporal boosting. It went from scoring 84.1% to 98.2%. Giving additional feedback can be an effective method to increase the agent's performance.

Table 4.2: Capture the Flag Results with Additional Feedback

Parameters	Capture Rate	Avg. Number of Turns
Choose top model without temporal boosting	98.2%	31.2
Choose top model with temporal boosting	51.2%	34.8
Probabilistically choose the model without temporal boosting	100%	73.1
Probabilistically choose the model with temporal boosting	99.6%	91.1

4.1.4 Conclusion

The framework was able to successfully combine the strengths of each individual model and successfully perform a task that none of them could possibly do alone. There was a transfer of knowledge from the animator to the system enabling the character to successfully capture the flag. In addition, the character's performance continued to improve through multiple rounds of feedback. The feasibility study shows that the framework is able to learn the animator's preferences, improve the performance of the character, and create a new behavior.

4.2 Case Study 1: Multi-Agent Capture the Flag

4.2.1 Introduction & Environment

This study is built upon the feasibility study but adds additional elements to test the framework. Multiple agents are introduced, the world is enlarged by a factor of 4, and impassible squares are added (see Figure 4.2). Specifically, the world is a 32x32 grid with 10% of the board randomly covered in obstacles. There are two blue agents and two red agents along with a flag for each team.

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

$\vec{P}$: The x and y position of the agent in the world.

$\vec{R}$: The translation vector from the agent's current position to the enemy's flag.

$\vec{B}$: The translation vector from the 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.

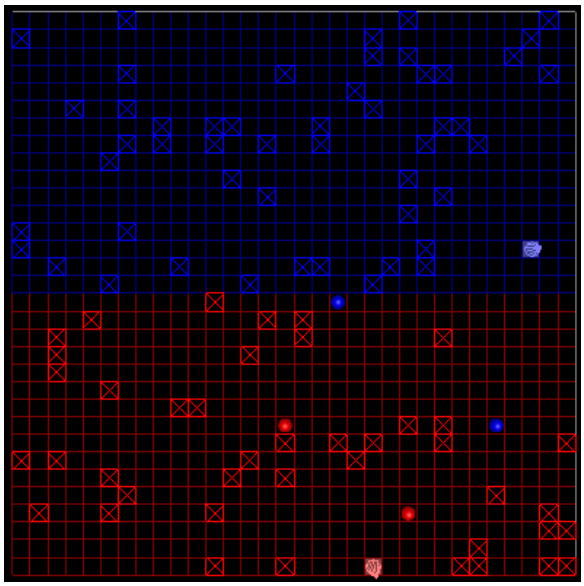


Figure 4.2: Shown here is the multi-agent CTF environment. The world is significantly larger (32x32), includes obstacles, and has multiple agents.

4.2.2 Behavioral Models

In order to deal with the more complex world, three new behavioral models were created.

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, Khatib 1986]. 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 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.

4.2.3 Results

For each simulation the map is randomly generated with 10% of the board covered in obstacles, the flags are randomly placed with the surrounding obstacles removed, and the agents are randomly placed on their side of the map. 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 600 rounds are simulated between the teams while recording the number of times each team captured the flag and the number of turns to capture the flag.

In order to establish baseline performance each cognitive model was tested against itself. The expected value with a random world and randomly placed agents is an even split between the two teams. The results (Table 4.3) confirm that the teams did split the captures when using the same models. With the arc defense model, both teams are only defending their flag and no captures are made.

To test the framework, one of the blue agents was given an ensemble of the three models while the other agents were restricted to one of the other models. The

blue agent with the ensemble controller had a notable impact on the results. When the other agents were using the A* model it resulted in a 8.4% spread between the teams—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 models the spread was not as large but was still in blue's favor at 4.0%. Finally with the arc defense models there was a 100% spread between the teams because red never attempted to capture the blue team's flag. In this run the blue team captured the flag 19.7% of the time. The ensemble of behavior models increased the agent's performance demonstrating the effectiveness of the framework.

Table 4.3: Multi-Agent Capture the Flag Results

Cognitive Models	Baseline Results	Ensemble Results
A* model.	Red: 50.2% Blue: 49.8% Total Captures: 100/100 Average Turns: 46.4	Red: 45.8% Blue*: 54.2% Total Captures: 99.5/100 Average Turns: 51.4
Potential fields model.	Red: 49.2% Blue: 50.8% Total Captures: 88.5/100 Average Turns: 117.6	Red: 48.0% Blue*: 52.0% Total Captures: 90.2/100 Average Turns: 109.7
Arc defense model.	Red: 0% Blue: 0% Total Captures: 0/100 Average Turns: 500	Red: 0% Blue*: 100% Total Captures: 19.7/100 Average Turns: 430.6

*One agent on the blue team was using an ensemble.

4.2.4 Conclusion

In this case study, a multi-agent capture the flag game was simulated. One agent used the ensemble controller while the other three agents were restricted to a single model. In all cases the agent with the ensemble controller performed better than the agents using a single model. The results show that the ensemble character had an impact on the outcome of the simulation. This study suggests that the ensemble

controller framework can improve the performance of an agent in a complex, multi-agent environment.

4.3 Case Study 2: Flocking

4.3.1 Introduction & Environment

The flocking simulation consists of a large area with various trees, plants, mountains and virtual birds—see Figure 4.3. The birds attempt to flock together while avoiding the obstacles in the environment (Figure 4.4). In the case study presented in the previous section, the best action was chosen based on its weighting. In this case study, we augment the action of the ensemble controller to be able to blend actions together to create novel actions. This is possible because the models output real number acceleration vectors which can be combined. This study uses a continuous environment, and of all the case studies presented here, contains the largest number of concurrent virtual characters running multiple behavior models, stressing the performance of the solution.

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.

