2006

Emotion in flocking

Katherine Law

Masters Project Final Report
Emotion in Flocking

Katherine Law

June 4, 2004

Committee Members
Chairman: Joe Geigel
Reader: Nan Schaller
Observer: Walter Wolf

Abstract

Behavioral animation is a category of computer animation that enables objects to determine their own actions. This saves an animator from having to determine every detail of movement for each object in the animation. As the number of objects within an animation increases, specifying the position of each object becomes increasingly difficult. Flocking is an example of behavioral animation. Some examples of flocking can be seen in movies, such as the stampede of the antelopes in The Lion King, the herds of dinosaurs in Jurassic Park, and the massive battle scenes in The Lord of the Rings. In this project, I have enhanced flocking to convey emotion. This is achieved using only the movement of objects in relationship to one another. The effectiveness of the flock's motion in conveying a given emotion is judged by human observers via an on-line flocking system. This system allows users to rate how well an emotion is conveyed by the flock. User feedback is used to further adjust motion parameters with the goal of obtaining the best representative motion for each emotion.

Contents

List of Figures

1 Introduction

Behavioral animation allows an animator to program how an object or a group of objects will move without specifying each individual movement of an animation. An animator programs how an object will behave in a given situation, while the object determines when to behave in this manner. For example, if an object is moving forward and encounters an obstacle in its path, the object will turn away from the obstacle. The animator is responsible for programming the behavior when an obstacle is encountered, but the object executes the behavior only when this situation occurs. Flocking is a common use of behavioral animation.

Flocking is the coordination of animated objects so that they move together in a way that imitates the movement of a group of animals [9], such as a swarm of bees or a flock of birds. In the Computer Graphics world, objects within such groups are called *boids*. A flock consists of a group of *boids*. Each *boid* determines its own actions based on predefined steering behaviors. There are three basic steering behaviors [8]:

1. Separation - Avoid collisions with local *boids*.
2. Alignment - Move in the same direction as local *boids*.
3. Cohesion - Move towards the center of the local *boids*.

Each of these behaviors contributes to the movement of each *boid*. To create a realistic looking flock, the right balance of contribution and interleaving between these behaviors must be found.

My project adds emotion to flocking. In particular, I have developed a system that animates a swarm of bees to convey a selected emotion, such as angry or happy. The emotions are created by the way *boids* move in relationship to one another. The movement is modified by a set of steering behaviors. The contribution of each steering behavior can be set by the user. This allows for the best mix of steering behaviors for each emotion to be determined through feedback by the user.

2 Adding Emotion

It is difficult to create believable emotion in animation since emotion is an internal feeling, but behavior can be animated.

2.1 Behavior

According to *Webster.com*, behavior is defined to be "the response of an individual, group, or species to its environment." It is behavior that conveys emotion to a viewer.

Three steps are necessary for a behavior to take place [4]:

1. Perception - First, the agent must perceive its environment. This means being aware of the other objects or actors that are present. External events must also be captured.
2. Emotional reaction - The agent's emotion changes in response to its perception of the environment.
3. Physical reaction - Based on the emotion, the agent moves in a manner which conveys that emotion.

Repeating these steps encapsulates human reactions to events that transpire. For my project, the first two steps are irrelevant because the viewer will choose the emotion the flock is to portray. Thus, I have focused on the third step.

Animating a non-human object so that it behaves like a human is much easier than animating a group of objects to convey a human behavior. This is because a single object can be treated like a human body. With a flock, multiple objects must be animated in such a way that they work together to convey the appropriate message to a viewer.

2.2 Movement

Since behavior is a combination of movements, it is necessary to understand how movements can be combined to convey different emotions. Movement

is a form of communication. For example, if you see two people having a conversation, even if you don't know what is being said, you can fairly accurately guess their emotions. It is usually clear if they are happy or having a fight. This expression of emotion is being conveyed through body language, which is the foundation of how we interpret movement.

The best place to see movement portraying emotion can be found on the stage. Take for example dance. Spoken words are not necessary for a dancer to convey a message to an audience. Using only the movement of their body, dancers can tell entire stories without a sound.

According to Vaughan[10], movement can be broken down into four characteristics. These characteristics are:

1. Path - The path that the movement of an object follows can set a basis for emotion. For example, "while a straight line can convey formality and strength, a curved line can convey naturalness and warmth" [10]. Movement in a straight line conveys more purpose than a curved line. When we need to get something done, it is best to not wander around aimlessly.

2. Area - The amount of space that the movement fills can draw attention to it. For example, if someone is trying to flag down a taxi, they will wave their entire arm, maybe even jump. This will catch a driver's attention better than a head nod or a wave of just a hand.

3. Direction - The direction of the movement is also important. For example, a person running away from something suggests fright, while a person pacing back and forth suggests worry. The direction of the movement can tell a lot about what is happening, especially in conjunction with other objects in the environment helping to focus the viewer's attention.

4. Speed - The pace of the movement can be used to suggest a sense of urgency. The same movement at different speeds suggests different emotions. For example, if someone places a book upon a table, they appear to be calm and relaxed. Whereas, if they slam a book onto a table, they appear to be angry or frustrated. The speed of the movement can drastically change the meaning behind it.

It is the combination of these movement characteristics that creates behaviors and conveys emotion to a viewer. A small change in movement can completely change the viewer's perception.

It has been found that the more organic and complex the patterns of movement, the stronger the emotional response elicited from the viewer [10]. This is because people tend to interpret everything from their own context. People move in complex ways. Our body language combines facial expressions along with body positions and the actual movement of our limbs. In addition, as our movements are never exactly the same twice, it is difficult to program realistic movement.

3 Functional Specification

This application demonstrates how emotion can be conveyed using different movement patterns in flocking. The user may change the values of the given steering parameters, thereby changing the behavior of the flock. It lets the user rate how well they feel the selected emotion is expressed. Also, different values may be submitted that the user feels better expresses the selected emotion.

