유한상태기계를 사용한 비둘기들에 대한 집단행동의 설계 및 구현※

이재문*, 조세홍*
한성대학교 멀티미디어공학과*
{jmlee, chosh}@hansung.ac.kr

Design and Implementation of Group Behaviors for Doves by Using a Finite State Machine

Jae Moon Lee*, Sae Hong Cho*
Dept. of Multimedia Engineering, Hansung University*

요 약

본 논문은 비둘기들의 다양한 상태에 대하여 집단행동을 자연스럽게 시뮬레이션하는 시스템을 설계하고 구현하는 것이다. 이것을 하기 위하여 비둘기들의 집단행동은 '날아가기', '내려앉기', '먹이먹기' 및 '날아오르기'와 같이 4개의 액션모델로 나뉘었다. 각 액션모델을 구성하는 조종힘들이 찾아졌으며, 유한상태기계 기법을 사용하여 설계되었다. 설계된 시스템은 오우거 엔진과 집적하여 구현되었다. 구현된 시스템의 시뮬레이션으로부터 비둘기들의 자연스러운 집단행동을 표현하는 조종힘에 대한 다양한 파라미터 값들을 찾을 수 있었다.

ABSTRACT

This paper is to design and implement the system to simulate spontaneously the group behaviors for the various states of doves. To do this, the group behaviors of doves were divided into the four action models such as 'Flying', 'Landing', 'Eating' and 'Taking off'. The steering forces composing of each action model were found and each action model was designed by using the finite state machine. The designed system was implemented by integrating the Ogre engine. From the simulations of the implemented system, the values of the parameters for the steering forces were found so that it can represent the spontaneous group behaviors of doves.

Keywords : Group Behaviors, Finite State Machine, Steering Behaviors(집단행동, 유한상태기계, 조종행동)

접수일자 : 2010년 05월 10일 심사완료 : 2010년 06월 08일
교신저자(Corresponding Author) : 이재문
※ 본 연구는 2010년 한성대학교 교내연구비 지원과제임.

1. 서 론

집단행동은 다수의 에이전트들이 자율적으로 움직일 때 나타나는 행동을 말한다[1,2,3,5.9]. 이러한 집단행동은 영화나 게임 등에서 장면의 사실성을 증대시키거나 현실적으로 제작하기 어려운 장면을 시뮬레이션 하는데 사용된다. 집단행동의 가장 단순한 형태가 파티클 시스템이다[1,5,8]. 파티클 시스템은 영화나 게임에서 불, 비, 폭발 등의 효과를 나타내는데 사용된다. 파티클 시스템보다 진화한 집단행동이 무리 짓기이다[1,2,3,4,8,9]. 이는 무리 짓기에 속한 에이전트들이 상호 작용을 하면서 집단행동을 하기 때문에 파티클 시스템에 비하여 진화한 것이라 할 수 있다.

집단행동에 대한 대부분의 기존 연구는 자연 현상에서 일어나는 무리에 대하여 하나의 상태에 대한 자연스러운 행동에만 초점이 맞추어져 왔다[1,2,3,4]. 이러한 관계로 집단행동에서의 연구는 액션 모델에 대한 연구보다는 하나의 상태에 대한 조종층 연구에 치중되어 왔다. 본 논문에서는 이를 확장하여 무리들이 다양한 상태를 가질 수 있도록 상태별 집단행동을 자연스럽게 시뮬레이션 하는 것이다. 이를 위하여 본 논문에서는 무리를 비둘기 무리로 선정하였고, 비둘기 무리의 상태를 '날아기기', '내려앉기', '날아오르기' 및 '먹이먹기'로 정의하여 각 상태별로 자연스러운 집단행동을 나타날 수 있는 조종행동을 설계하였다. 비둘기 무리의 상태를 관리하기 위하여 유한상태기계 기술을 도입하였고, 하나의 상태에 대하여 다수의 단독 또는 집단 조종행동을 설계하였다. 설계된 시스템은 Ogre 그래픽 엔진[10]을 이용하여 구현되었다.

2장에서는 기존의 유한상태기계와 그룹행동에서 나타나는 다양한 조종행동에 대하여 소개한다. 3장에서는 비둘기 집단행동에 대한 설계 내용을 설명하며, 4장에서는 설계된 내용의 구현에 대하여 설명한다. 5장에서는 결론과 향후 연구에 대하여 소개한다.

2. 기존연구

2.1 유한상태기계

유한상태기계(FSM: Finite State Machine)는 게임에서 가장 많이 사용하는 인공지능 기술 중의 하나이다[4,8]. 유한상태기계는 임의의 주어진 시간 범위 내에 있을 수 있는 유한개의 상태를 가지고 주어지는 입력에 따라 어떤 상태에서 다른 상태로 전환시키거나 출력이나 액션이 일어나게 하는 장치 또는 그 모델을 의미한다. 유한상태기계는 인공지능 프로그래머들이 게임 에이전트에게 지능을 부여하기 위한 선택 도구로 사용되어왔다.