4.3.2 Behavioral Models

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 has three simple steering behaviors that control how the individual boid maneuvers based on the positions and velocities of nearby flock mates. The three steering behaviors are cohesion (the attraction of boids to each other), alignment (match the velocity and heading of flock mates) and separation (avoiding any crowding of local flock mates). These forces allow the boids to flock based completely on their local perception of the world. In addition to the three basic steering behaviors, an obstacle avoidance behavior is also included allowing the boids to dodge

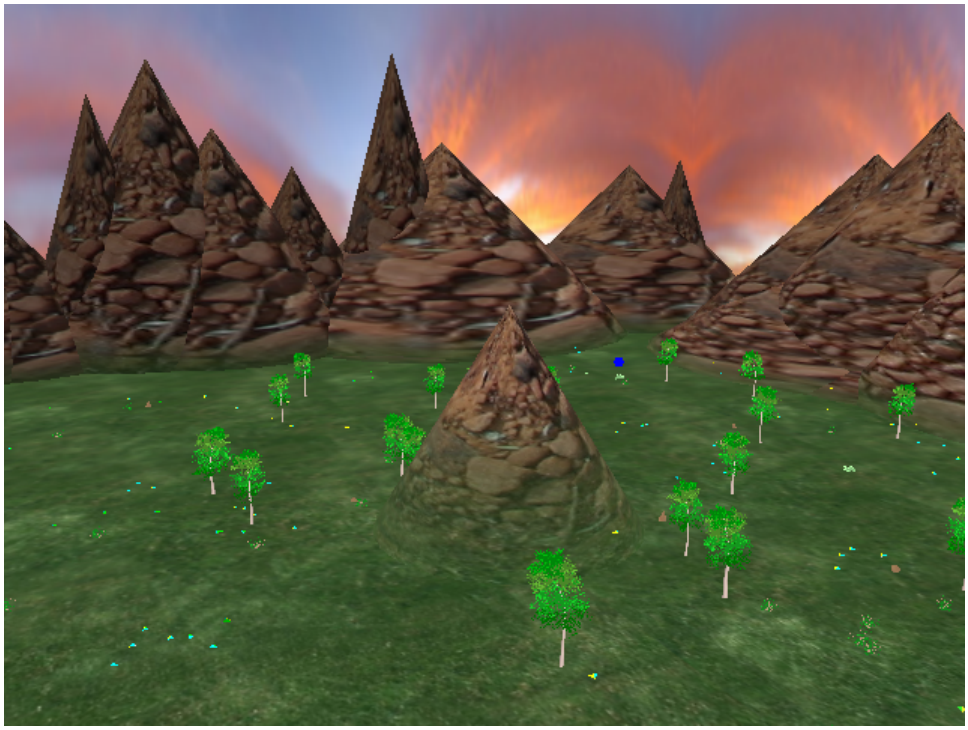


Figure 4.3: The flocking world is made up of trees, plants, mountains and birds. When the birds reach the edge of the area, represented by the surrounding mountains, they wrap around to the other side. There are different obstacles the birds must avoid—the large mountain in the middle of the field and the randomly placed trees.

obstacles. A base implementation used in [Platt 2004] was slightly modified to fit this study's particular environment.

A V-formation model [Gervasi and Prencipe 2004]. Gervasi and Prencipe's work involves the distributed coordination and control of anonymous, memoryless virtual birds that can freely move in a two-dimensional plane with a designated leader. Their algorithm was used as a basis to form a V-formation model that works in a three-dimensional world and without any predesignated leaders. In addition, the work of Leonard and Fiorelli [2001] on coordinating groups of autonomous vehicles was used as a reference for this model.

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.

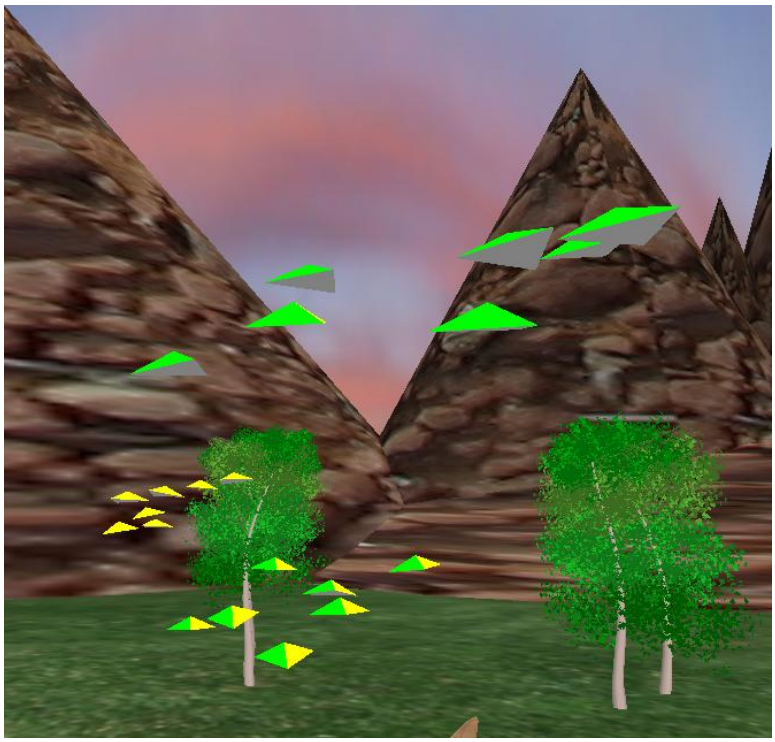


Figure 4.4: Three groups of flocking birds can be seen here. They attempt to stay together while avoiding obstacles such as trees and mountains.

4.3.3 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 (see Figure 4.5); consequentially, the birds do not start out in flocks and must seek nearby birds to form flocks.