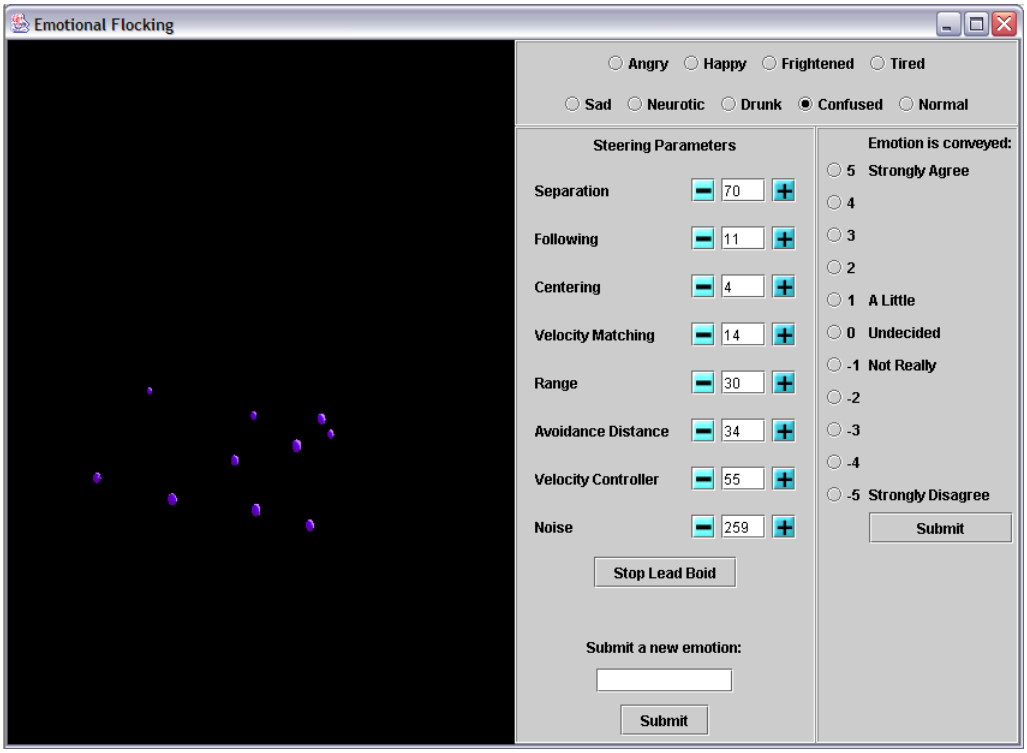

Figure 1: Screen shot of final application

Figure 1 shows the layout of the web interface. It contains three sections: Steering Behavior Interaction, Emotion Presets, and User Ratings.

3.1 Steering Behavior Interaction

Controlling the flock is done by modifying the values of eight steering parameters: separation, following, centering, velocity matching, range, avoidance distance, target velocity, and noise. Three of these parameters correspond to the steering behaviors mention previously. The other five have been added to better control the flock. A short description of each is listed below:

1. Separation - The tendency to move away from other *boids*. This corresponds to the steering behavior *Separation*.

2. Following -The tendency to follow lead *boid*.

3. Centering - The tendency to move toward other *boids*. This corresponds to the steering behavior *Cohesion*.

4. Velocity Matching - The tendency to move in same direction as other *boids*. This corresponds to the steering behavior *Alignment*.

5. Range - Maximum distance between *boids* and neighbors.

6. Avoid Distance - Minimum distance between two *boids* for them to move away from each other.

7. Velocity Controller - Controls the speed of the *boids*.

8. Noise - Vary the path so all *boids* are not heading in exactly the same direction. This makes *boids* move around each other when the lead *boid* is not moving.

The parameters *Separation*, *Following*, *Centering*, and *Velocity Matching* are the main steering behaviors that govern how the flock moves. Each of these behaviors carries a weight of contribution to the change of the position of each *boid*. These four weights always total 100%, so when one is changed, the others are affected.

The parameters *Range* and *Avoid Distance* are helper variables for the *Separation* steering behavior and are both distances.

The parameter *Noise* simulates randomness in the flock to make it more realistic. Without *Noise*, the *boids* would stay in the same relative position to one another when moving around the screen. This value controls the amount of extra noise to add. 0 indicates no noise is added, 200 indicates that twice as much noise as 100. (Note - The slower the *Velocity Controller*, the more *Noise* is needed to vary the path of the *boids*).

There is a *lead boid* not visible to the user that all other *boids* follow using the *Following* steering behavior. The button initially labeled "Stop Lead Boid" allows the user to control the *lead boid*. Pressing this button makes the *lead boid* stop moving and come to the center of the screen. The label will change to "Move Lead Boid." Pressing the button again and the *lead boid* will resume it's previous path of movement.

The 3D space this application is working in:
X coordinates: 0 - 300
Y coordinates: 0 - 300
Z coordinates: -200 - 150

3.2 Emotion Presets

The set of predefined emotions includes: angry, happy, sad, frightened, tired, neurotic, drunk, and confused. The weights for the various parameters will automatically change to those predefined for the selected emotion. The flock's motion will reflect the selected emotion.

3.3 User Rating

The user can rate how well the selected emotion is conveyed. The rating system ranges from -5 to 5, where -5 is *Strongly Disagree* and 5 is *Strongly Agree*. If any of the weights are manually changed, a rating of the modified emotion may be submitted as well.

The combination of these two user inputs(weights and user rating) are used to modify the default weights for a given emotion. The contributions of each steering parameter will be modified via successive averaging of user responses. As more feedback is submitted, the weights will take this feedback into account and make the changes necessary to reflect more realistic behavior. Thus, over time, the motion will better reflect what users believe looks best. For instance, a user of the system may select *Angry*. After some modification of the settings, he or she may find that the new settings results in a better display of anger by the flock. The user can submit an appropriate rating and the default setting for anger will be adjusted when the application is started again.

If the weights are manually changed, new emotion suggestions may be submitted that are best represented by the movement.