본 연구에서 유한상태기계는 자동 에이전트가 가질 수 있는 행동의 범위를 제약하는 것에 사용된다. 즉, 아무리 자동적으로 움직이는 에이전트일지라도 시스템은 제한된 범위에서 에이전트가 자동으로 행동하도록 제어하고자 할 것이다. 예를 들어 축구 게임을 고려해 보자. 축구 게임에서 움직이는 개체는 선수들이고, 각 선수는 공격, 수비 등이 상태로 나타난다. 즉, 선수들은 팀의 상태에 따라 공격형 집단행동과, 수비형 집단행동으로 구분될 수 있다. 따라서 시스템은 팀의 상태에 따라 선수들이 그 상태에 적합한 자동행동을 하도록 제어하고자 할 것이다. 본 논문에서는 비둘기들의 집단행동에 대하여 유한상태기계 기법을 적용한다.

2.2 에이전트의 자동행동

[1,2]에서는 에이전트의 자동행동을 크게 액션모델 선택, 조종층 및 이동층으로 나누었다. 액션모델 선택은 에이전트가 특정한 목표를 정하고, 그에 따른 행위를 결정하는 것이고, 조종층은 선택된 목표를 성취하기 위하여 정해진 계획에 따른 궤적을 계산하는 것이다. 조종층은 이러한 궤적에 따라 움직일 수 있도록 어디로 가야하고 얼마나 빠르게 움직여야 하는 지를 나타내는 조종력을 계산한다. 이 조종력의 계산은 장애물과 같은 주변 환경적인 요인뿐만 아니라 다른 에이전트들의 영향도 고려하

여야 한다. 이동층은 조종층에 의하여 계산된 조종력을 사용하여 해당 에이전트의 위치, 방향 및 속도 등을 새롭게 계산하는 층이다. 세 가지 층에서 자동적 움직임을 위하여 가장 많은 계산을 필요로 하는 층이 조종층이다. 조종층에서의 조종력 계산은 응용 환경에 따라 다음과 같이 단독 조종행동과 집단 조종행동으로 나뉜다[1,2,3,4].

● **단독 조종행동**

단독 조종행동이라 에이전트가 움직이기 위하여 조종힘을 계산하게 되는데 다른 에이전트의 영향 없이 해당 에이전트 단독으로 조종힘을 계산하는 경우이다. [2]에서는 이러한 단독 조종행동을 '찾아가기', '도망가기', '추격하기', '도피하기', '배회하기', '도착하기' 및 '장애물 피하기'등을 제안하였으며, [4]에서는 이들외에 '벽피하기', '끼워넣기', '숨기기', '경로따라하기', '오프셋추격하기' 등을 추가하였다. 다음은 [2]에서 제시한 단독 조종행동에 대한 간단한 설명이다.

- 찾아가기 및 도망가기 : 찾아가기는 목표물을 향하여 에이전트를 이끄는 힘을 생성하는 조종 행동이다. 이 조종행동의 힘은 목표물에 도달하기 위한 속도와 현재의 속도 차이를 힘으로 나타낸다. 도망가기는 정확히 찾아가기와 반대의 경우이다.
- 배회하기 : 배회하기는 아무런 목표없이 이리저리 돌아다는 조종행동이다. 아무런 목표가 없다고 해서 무작위로 조종힘을 생성하는 경우 비현실적인 조종행동을 초래한다. 본 연구에서는 [4]에서와 같이 에이전트 앞에 하나의 원을 투사하고 경계선을 따라 이동하도록 제한된 목표 방향으로 조종해 나가는 방식을 사용하였다.
- 도착하기 : 이 조종행동은 찾아가기와 매우 유사하다. 그러나 찾아가기가 목표물을 향하여 거침없이 이동하기 때문에 대부분의 경우 목표물을 지나치게 된다. 도착하기는 이렇게 목표물을 지나치지 않게 하기 위하여 목표물과 가까워질수록 속도를 줄임으로서 부드럽게 목표물에 안착하는 느낌을 주는 조종행동이다.

● **집단 조종행동**

집단 조종행동이란 에이전트가 새로운 조종힘을 계산할 때 주변에 존재하는 이웃 에이전트의 영향을 고려하여 조종행동을 말한다. 무리 짓기의 행동이 대표적인 집단 조종행동의 조합이다. [1,2,3,4]에서는 집단 조종행동을 분리, 정렬 및 결합 행동으로 정의하였다.