Statistical Results

To gather the results, 3 different simulations were done for 3 minutes each, and their outputs were averaged. Table 4.4 shows the number of collisions per second with

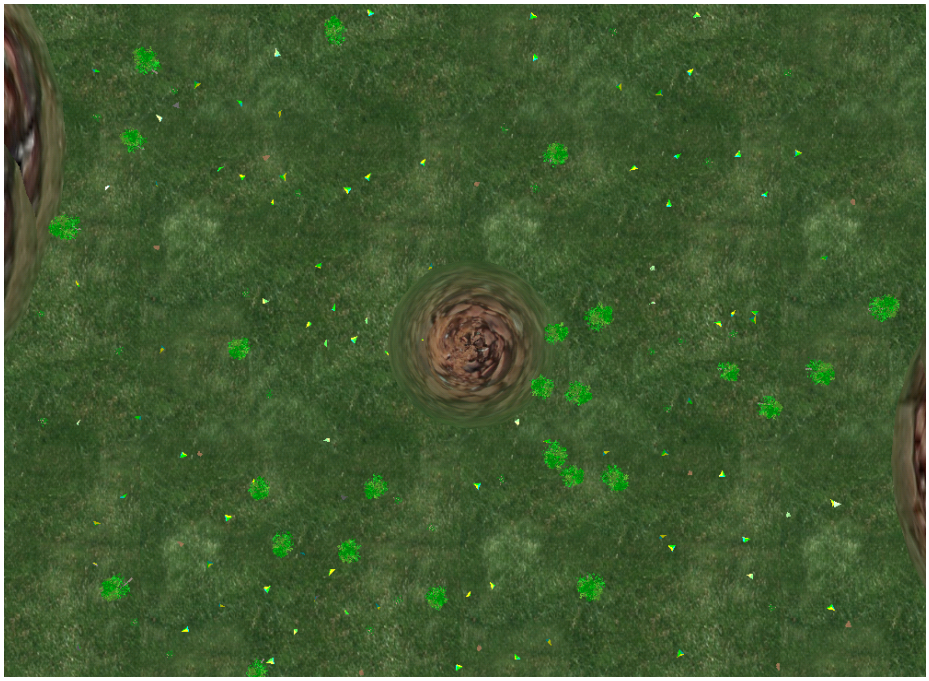


Figure 4.5: When starting a simulation the virtual birds are placed randomly on the map with random horizontal and vertical headings. The birds must actively seek out nearby birds to form flocks.

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 rate of 16.25 and 8.43 per second respectively, which was much higher than the boid model rate of 3.0 collisions per second. When the birds are using an ensemble of the models they achieved a rate of 3.23 collisions per second which is very close to our baseline even though they are using all three models at the same time. The ensemble was able to successfully learn the animator's preference to avoid the obstacles thus meeting our goals statistically.

Table 4.4: Flocking Results

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

Visual Results

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. The visual goals can be considered subjective in nature because there is no metric to define how well the birds are meeting the goals and also the written format of this work makes it hard to convey the degree to which the goals are met. Although subjective, it is important for birds to meet our visual goals or little has been gained by using the ensemble. In Figure 4.6 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 4.7). The new action allows the birds to successfully avoid the tree when close, but to get back into formation shortly after—thus meeting our primary visual goal.

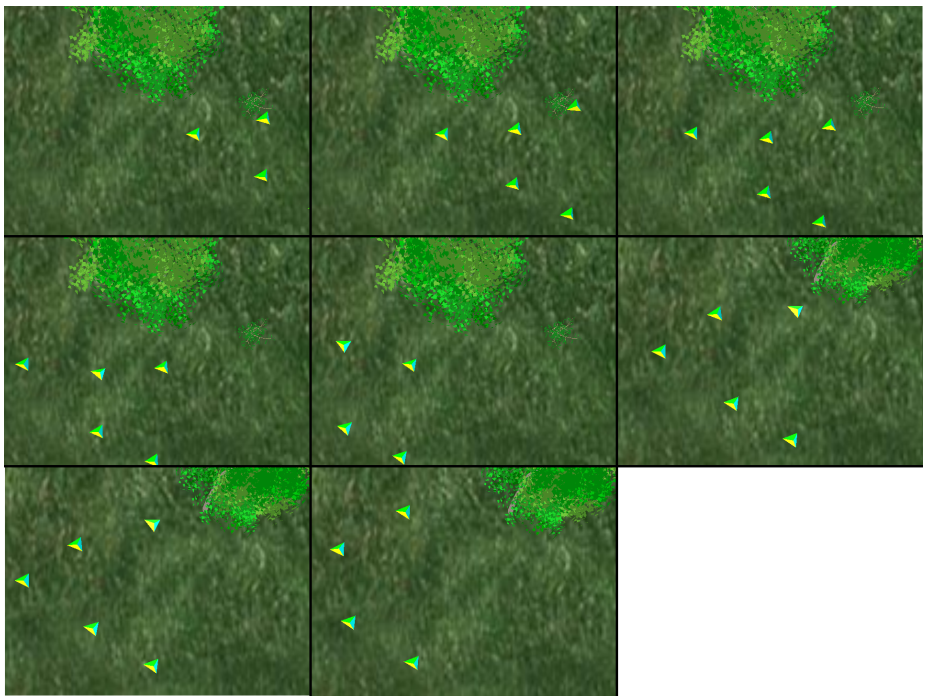


Figure 4.6: The birds are able to keep in formation while avoiding the obstacle.

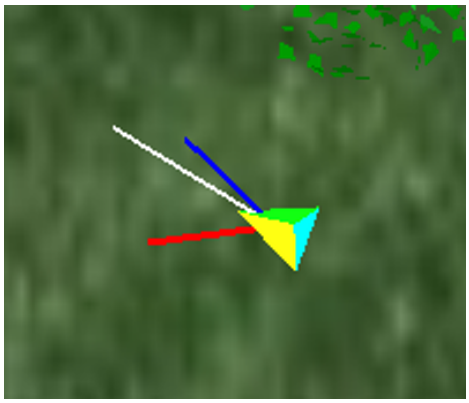


Figure 4.7: Seen here is a bird and the suggested actions of the models as well as the chosen action. The blue, red, and green (not visible) lines represent the different suggested actions by the behavioral models and the white line represents the actual action taken. As can be seen, the final action is a weighted combination of the different suggested actions.

The bird that is currently leading has also successfully learned not only to avoid obstacles but to perform various stunts (Figure 4.8) meeting our second goal. Figure 4.9 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. Statistically we know the birds are successfully avoiding the obstacles. To get to this point in time, the birds had to first find nearby birds, second dynamically form flocks, and third form a V-formation. As can be seen in Figure 4.9 they are avoiding the obstacles, flocking in V-formations, and performing various stunts.