4 Software Architecture

4.1 The Flock

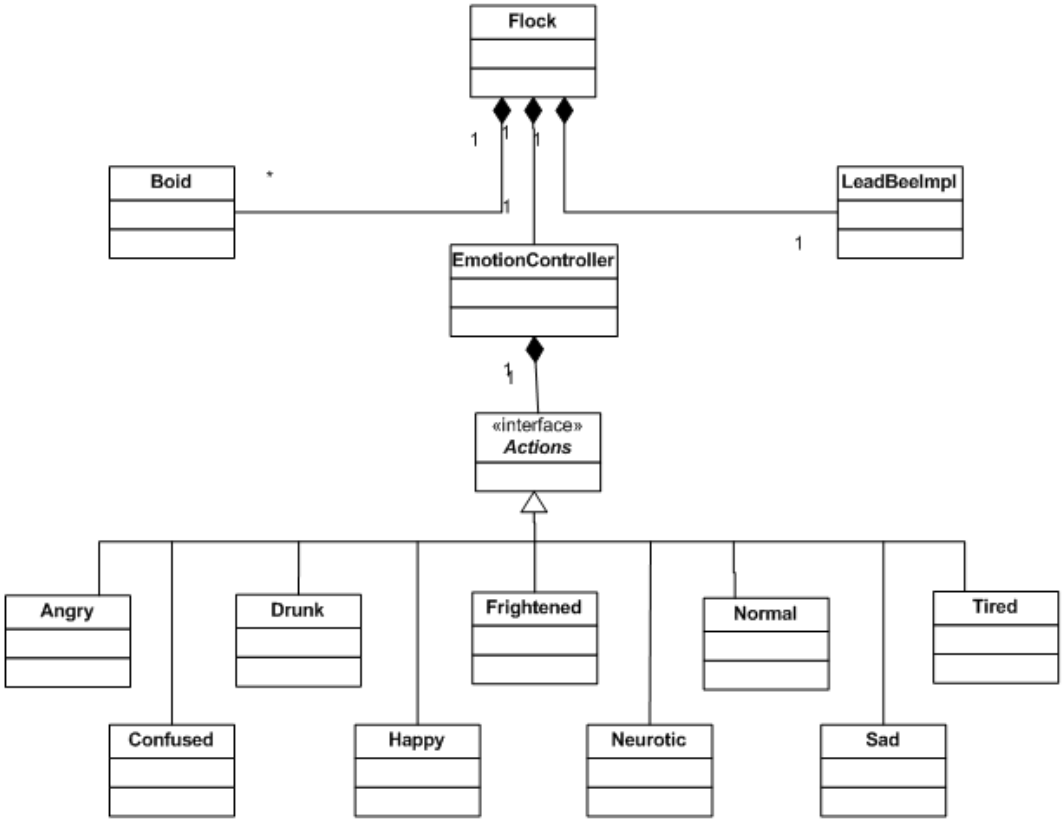

Figure 2: Class Diagram of Flock Engine

Figure 2 is the class diagram of the flock system. Each class is described below in the *Class Responsibility Table*.

Class Responsibility Table:

Flock	The *Flock* object controls the movement of all *boids* in the flock.
LeadBeeImpl	The *lead boid* follows a random path. This path is generated by creating random points around the screen. The position of the *lead boid* is calculated using interpolation.
Boid	One *Boid* object represents one object in the flock. Each *boid* keeps track of its current position, velocity, neighbors, and internal states.
EmotionController	The *Emotion Controller* controls all of the steering parameters.
Actions	The *Actions* object is the superclass for all emotion objects
Angry	This is a subclass of *Actions*. This creates the angry behavior.
Confused	This is a subclass of *Actions*. This creates the confused behavior.
Drunk	This is a subclass of *Actions*. This creates the drunk behavior.
Frightened	This is a subclass of *Actions*. This creates the frightened behavior.
Happy	This is a subclass of *Actions*. This creates the happy behavior.
Neurotic	This is a subclass of *Actions*. This creates the neurotic behavior.
Normal	This is a subclass of *Actions*. This creates the normal behavior.
Sad	This is a subclass of *Actions*. This creates the sad behavior.
Tired	This is a subclass of *Actions*. This creates the angry behavior.

4.1.1 GUI/Flock Interaction

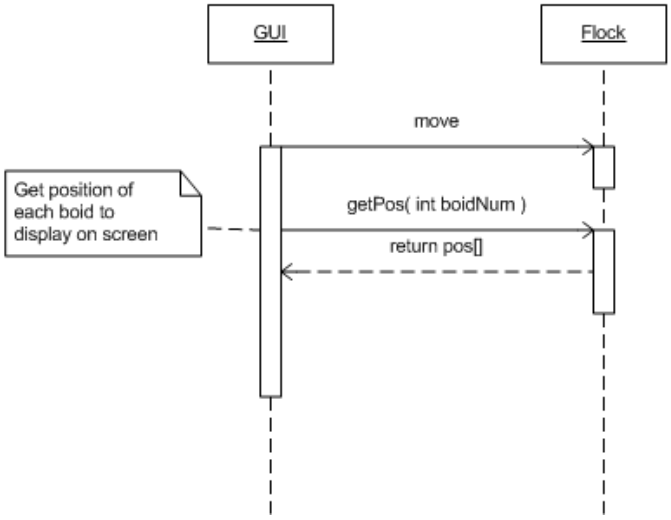

Figure 3: Sequence diagram of *display()* method

Java for OpenGL (JOGL)'s [1] *Animator* class was used to animate the flock. The *Animator* continually calls the *display()* method. This method is the means by which the flock is displayed on the GUI, as shown in Figure 3. The *GUI* instantiates a *Flock* object with the number of *boids* (*size of flock*) desired.

The GUI calls the method *move()* on the *Flock* object. This is where the new position of every *boid* is calculated. The details of these calculations are presented in the next two sections.

Once all *boids* have their new positions, the GUI will display each *boid* as a sphere. This is done by calling the *getPos(int boidNum)* method repeatedly. *BoidNum* is an integer from 0 to the *size of flock*. This returns the position of the given *boid* as a float vector of size three. Position 0 contains the X coordinate, position 1 contains the Y coordinate, and position 2 contains the Z coordinate of the *boid* location.