4.3.4 Conclusion

The continuous environment in this case study presented an important challenge that the framework was able to overcome. The ensemble characters were a success and met the statistical goals and the subjective visual goals. The linear blending of actions not only produced completely new actions not suggested by any single model but also increased the performance of the birds. They were able to avoid obstacles while at the same time staying in formation. Finally, there was no significant performance decrease with the ensemble characters.

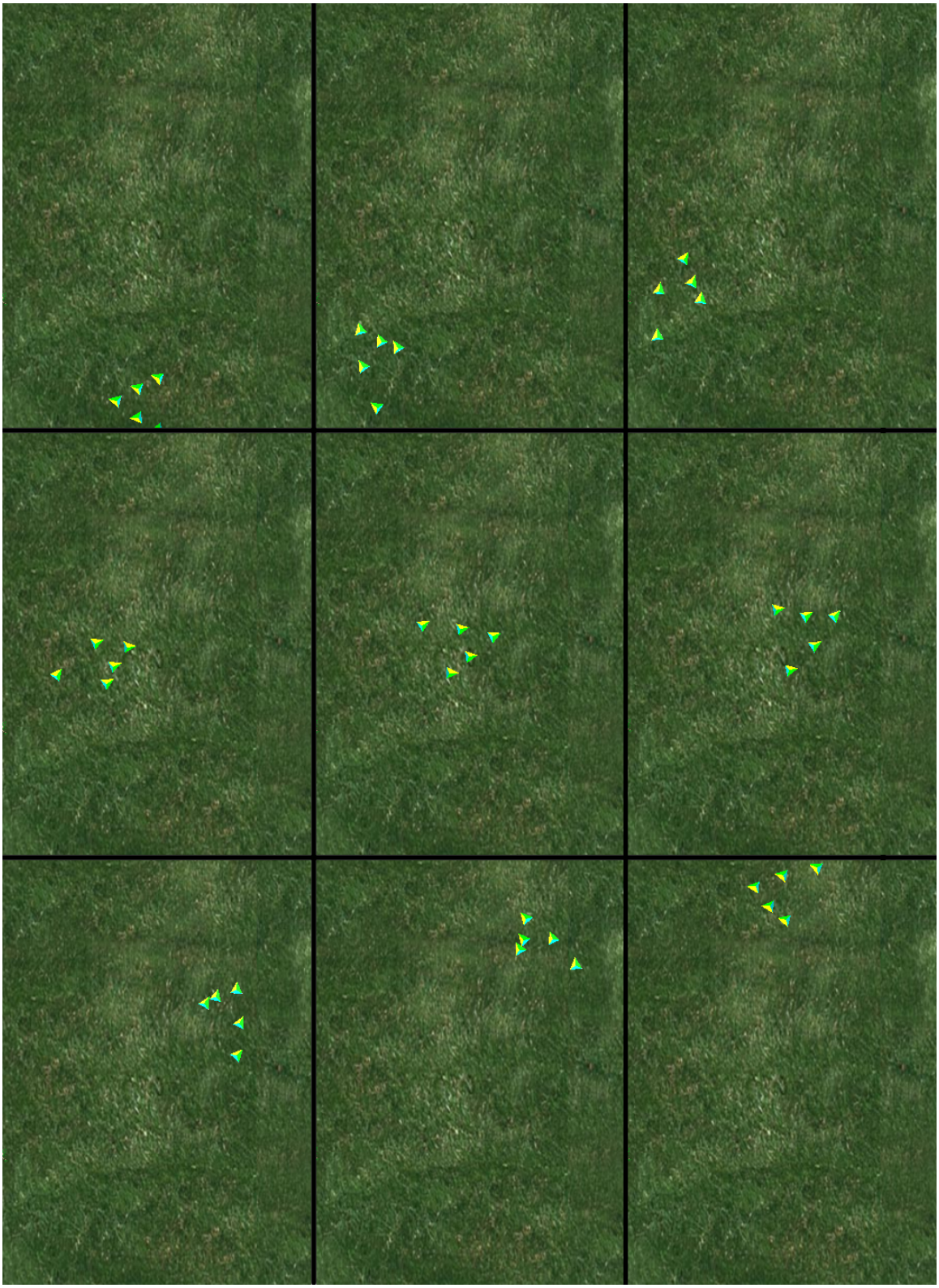
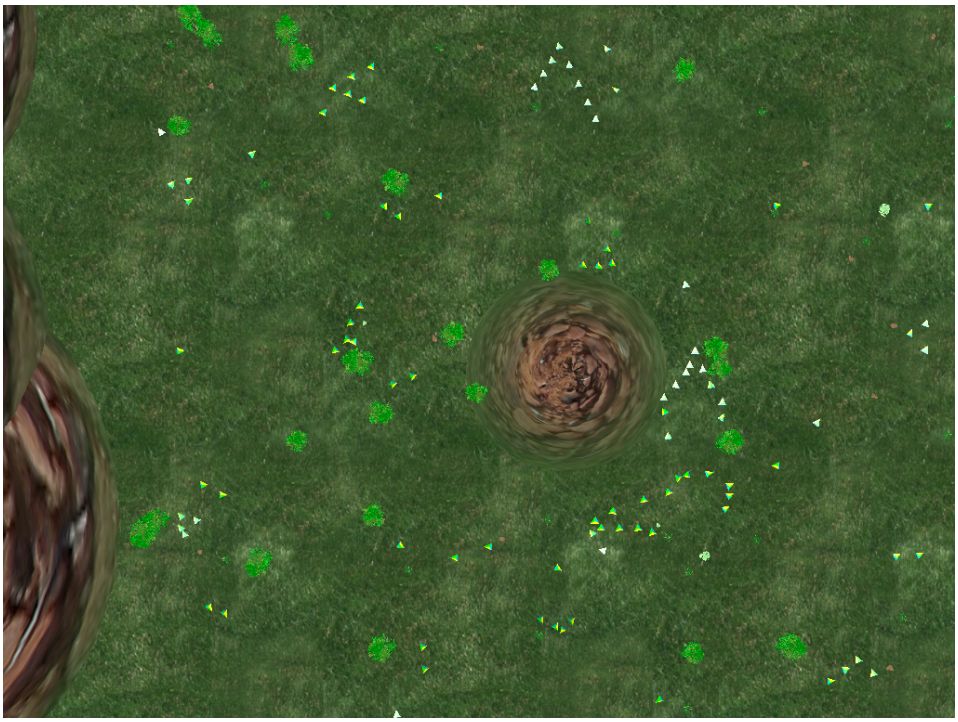
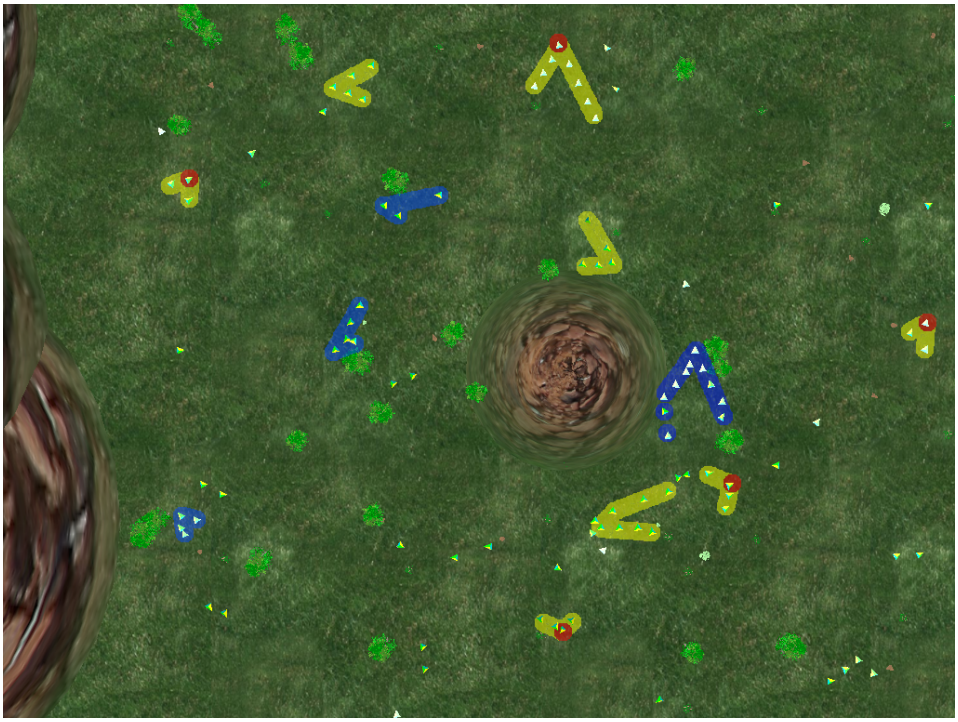


Figure 4.8: The leading bird is performing a figure S stunt with the other birds following. The birds following are able to stay in formation throughout the maneuver.



(a) Field



(b) Highlighted Field

Figure 4.9: Subfigure (a) shows the field with all 100 birds using the ensemble controller, and subfigure (b) shows the same map 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.

4.4 Case Study 3: Crowd Simulation Using a Combat Situation

4.4.1 Introduction & Environment

The object of this case study is to test the framework in a continuous environment, with complex characters, with multiple goals, and with the layering of ensemble controllers. The environment is a large continuous area where one army attempts to secure two target locations on the map. The attacking team has three different types of characters—a paladin, a mage, and a nurse—while the defending team is a horde of skeletons. A new character is created using an ensemble of the characters to create a "super character" that can perform all of the actions of the paladin, mage, and nurse (the top action is chosen). 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.

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.

4.4.2 Troop Cognitive Models

The are five different characters created for this case study each with 100 health points and different abilities. (see Figure 4.11).

- The paladin is a close combat character that actively seeks out nearby enemies and has the strongest attack of any character—Figure 4.11(a).
- The mage can attack from a distance by throwing fireballs at the enemy—Figure 4.11(b). Although her attack isn't as strong as the Paladin's, because

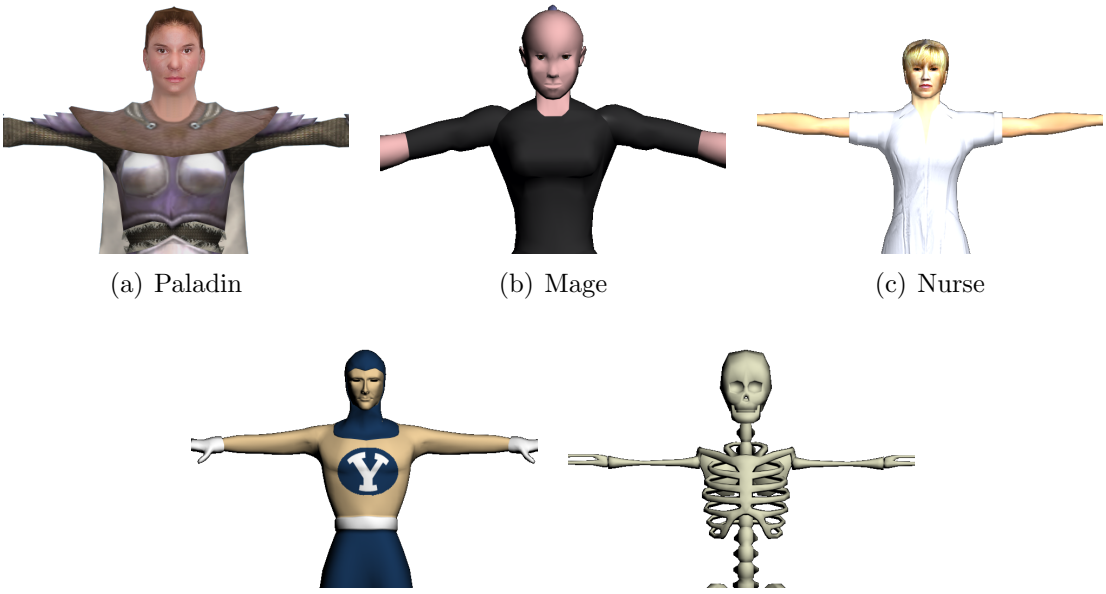
she can attack from a distance before the Paladin can, it is an even fight between the two.