4.1.2 Move Sequence

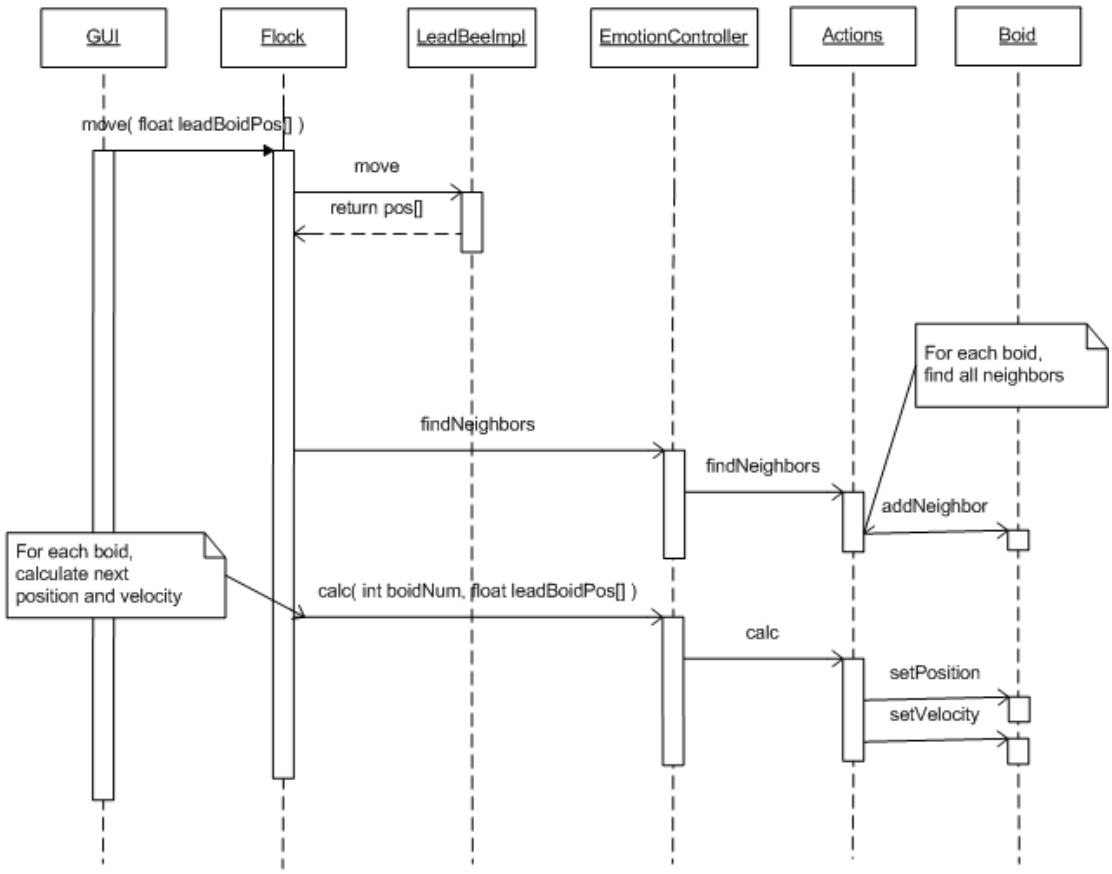

Figure 4: Sequence diagram of *move()* method

Figure 4 is an expansion of the *move()* method in Figure 3. When this method returns, each boid has a new position.

First, the position of the *lead boid* is calculated. The path of the *lead boid* is determined randomly at the start of the application. The path is made up of 20 points, each containing an X, Y, and Z coordinate. The coordinates of each point are randomly picked. A random length of time is also given to each point. This is the time, in milliseconds, it takes the *lead boid* to travel from one point to the next. Given the points and the time, *Cubic*

Interpolation [5] is used to determine the position of the *lead boid* at a given time.

Next, the neighbors of each *boid* is determined. The steering parameter *Range* is used to determine if two *boids* are neighbors of each other. Each *boid* position is compared to every other *boid* position exactly once. If the distance between two *boids* is less than or equal to the *Range*, then each *boid* adds the other to its list of neighbors.

Lastly, the position of each *boid* is calculated. One by one, each *boid* is sent to the *EmotionController*. The *EmotionController* keeps track of which emotion is currently selected. The *boid* is then sent to *Actions* where the positions are calculated. However, *Actions* is a superclass to *Angry*, *Happy*, *Sad*, *Confused*, *Frightened*, *Neurotic*, *Tired*, *Drunk*, and *Normal*. So which one of these nine classes is currently active depends on which emotion is currently selected. [6] defines this as the *State Pattern*. The calculation of the *boid* position is described in the next section.

Instead of calling the method *move()* with no parameters, the method *move(float[] leadBoidPos)* may be called. This alternative is available to give some freedom to the use of this *flock* implementation. The *lead boid* may be controlled from an external source. The movement of the rest of the flock is not affected in any way.

4.1.3 Calc Method

Each emotion uses the same basic methods to calculate the position. To get the different behaviors in the movement of the flock requires slight modifications to these methods. But first, flocking with no emotion is described.

The class *Actions* contains five vectors representing the X, Y, and Z coordinates of the change in velocity: *avoid*, *follow*, *center*, *vmatching*, and *noise*. These direction vectors are used to help calculate the next position of the individual *boids*. Each is cleared at the beginning of the respective methods. The following methods are called by the *calc* method:

avoidBoids(Boid boid)

This method calculates the change in position the *boid* must have to avoid its neighbors. The distance between the *boid* and each neighbor is calculated. If

the distance is less than the steering parameter *Avoid Distance*, the difference between the *boid* and the neighbor is added to the vector *avoid*.

followLeader(Boid boid)

This method calculates the change in position the *boid* must have to follow the *lead boid*. The difference between the *lead boid* and the current *boid* is added to the vector *follow*.

centerBoids(Boid boid)

This method calculates the change in position the *boid* must have to come closer to its neighbors. For each neighbor, the difference between the neighbor minus the *boid* is added to the vector *center*.

matchVelocities(Boid boid)

This method calculates the change in position the *boid* must have to travel in the same direction as its neighbors. The velocity of each neighbor is added to the vector *vmatching*.

noise(Boid boid)

This method adds more movement to the *boids* in the flock. This is not a true noise function, but it gives the appearance of random movement. Without this, the *boids* would tend to stay in the same relative position to one another as the flock moves around the screen. This is achieved by assigning the old velocity to the vector *noise*.

calcPosition(Boid boid, float[] leadBoidPos)

This method takes all the vectors containing the change of positions of the different steering behaviors to calculate the resulting velocity and position of the *boid*.

First, each vector is normalized with respect to direction. Next, each vector is multiplied by the respective steering parameter weight. The result is added

to the current *velocity* vector, which is initialized to zero. For example, each element in the vector *avoid* is multiplied by the *avoid weight* of the steering parameter *Separation*. Each element is then added to the corresponding element of the new *velocity* vector.

The new *velocity* vector is multiplyed by the *Velocity Controller*. It is then sent to the method *checkBoundaries* to make sure the *boid* stays within a bounding box.

Finally, the *velocity* is used to update the current position of the *boid* using Euler Integration [7]. The velocity and position are both saved in the current *boid*.

checkBoundaries(float[] pos, float[] velocity)

This method keeps the *boids* within a certain box area. The velocity is added to the current position of the *boid* and compared to the respective boundary. If the result is outside the boundary, the velocity is negated. The new velocity is returned.

4.1.4 State Pattern

the *State Pattern* is used to implement the different emotions while maintaining the same interface. The *State Pattern* "allows an object to alter its behavior when its internal state changes" [6].

The *EmotionController* is called the *context*. This controls the current *state*, or *Actions*. The *concrete states* are the subclasses of *Actions*: *Angry*, *Happy*, *Sad*, *Confused*, *Frightened*, *Neurotic*, *Tired*, *Drunk*, and *Normal*. The following is a description of how each *concrete state* modifies the implementation of the *state* to alter the behavior of the flock.

Normal

This class changes none of the methods. This shows what normal flocking looks like.

Angry

The emotion *Angry* is achieved by changing the values of the steering parameters. The *boids* move very fast in a clump. Randomly, *boids* fly out of the clump and come back. This class overrides no methods to achieve the different behavior. Instead, this is done by setting the steering parameters *Separation*, *Velocity Controller* and *Noise* fairly high with the *Range* low.

Sad

As with *Angry*, the emotion *Sad* also overrides no methods. The steering parameter *Velocity Controller* is set very low and the steering parameter *Avoid Distance* is set high.