- The Nurse heals other characters around her but is unable to attack the enemy or heal herself—Figure 4.11(c).
- A "super character" is created that uses an ensemble of the above cognitive models to perform the actions of all three—Figure 4.11(d).
- The skeleton character is similar to the paladin. It is only able to attack at close range and is similar in strength—Figure 4.11(e).

Figure 4.10: Shown here is the combat situation environment. The super character is attempting to secure the location specified by the target.

4.4.3 Commander Cognitive Models

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



(a) Paladin (b) Mage (c) Nurse

(d) Super Character (e) Skeleton

Figure 4.11: This figure shows each character present in the case study.

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. It 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 skeletons patrols the area around the two target locations and attacks when the other team gets close.

4.4.4 Results

The attacking team is made up of 12 characters—4 paladins, 4 mages, and 4 nurses. The defending team is made up of 8 skeletons. 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 4.5).

For the ensemble simulations, two paladins and two mages 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 following statistics were gathered:

Simulation Time. The amount of time it took before an ending condition was reached.

Winning Percentage. The overall winning percentage of the attacking team.

Defending Team Damage. The amount of damage the defending team took from the attacking characters.

Defending Team Deaths. The number of characters on the defending team that died in the battle.

Attacking Team Damage. The amount of damage the attacking team took.

Attacking Team Deaths. The number of characters on the attacking team that died in battle.

Attacking Team Healing. The amount of health that was restored to the attacking team characters.

A two sided t-test was performed on each variable and is only considered significant above the 95th percentile. Fifty-one battles were simulated for each commander with and without the super characters.

Table 4.5 shows the results with the rush commander. The super characters had a large impact on battles. Not only did the winning percentage rise from 58.8% to 78.4% but the attacking team performed significantly better in almost every category showing that the super character was performing the unique skills of all three cognitive models.

Table 4.6 shows the results with the end-around commander. Compared to the rush commander, the end-around commander (80.4% compared to 58.8%) has a much better winning percentage. Even though the base simulation had a high

Table 4.5: Simulation Results With Rush Commander

Statistic	Without Ensembles	With Ensembles	T-Test
Simulation Time	170.6s	181.7s	0.123
Winning Percentage	58.8%	78.4%	**0.033**
Defending Team Damage	664.1	805.2	$\mathbf{2.07 * 10^{-4}}$
Defending Team Deaths	5.7	7.5	$\mathbf{5.20 * 10^{-5}}$
Attacking Team Damage	1087.4	1020.8	0.34
Attacking Team Deaths	8.6	7.0	**0.024**
Attacking Team Healing	163.4	229.6	$\mathbf{5.44 * 10^{-5}}$

winning percentage, adding the super character still had a large impact. The winning percentage with the super characters rose to 98.0%.

Table 4.6: Simulation Results With the End-Around Commander

Statistic	Without Ensembles	With Ensembles	T-Test
Simulation Time	222.9s	232.6s	0.42
Winning Percentage	80.4%	98.0%	**0.0038**
Defending Team Damage	732.9	731.8	0.98
Defending Team Deaths	6.7	7.0	0.56
Attacking Team Damage	891.5	765.4	.078
Attacking Team Deaths	6.4	4.6	**0.0061**
Attacking Team Healing	182.7	212.7	0.15

In addition to creating an ensemble of the troop cognitive models, an ensemble of the commander cognitive models was also made. The results show that without using a commander ensemble, the end-around commander has a much higher winning percentage (80.4% to 58.8%) but is a lot slower than the rush commander (222.9s to 170.6s). 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.

The same simulation procedures were used—51 battles for each commander with and without the super characters—and the results were recorded. Table 4.7 shows the results of the simulation using ensemble commanders but not ensemble

troops. Without super characters the ensemble of the commander cognitive models performed as desired. While not as fast as the rush commander, the simulation time was reduced significantly from the end-around commander's time of 222.9s to 193.7s. Also the winning percentage jumped to 88.2%, which was better than either individual commander. Table 4.8 shows the results of using both ensemble commanders and ensemble troops. This simulation produced the best results. The commander won quickly and often and provided the best overall performance seen in the case study.

Table 4.7: Simulation Results With 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
Defending Team Damage	781.3	664.1	0.077	732.9	0.23
Defending Team Deaths	7.2	5.7	**0.0019**	6.73	0.25
Attacking Team Damage	874.4	1087.4	**0.0045**	891.5	0.80
Attacking Team Deaths	6.2	8.6	**0.00076**	6.35	0.79
Attacking Team Healing	178.8	163.4	0.35	182.7	0.83

Table 4.8: Simulation Results With 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
Defending Team Damage	761.8	805.2	0.17	731.8	0.55
Defending Team Deaths	7.2	7.5	0.39	7.01	0.71
Attacking Team Damage	800.2	1020.8	**0.00030**	765.16	0.59
Attacking Team Deaths	5.2	7.02	**0.0035**	4.63	0.29
Attacking Team Healing	190.9	229.7	**0.013**	212.6	0.26

4.4.5 Conclusion

In this crowd simulation study, an attacking team made up of three different character types (paladin, mage, and nurse) attempts to secure two target locations on the map defended by a horde of skeletons. Overall, incorporating the framework into this case study worked well. Performance was improved in nearly every aspect and combining multiple model ensembles brought even better performance. The results were found to be statistically significant at the 95th percentile. In addition, the super character performed the unique actions of all three behavioral models allowing for new behaviors and animations.

4.5 Machine Learning Algorithms

4.5.1 Algorithms

The machine learning algorithm used to learn the context sensitive rating in all of the case studies was k-nearest neighbors (K-NN) [Cover and Hart 1967]. K-NN was chosen because of its robustness to noise and its ability to easily work in continuous environments. This is important because the feedback given to the framework comes directly from the animator. He/she will most likely not give consistent feedback resulting in noisy training data.