Happy

The emotion *Happy* is achieved by modifying the *checkBoundaries* method. The lower boundary the *boids* may travel is the Y coordinate of the *lead boid*. If the current position plus the new velocity of a *boid* is below the position of the *lead boid*, the Y coordinate of the velocity is made positive. This creates a bouncing affect only when the flock is around the *lead boid*. If the flock is too far away from the *lead boid*, the flock will try to catch up to the leader. When this happens, the flock looks normal.

Frightened

The emotion *Frightened* is achieved by modifying the *followLeader* method. The *boids* only follow the *lead boid* when the leader is a certain distance away. If the leader is within a certain distance, the *boids* do not follow the leader. The steering parameter *Avoid Distance* is set low so the *boids* clump together. The *noise* method is also modified. Each value of the *noise* vector is randomly increased or decreased slightly. This gives the *boids* a little shake to their movement.

Neurotic

The emotion *Neurotic* is achieved by modifying the *noise* method. The values of the *noise* vector are determined randomly. This gives the *boids* a lot of shake.

Confused

The emotion *Confused* is achieved by modifying the *noise* method. The Y value of the *noise* vector is changed. This is done by reducing the value by half. By reducing the Y value, the *boids* move side to side and back to front more prominently. This creates a wondering effect that makes the *boids* look like they don't know which way to go.

Drunk

The emotion *Drunk* is achieved by modifying the *noise* method. The values of the *noise* vector are randomly increased or decreased slightly like in the *Frightened* emotion, except even less. The steering parameter *Velocity Controller* is set very low while *Noise* is set extremely high. The *boids* move around fairly slowly in a random path.

Tired

The emotion *Tired* is achieved by modifying the *avoidBoids* method. The Y value of the *avoid* vector is drastically reduced. This allows the boids to lie on the same plane. The method *fall* is introduced. If the *boid* is above a certain height, the *boid* will descend. Once the floor is reached, the *boid* will follow the *lead boid*. If the *boid* rises to a random height, the *boid* will fall again. The steering parameter *Velocity Controller* is set very low so the *boids* move very slowly.

4.1.5 User Interaction

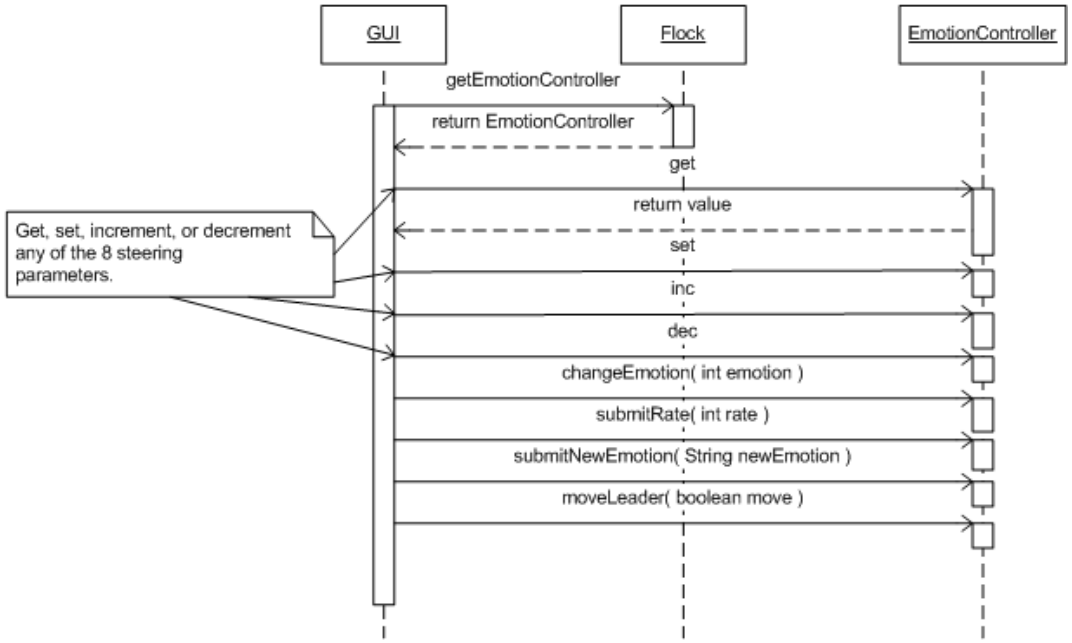

Figure 5: Sequence diagram of user interaction

Figure 5 represents how the user can interact with the system. All interaction is done through the *EmotionController*. All eight steering parameters may be incremented, decremented, or set by the user. The user may change the current emotion and submit a rate for the selected emotion. If the user modified any of the steering parameters, a new emotion name may be submitted. However, althought logged, any new emotion submitted will not appear in subsequent viewing of the application. Lastly, the user is able to start and stop the movement of the *lead boid*. If the *lead boid* is stopped, it moves to the center of the screen. When it is started again, it continues its previous path.

4.2 PHP Server Interface

PHP is a general-purpose scripting language suited for Web Development [3]. In this implementation, PHP is used both an html preprocessor, and a backend to the data storage unit.

1. As a backend to the data storage, PHP is used to retrieve the information given by the user. This information includes different steering parameter values and the ratings given by the individual.

2. As an html preprocessor, it retrieves the information from the database to compute the derived preset values of each emotion. PHP is then responsible for sending this data to the running application.

In this manner, PHP effectively enabled the storage and retrieval of data for a fully functional system without complicated protocols regarding socket connections.

4.2.1 Submission

The preset values for each emotion will be calculated using a weighted average of all previously submitted ratings. All ratings submitted by a user will be between -5 to 5.

All submitted ratings will be saved in a separate file for each emotion. Each file will include one line per user response that contains the rating given plus all of the steering parameters. The user input will come in two different ways:

1. The rating a user gives to the preset values for an emotion, which includes the original values supplied the first time the system is run.

2. The rating and changed values that any user submits as an improvement on the preset values.

Each value will be separated by a":". The order of values on each line of a file will be the following:

1. rate

2. separation
3. following
4. centering
5. velocity matching
6. range
7. avoidanceDist
8. velocityController
9. noise

The ratings are submitted by connecting the the PHP server: *http://www.cs.rit.edu/~kal4249/GradProject/submit.php*. Arguments are attached to this by adding "?*emotion*+*values*". The *values* are added to the end of this in the above order, each separated by a '+'. The *emotion* is the name of the currently selected emotion. This is used to put the values in the correct file. There are nine files, one for each emotion:

1. ANGRY.txt
2. CONFUSED.txt
3. DRUNK.txt
4. HAPPY.txt
5. FRIGHTENED.txt
6. NEUROTIC.txt
7. SAD.txt
8. TIRED.txt
9. NORMAL.txt

A new emotion is submitted through the same PHP server. The arguments attached to this are "?NEW+*emotion*+*values*". The *emotion* and *values* are inserted into the file *NEW.txt*. The *values* are added in the above order, minus the *rate*.

4.2.2 Preset Calculation

The following algorithm will be performed on each file at the start of the program to calculate the preset values of each emotion:

1. All ratings are normalized by dividing the current rate value by the total of all rating values.
2. For each steering parameter:
   (a) For each line of user input, multiply steering parameter by normalized rating.
   (b) Add up all of the scaled steering parameter values over all the lines. This total will equal the preset value for the steering parameter.

For example, if the file ANGRY.txt contains:
5:80:12:4:4:30:12:1.03:2.59
3:80:12:4:4:30:12:1.45:2.59
4:70:14:8:8:30:18:1.42:2.68

Step 1 - the normalized rates will be:
$5/(5+3+4) = .417$
$3/(5+3+4) = .25$
$4/(5+3+4) = .333$

Step 2a - the *separation* parameter * normalized rates:
$80 * .417 = 33.36$
$80 * .25 = 20$
$70 * .333 = 23.1$

Step 2b - the *separation* preset value is 76.46:
$33.36 + 20 + 23.1 = 76.46$

The other seven steering parameter preset values are calculated in the same way. Once calculated, the following will be sent back to the application as the presets for *Angry*:

76.46 12.666 5.334 5.328 30 13.998 1.27 2.62

The current preset values are obtained by connecting to the PHP server: *http://www.cs.rit.edu/∼kal4249/GradProject/CalcPreset.php*. The preset values are returned in one line for each emotion. The order of the values are the same as the submission order. The order the application receives the preset values for each emotion is:

1. NORMAL
2. ANGRY
3. HAPPY
4. SAD
5. FRIGHTENED
6. NEUROTIC
7. TIRED
8. CONFUSED
9. DRUNK

5 Website

The project website can be found at *http://www.cs.rit.edu/∼kal4249/GradProject/Project.php*. Here you can find a link to a couple simple demos of the emotions *Confused* and *Neurotic*.

There is a link to the application itself, which can be found at *http://www.cs.rit.edu/∼kal4249/GradProject/eflocking.php*. Java Webstart is needed to run the application.

Comments and suggestions may be submitted here as well.

6 Results

6.1 Application

The application allows users to submit ratings for each emotion. With each rating, the steering parameter values are also submitted. These values represent either the current preset values for an emotion or the result of changed values by a user.

Each rating a user submits may be a different configuration of steering parameter values. This means that each subsequent user views different preset values. However, the underlying inheritance structure supplied by the different states remains intact. Thus the differences are a means of fine tuning the behavior of the flock.

6.1.1 Parameter Values

The tables *Initial Values* and *Final Values* list the initial and final values of each emotion and the steering parameter values used to create each emotion. The following list is corresponds to the column headings for these tables. The *Final Values* table includes the number of ratings users have submitted for each emotion.

- **S** - Separation
- **F** - Following
- **C** - Centering
- **VM** - Velocity Matching
- **R** - Range
- **AD** - Avoid Distance
- **VC** - Velocity Controller
- **N** - Noise

Initial Values:

	S	F	C	VM	R	AD	VC	N
Angry	50.63	15.19	18.98	15.19	30	12	1.03	2.59
Happy	70.71	11.11	4.04	14.14	30	34	.42	2.59
Sad	60.61	12.12	15.15	12.12	124	113	.04	1.49
Neurotic	65.31	9.18	13.27	12.24	30	13	1.28	2.53
Confused	70.41	11.22	4.8	14.29	30	34	.42	2.59
Drunk	67.68	14.14	11.11	7.07	12	54	.02	14.69
Tired	70.7	11.11	4.04	14.14	30	34	.42	2.59
Frightened	60.6	12.12	15.15	12.12	30	12	.79	.33
Normal	50.63	12.19	18.99	15.19	30	20	1	1

Final Values:

	S	F	C	VM	R	AD	VC	N
Angry	76.4	10.05	12.69	.86	35	12	2.04	2.54
Happy	70.39	11.23	4.08	14.29	25	28	.45	2.34
Sad	56.7	19.28	13.51	10.5	44	40	.01	.66
Neurotic	65.98	8.96	13.04	12.02	31	13	1.28	2.53
Confused	70.68	10.88	5.08	13.93	30	34	.56	2.59
Drunk	68.4	13.98	10.99	6.99	22	55	.01	13.81
Tired	71.68	11.12	4.04	14.16	28	32	.4	2.49
Frightened	61.33	14.16	13.52	10.98	41	12	.75	.32
Normal	55.83	10.84	15.51	17.81	35	23	.61	1.18

The following two graphs represent the initial and final values of the four main steering behaviors (*Separation*, *Following*, *Centering*, and *Velocity Matching*) for each emotion.

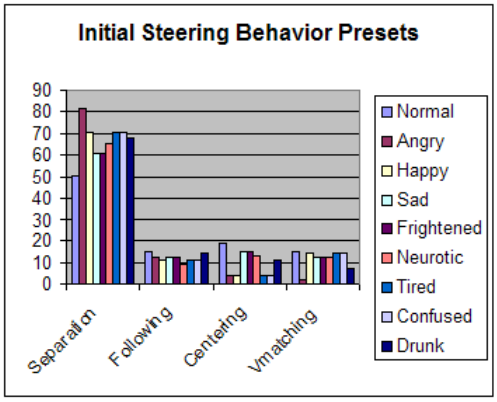

Figure 6: Initial values of Steering Behavior preset weights

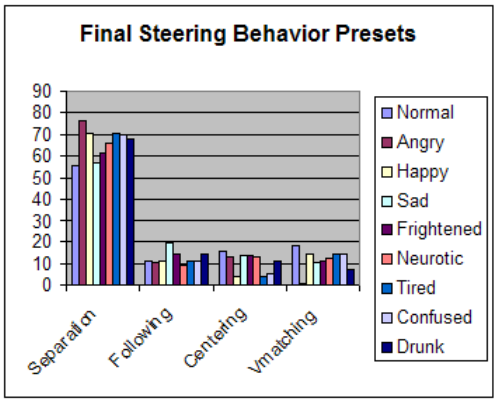

Figure 7: Final values of Steering Behavior preset weights

The following two graphs represent the initial and final values of the four steering parameters (*Range*, *Avoidance Distance*, *Velocity Controller*, and *Noise*) for each emotion.