To test the impact the machine learning algorithm has on the framework the case studies were rerun using the regression tree algorithm [Breiman and Breiman 1984]. Only t-test scores above the 95th percentile are considered significant when comparing the algorithms. A regression tree is quite different from the K-NN learner. This algorithm attempts to predict the regression variable by partitioning the data using a tree of split (if-then-else) conditions. When using continuous variables the regression tree is only able to provide a rough approximation because it must choose a split point. In contrast, K-NN simply looks at its closest K neighbors and uses a weighted combination of their values to predict the regression variable. Like K-NN, regression trees have moderate to quick training times and are moderately robust to noise.

4.5.2 Feasibility Study Results

Tables 4.9 and 4.10 show the results with the top model being chosen and Tables 4.11 and 4.12 show the results when probabilistically choosing the model. In three of the four cases, K-NN performed statistically better than the regression tree. The training data for this case study was particularly noisy, which most likely reason K-NN to perform better.

Table 4.9: Feasibility Study Learning Algorithm Comparison with Top Model Chosen and Temporal Boosting

	K-NN	Regression Tree	T-Test
Capture Rate	58.3%	52.8%	0.11
Number of Turns	34.9	36.2	0.26

Table 4.10: Feasibility Study Learning Algorithm Comparison with Top Model Chosen and No Temporal Boosting

	K-NN	Regression Tree	T-Test
Capture Rate	84.1%	78.7%	**0.046**
Number of Turns	27.4	36.2	0.57

Table 4.11: Feasibility Study Learning Algorithm Comparison with Probabilistically Choosing the model and Temporal Boosting

	K-NN	Regression Tree	T-Test
Capture Rate	96.4%	91.2%	**0.0005**
Number of Turns	97.02	94.6	0.34

Table 4.12: Feasibility Study Learning Algorithm Comparison with Probabilistically Choosing the model and No Temporal Boosting

	K-NN	Regression Tree	T-Test
Capture Rate	99.5%	92.1%	$\mathbf{4.53 * 10^{-8}}$
Number of Turns	97.02	94.6	0.037

4.5.3 Capture The Flag Results

The A* model was used for comparison between the learning algorithms. The blue team had one character using the model ensembles and the same simulation parameters were used (see Section 4.2.3). Unlike the feasibility study this study showed no statistical difference between the learning algorithms (see Table 4.13). The t-test scores were all very high, greater than 0.32, and the scores had small differences.

Table 4.13: Capture the Flag Learning Algorithm Comparison

	K-NN	Regression Tree	T-Test
Blue Winning Percentage	53.7%	52.4%	0.64
Red Winning Percentage	46.9%	47.4%	0.57
Total Capture Percentage	99.6%	99.9%	0.35
Number of Turns	51.2	49.9	0.32

4.5.4 Flocking Results

The flocking case study favored the K-NN learner (see Table 4.14). The regression tree learner still performed quite well with a collision rate of 5.01 which is much better than the 16.25 and 8.43 collisions per second of the V-formation model and Stunt model respectively. With the K-NN learner, when approaching an obstacle the bird would begin to avoid it while trying to stay in formation until it had completely passed the obstacle. With the regression tree, there appeared to be a point where the bird would completely stop trying to avoid the obstacle and would get back into formation. This is most likely due to where the split point was chosen on the continuous attribute. However, despite this, the regression tree still did quite well.

Table 4.14: Flocking Learning Algorithm Comparison

	K-NN	Regression Tree	T-Test
Collision Rate	3.23	5.01	**0.049**

4.5.5 Crowd Simulation Results

The crowd simulation results were rerun with the regression tree (Tables 4.15 and 4.16). With just the commander cognitive models there was no significant difference between the two algorithms. When both the commander and troops use an ensemble of the models the regression tree actually performed significantly better in four of the seven categories. This case study helps to show that there will be different domains where each learner will perform better than the others.

Table 4.15: Crowd Simulation Learning Algorithm Comparison with Ensemble Commanders and No Ensemble Troops

Statistic	K-NN	Regression Tree	T-Test
Simulation Time	193.7s	191.6s	0.77
Winning Percentage	88.2%	84.3%	0.57
Defending TeamDamage	781.3	763.6	0.63
Defending Team Deaths	7.2	7.0	0.61
Attacking TeamDamage	874.4	909.3	0.58
Attacking Team Deaths	6.2	6.4	0.73
Attacking Team Healing	178.8	180.4	0.92

Table 4.16: Crowd Simulation Learning Algorithm Comparison with Ensemble Commanders and Troops

Statistic	K-NN	Regression Tree	T-Test
Simulation Time	191.2s	200.5s	0.18
Winning Percentage	94.1%	98.0%	0.31
Defending TeamDamage	761.8	829.8	**0.030**
Defending Team Deaths	7.2	7.9	**0.021**
Attacking TeamDamage	800.2	666.6	**0.018**
Attacking Team Deaths	5.2	3.9	**0.012**
Attacking Team Healing	190.9	183.6	0.63

4.5.6 Conclusion

These particular case studies tend to slightly favor K-NN over regression trees. K-NN performed better in 2 of the 4 studies while regression trees performed better in 1 study with other study being a toss-up. However, the differences in performance were small even when statistically significant. Overall, both algorithms performed well and showed that the machine learning algorithm has a small impact on the framework. On the studies found in this thesis, the success of the framework depends more heavily on the animator giving accurate feedback than on the machine learning algorithm.

Chapter 5

Conclusion & Future Work

5.1 Conclusion

Limitations with 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 thesis 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 3.2, Figure 3.2). The ensemble controller is able to synergistically combine the individual models together (leveraging the strengths of each one) and improve the character's performance. The manner in which the models are combined together is determined by the animator's feedback given to the system and varies depending upon the current environment state (sections 3.3, 3.5).

A simple capture the flag feasibility study was performed (section 4.1) that showed the framework is able to correctly learn the feedback given by the animator and allow the character to perform a new task—capture the flag. The next study, multi-agent capture the flag (section 4.2), built upon the first study and included a more complex environment and multiple agents. The flocking case study (section 4.3) and the crowd simulation using a combat situation case study (section 4.4) continued to stress the framework by introducing a continuous environment, complex cognitive models, large numbers of agents, layering of model ensembles, and blending of suggested actions. In each of the case studies the framework was empirically shown to increase the character's performance. Furthermore the characters exhibited new behaviors and animations not present in the individual models.