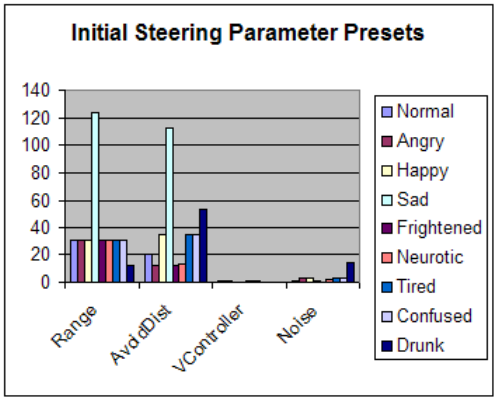

Figure 8: Initial values of Steering Parameter presets

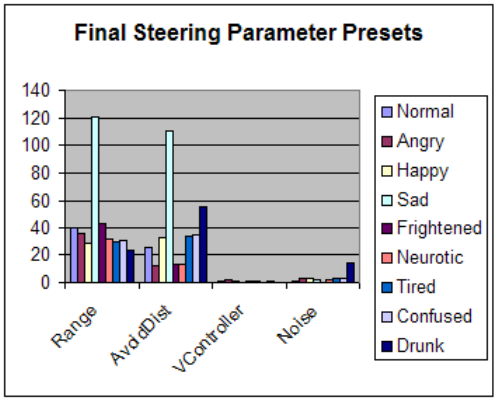

Figure 9: Final values of Steering Parameter presets

To get a clearer understanding of how the values have changed, Figures 10 and 11 show the difference between the initial and final values of the steering parameters for each emotion.

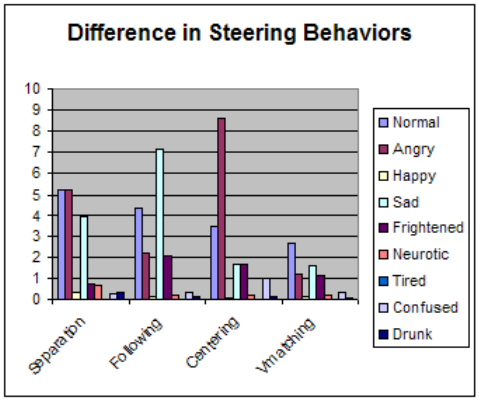

Figure 10: Difference between initial and final Steering Behavior preset weights

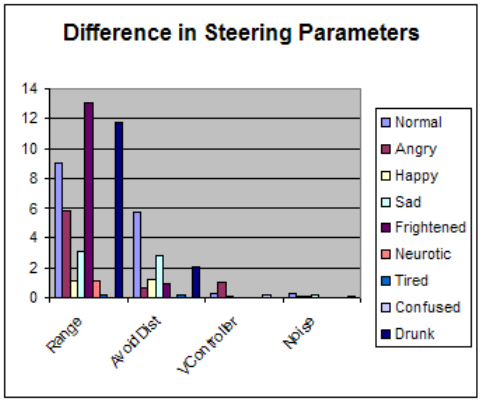

Figure 11: Difference between initial and final Steering Parameter presets

Overall, there is not a lot of change in the values. *Normal*, which represents flocking with no emotion, and *Sad* had the most change over all the steering parameters. These two emotions received some of the lower ratings, so users might have tried to adjust the values to their liking more. *Angry* also has noticable change, but this is due to finding an error in the *matchVelocities()* method after the application was released. I submitted values to change the preset values to represent the behavior that was previously displayed. The only other notable changes occur for *Frightened* and *Drunk*, both with *Range*. I believe *Range* has little affect in this application with these emotions because it is working with a small flock in a relatively small area. *Range* is more meaningful when *boids* can travel outside the range distance so *boids* are not always neighbors of every other *boid*, which I don't believe can happen often with these emotions.

6.1.2 Ratings

Some emotions received better ratings than others. The following table displays the number of ratings received for each emotion and the average rating as of 6/15/04. Ratings range from 5 (Strongly Agree) to -5 (Strongly Disagree). The average rating for each emotion is calculated by the following formula:

$\frac{\sum_{i=1}^{n} r_i}{n}$,
where r is the user rating and n is the total number of user ratings.

	# of Ratings	Average Rating
Angry	19	4.05
Happy	23	2.91
Sad	22	1.14
Neurotic	19	3.58
Confused	23	3.61
Drunk	21	3.19
Tired	20	3.3
Frightened	26	3.5
Normal	16	3.06

6.1.3 Summary of Movement Characteristics

The table *Movement Summary* lists how each emotion uses the four characteristics of movement, described in Section 2.2. Also included is a short description of what kind of behavior that was trying to be conveyed.

Movement Summary:

	Path	**Area**	**Direction**	**Speed**	**Behavior Goal**
Angry	Circling	Large amount of space used	Any direction	Very fast	Angry swarm
Happy	Sine wave	Medium amount of space used	Up and down	Moderate	Bouncing
Sad	Straight	Large amount of space between *boids*	Any direction	Very slow	Lethargic
Neurotic	Very shaky	Moderate amount of space	Any direction	Fast	Crazy
Confused	Straight	Lots of space	Back and forth	Moderate	Wondering lost
Drunk	Wavy	Moderate amount of space	Any direction	Slow-Moderate	Swaggering
Tired	Straight	Little amount of space	Down, sometimes up	Very slow	No energy
Frightened	Straight	Little space between boids	Every direction	Fast, then slow	Huddled together, dodging from place to place
Normal	Straight	Moderate amount of space	Every direction	Moderate-Fast	Flocking without emotion

The different combinations of the four characteristics of movement defines how each emotion is perceived. All four characteristics are necessary to differentiate between emotions. For example, *Angry* and *Neurotic* mainly differ in the *Path*. *Tired* and *Sad* differ in direction.

6.2 Integration With Other Systems

The flocking system should be easy to integrate into other systems. This system is responsible for the calculation of *boid* positions. The visual representation of the *boid* is up to the application using the system.

This application uses simple spheres to represent each *boid*. In a system like *Virtual Theatre*, the *boids* may be represented by complex models developed in Maya. This can be achieved by extending the *Boid* class with each subclass defining it's own visual representation of the boid. In the Virtual Theatre case, where MUPPETS [2] was used as the graphics engine, this was achieved by defining a *boid* as a *MuppetsObject* which is the core class for visual objects in the MUPPETS system.

6.3 Challenges

1. It is hard to convey emotions using only movement. Emotions are a human trait. Expression of emotion is generally communicated through facial expressions, body gestures, and/or speech tones. This system uses only movement relationships between objects. This was a problem which was forseen and infact was the main motivation for this implementation. It was my intent to define a clear separation between emotion in movement vs. emotion in facial expressions or other guestures.

2. This program is highly computationally intensive. As the number of *boids* increases, the computing power necessary to run the system must increase. This is due to the fact that each *boid's* position must be calculated at every timestep based on the neighboring *boid* positions. The more neighbors present, the more calculations there are.

   This application animates a flock containing ten *boids*. It is unclear what the affect of different software suites has on the implementation. For instance, on the same computer, using Mozilla in one case and Internet Explorer as the other case, the performance varies significantly. This may affect a user's perception of the behavior, ergo their ratings may be skewed.

3. I have had difficulty collecting results. To collect ratings, an E-mail was sent to the ACM SIGCSE mailing list, the ACM SIGGRAPH educators

list, and the local ACM SIGGRAPH chapter mailing list. Friends and family were also asked to contribute input.

A week after the E-mails were sent out, a counter was included on the page. After two weeks, there have been 113 hits to the project webpage. The largest number of ratings submitted for any one emotion was 27. This is significantly less than the number of people who viewed the webpage, which does not include the number of viewing in the first week .

I believe the lack of ratings is largely due to the fact that very few people have Java Webstart installed on their computers. If someone does not have what they need to run a program already, the chances of them installing what is needed is low.

If Java Webstart is installed on a person's computer, they still may not run the application because there is a security warning. The application uses Java for OpenGL (JOGL) [1], which is an extension to Java. The only way to run the application is to install JOGL in Java Virutal Machine on the client's computer. It has been found that people feel uneasy about such a security warning, therefore did not run the application.

Also, this application seems to only run on Windows operating systems. I chose to use Java Webstart so the application could be run on multiple platforms. There are native libraries for JOGL available for different platforms that Java Webstart will install, but from my limited testing, this has not worked.

Unfortunately, this leaves a very limited number of people who can run the application. Of the people who can run it, not everyone will rate all of the emotions. The consequence of this is that the data gathered may not be as mature as it would be if more people could view the application.

7 Future Work

7.1 Behavioral Model

As stated previously, three steps are necessary to create a full behavioral model: perception, emotional reaction, and physical reaction. This project only focused on the physical reaction. The other two steps must be implemented to create a behavioral model that can interact with a virtual environment on its own.

For example, if the behavioral model was created around the *lead boid*, the *lead boid* would be made to travel around the virtual world without a predefined path. The *lead boid* would be able to sense obstacles it it's path and avoid them. The leader could also have its own emotional state that reacts to the environment. This emotional state would be passed onto the rest of the flock, which conveys the emotion.

7.2 Boid Model

This project uses plain spheres to represent each *boid*. Instead of spheres, a model of a bee or bird may be used. In doing this, it would extend the amount of animation that could be done to the flock to convey an emotion. Currently, only the relationship between the *boid's* positions created by movement is used to convey emotion. A model more complex than a sphere will be able to help convey an emotion by animating the model to express an emotion.

This would greatly improve an emotion like *Sad*. *Sad* is a hard emotion to simulate using only movement. With a bee model representing each *boid*, *sad* can be better expressed by hanging the bee's head, antenna, and stinger lower than they normally are positioned. This, in combination with the given movement, better conveys the emotion.

7.3 Genetic Algorithm

A rating system is used to collect user feedback of how they feel an emotion is portrayed. This input is then used to modify the presets which control the flock's movement. A different way to gather this data would be to use a genetic algorithm to find emotions.

The population of the genetic algorithm will be the input from each of the users. As more and more users utilize the system, we will see more and more chromosomes added to the population. The new members of the population give new variances and traits to the emotions. Running the genetic algorithm on this updated population should yield a convergence to an optimal solution, where an optimal solution is an accurate representation of an emotion.

7.4 Additional Emotions

This project animates only a few emotions. Many more, such as *Surprise*, *Wonder*, or *Amusement*, could be added to expand the range of emotions and behaviors that are displayed. The only limitation is the animator's creativity.

7.5 Platforms

This project uses JOGL to display the animation. JOGL is an extension to java, which most computers do not have. In order for this application to be run, JOGL must be installed on the client computer. Java Webstart enables this to happen without too much difficulty. This means that the client computer must have Java Webstart in order to start the application, but not all do. So far, only computers running Windows are able to launch this. A way to run this application in any environment is needed.

8 Conclusion

I believe that this project has successfully shown that emotion can be conveyed through movement in flocking. A great majority of ratings that were received agreed that the emotion was expressed.

It is disappointing that the number of ratings received was not higher. A better way to distribute the application needs to be found that enables the program to be run on multiple platforms and without security warnings.

The flock shown in this project best represents a swarm of bees. I believe it may also represent other insects or a school of fish. However, different types of flocks may require different behaviors to convey the emotions represented

here. For example, an angry flock of birds might not fly around in a fast clump as shown in this application. Instead, birds might jab more at the center of the flock to represent massive pecking.

This project is a proof of concept that some emotions may be conveyed using only movement. However, some emotions are conveyed more effectively than others. *Angry* and *Neurotic* were highly accepted as conveying the emotions. *Sad* was least accepted. Creating a model to replace the simple sphere would help this greatly. Then, in addition to the movement, body language or facial expressions can be used to enhance the emotion. Not a lot is needed, especially if the main focus of conveying an emotion is the movement. A smile or a drooped head, while very simple, could greatly improve the expression of an emotion. These findings support the idea that more features than just movement may be required to convey an emotion. Using only movement can accurately express emotion in some cases, however in other cases, this limits the extent of realism in an animated flock. Other characteristics, such as facial expressions or shape and color altering, should enhance the desired effect.

References

[1] Jogl: Java for opengl.

[2] Muppets: Multi-user programming pedagogy for enhancing traditional study. `http://muppets.rit.edu`.

[3] Php: Hypertext preprocessor. `http://www.php.net`.

[4] P. Becheiraz and D. Thalmann. A behavioral animation system for autonomous actors personified by emotions. In *Proceedings of the 1st Workshop on Embodied Conversational Characters*, pages 57–65, 1998.

[5] R. Dunlop. Introduction to catmull-rom splines. `http://www.mvps.org/directx/articles/catmull/`.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison Wesley, 1995.

[7] R. Parent. *Computer Animation Algorithms and Techniques*. Morgan Kaufmann Publishers, 2002.

[8] C. W. Reynolds. Steering behaviors for autonomous characters. In *Proceedings of Game Developers Conference*, pages 763–782, 1999.

[9] X. Tu and D. Terzopoulos. Artificial fishes: physics, locomotion, perception, behavior. In *Proceedings SIGGRAPH*, pages 43–50, 1994.

[10] L. C. Vaughan. Understanding movement. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 548–549. ACM Press, 1997.