The framework successfully leveraged multiple cognitive models. 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 the feedback given to the system.

This thesis 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.

5.2 Future Work

The challenges of creating cognitive and behavioral model ensembles has only begun to be explored. Future work includes the exploration of different learning algorithms and their impact on performance, how best to weight each model—perhaps a weight scheme more like boosting (or any one of the many different weight techniques), a rigorus mathematical analysis of the system, analysis of the scalability of the system to thousands of agents, what constitutes a good state representation, and how to improve animator control over the character. Further work could be done to reduce the burden on the animator. The current framework requires the animator to give feedback on each model individually. A more ideal system would only ask for feedback when the animator disliked a particular model and would then automatically explore the alternative models. Finally, given the enormous variability in how cognitive and behavioral models are created and integrated into the computer graphics modeling hierarchy, this framework will not work for all models. Additional work could be done in those situations to determine how best to implement an ensemble framework.

The ensemble controller bears a resemblance to a perceptron [Rosenblatt 1958]. Perceptrons are not among the most powerful machine learning techniques. The ensemble controller could potentially be replaced with a high order machine learning

technique. Challenges to this approach would include keeping the feedback mechanism easy and simple, figuring out how to do the model selection, and developing a system that would not require domain knowledge of the problem (keeping the environment and cognitive models as black boxes).

Bibliography

Andrews, J. R., and Hogan, N. 1983. Impedance Control as a Framework for Implementing Obstacle Avoidance in a Manipulator. *Control of Manufacturing Processes and Robotic Systems*, 243–251.

Bellman, R. 1961. *Adaptive Control Processes: A Guided Tour*. Princeton University Press.

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

Blumberg, B., Downie, M., Ivanov, Y., Berlin, M., Johnson, M., and Tomlinson, B. 2002. Integrated learning for interactive synthetic characters. *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 417–426.

Breiman, B., and Breiman, L. 1984. *Classification and Regression Trees*. Chapman & Hall/CRC.

Breiman, L. 1996. Bagging Predictors. *Machine Learning 24*, 2, 123–140.

Burke, R., Isla, D., Downie, M., Ivanov, Y., and Blumberg, B. 2001. CreatureSmarts: The Art and Architecture of a Virtual Brain. *Proceedings of the Game Developers Conference*, 147–166.

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

Dietterich, T. 1997. Machine learning research: Four current directions. *AI Magazine 18*, 4, 97–136.

Dinerstein, J., and Egbert, P. 2005a. Fast Multi-level Adaptation for Interactive Autonomous Characters. *ACM Transactions on Graphics (TOG) 24*, 2, 262–288.

Dinerstein, J., Egbert, P., Garis, H., and Dinerstein, N. 2004. Fast and Learnable Behavioral and Cognitive Modeling for Virtual Character Animation. *Computer Animation and Virtual Worlds 15*, 2, 95–108.

Dinerstein, J., Ventura, D., and Egbert, P. 2005b. Fast and Robust Incremental Action Prediction for Interactive Agents. *Computational Intelligence 21*, 1, 90–110.

Faloutsos, P., van de Panne, M., and Terzopoulos, D. 2001. Composable controllers for physics-based character animation. In *SIGGRAPH 2001, Computer Graphics Proceedings*, ACM Press / ACM SIGGRAPH, E. Fiume, Ed., 251–260.

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

Funge, J., Tu, X., and Terzopoulos, D. 1999. Cognitive modeling: knowledge, reasoning and planning for intelligent characters. *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, 29–38.

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

Grzeszczuk, R., Terzopoulos, D., and Hinton, G. Neuroanimator: Fast neural network emulation and control of physics-based models. SIGGRAPH 98 Conference Proceedings. *Annual Conference Series*, 9–20.

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

Hart, P., Nilsson, N., and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems Science and Cybernetics 4*, 2, 100–107.

Isla, D., and Blumberg, B. 2002. New Challenges for Character-Based AI for Games. *AAAI Spring Symposium on AI and Interactive Entertainment, Palo Alto, CA, March*.

Jordan, M., and Jacobs, R. 1993. Hierarchical mixtures of experts and the EM algorithm. *Neural Networks, 1993. IJCNN'93-Nagoya. Proceedings of 1993 International Joint Conference on 2*.

Khatib, O. 1986. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. *The International Journal of Robotics Research 5*, 1, 90.

Leonard, N., and Fiorelli, E. 2001. Virtual leaders, artificial potentials and coordinated control of groups. *Decision and Control, 2001. Proceedings of the 40th IEEE Conference on 3*, 2968–2973.

Monzani, J., Caicedo, A., and Thalmann, D. 2001. Integrating Behavioural Animation Techniques. *Computer Graphics Forum 20*, 3, 309–318.

Opper, M., and Haussler, D. 1991. Generalization performance of Bayes optimal classification algorithm for learning a perceptron. *Physical Review Letters 66*, 20, 2677–2680.

Perlin, K., and Goldberg, A. 1996. Improv: a system for scripting interactive actors in virtual worlds. *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, 205–216.

Platt, R. 2004. 3D Boids Simulation. http://www.navgen.com/3d_boids/.

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

Rosenblatt, F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychol Rev 65*, 6, 386–408.

Stone, P., and Veloso, M. 2000. Multiagent Systems: A Survey from a Machine Learning Perspective. *Autonomous Robots 8*, 3, 345–383.

Tomlinson, B., Downie, M., Berlin, M., Gray, J., Wong, A., Burke, R., Isla, D., Ivanov, Y., Johnson, M., Lyons, D., et al. AlphaWolf. *Proceedings of SIGGRAPH 2001: conference abstracts and applications*, 2.

Tu, X., and Terzopoulos, D. 1994. Artificial fishes: physics, locomotion, perception, behavior. *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, 43–50.

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

Witten, I., and Frank, E. 2000. *Data mining: practical machine learning tools and techniques with Java implementations*. Morgan Kaufmann Publishers Inc. San Francisco, CA, USA.

Wolpert, D. 1992. Stacked generalization. *Neural Networks 5*, 2, 241–259.