- 분리하기 : 에이전트를 이웃지역 내의 에이전트들로부터 멀어지게 하는 조종행동이다. 에이전트들의 무게중심으로부터 멀어지게 하는 힘으로 이웃들 간의 거리를 너무 가까워지지 않게 한다.
- 정렬하기 : 에이전트의 방향을 자신의 이웃 에이전트들이 향하는 방향과 같게 하는 조종행동이다. 방향을 같게 하는 힘이지만 상황에 따라 보다 큰 무리가 작은 무리에게 미치는 영향력을 계산하는데 사용하기도 한다. 정렬은 이웃 에이전트들의 방향의 평균을 자신의 방향으로 적용하는 방식을 따르기 때문이다.
- 결합하기 : 이웃 에이전트들의 무게중심을 향하여 움직이게 하는 조종행동이다. 분리와 정렬로 인해 무리에서 너무 멀어지면 가깝게 모여 무리를 짓게 하는 힘이다.

3. 비둘기 집단행동의 설계

3.1 비둘기들에 대한 액션모델의 설계

본 절에서는 자연에서 생활하는 비둘기들에 대한 액션모델의 설계에 대하여 설명한다. 자연에서 비둘기 떼들을 살펴보면 통상적으로 비둘기들은 날아다닌다. 그러한 상태에서 피곤하거나 허기를 느끼는 경우 지면위에 내려앉을 것이다. 내려앉은 상태에서 휴식을 취하거나 먹이를 찾는다. 먹이를 찾은 경우 먹이를 먹고, 충분한 먹이를 먹은 후 비둘

기들은 또다시 날아다닐 것이다. 본 논문에서는 비둘기 떼들의 이러한 상황에 대하여 '날아가기', '내려앉기', '먹이먹기' 및 '날아오르기'로 액션모델을 정의한다.

2장에서 언급하였듯이 하나의 액션모델은 여러 단독 또는 집단 조종힘의 조합으로 나타난다. 또한 하나의 액션 모델에 포함된 여러 조종힘은 우선순위가 있어야 하며, 가중치가 적용되어야 한다. 본 논문에서는 자연 상태에서 발견되는 다양한 비둘기의 동작을 기준으로 다양한 실험을 통하여 각 액션모델에 내포되어야 하는 조종힘을 다음과 같이 설계하였다.

- 날아가기(*Flying*) : 비둘기 상태들 중 가장 기본적인 액션모델이다. 이 모델은 비둘기들이 무리를 지어 날아가기 때문에 [2]에서 제시하는 일반적인 무리 짓기가 적용되는 상태이다. '분리하기'를 사용하여 이웃 간의 거리를 충분히 넓히고 '정렬하기'를 사용하여 서로 같은 방향으로 움직이도록 한다. 무리로부터 너무 멀리 떨어지는 것을 막기 위하여 '결합하기'도 적용된다. 또한 단순한 형태로 날아다니는 것을 막기 위하여 '배회하기'도 적용된다.
- 내려앉기(*Landing*) : 비둘기가 허기를 느끼게 되면 지면상의 임의의 지점이나 먹이가 있는 지점으로 내려앉게 된다. 먹이가 있는 지점을 나타내기 위하여 게임 공간의 지면 중 특정 영역을 설정하고 비둘기 떼들이 이 영역으로 내려앉기를 하도록 설계하였고, 각 비둘기에 대해서는 이 영역중 하나의 지점을 할당하여 이 지점으로 내려앉도록 설계하였다. 이 영역은 비둘기들이 무리를 지어 날아가는 공간보다는 비교적 좁도록 하였고, 비둘기들이 일시에 이 영역으로 모이게 되므로 비둘기간 충돌이 일어나지 않아야 한다. 따라서 '분리하기'가 무엇보다 중요하게 적용되어야 한다. 내려앉기 상태에서 비둘기들은 먹이가 보이는 곳으로 바로 날아가기를 원하기 때문에 '정렬하기'와 '결합하기' 보다는 먹이를 향하여 '도착하기' 행동이 크게 작용할 것이다. 따라서 이 액션모델에 대한 조종힘을 '분리하기'와 '도착하기' 힘의 합으로 설계하였다.
- 날아오르기(*TakingOff*) : 비둘기가 피로나 허기가 느껴지지 않으면 비둘기는 다시 날아오르는 것으로 설계하였다. 이것을 시뮬레이션하기 위하여 내려앉기와 유사하게 공중의 특정 영역을 설정하고, 각 비둘기에 대해서는 이 영역중 하나의 지점으로 날아오르도록 설계하였다. 이러한 관점에서 날아오르기는 내려앉기와 정확히 반대 액션모델이지만 그들의 구성 힘은 동일하다. 따라서 날아오르기에 대한 힘도 '분리하기'와 '도착하기' 힘의 합으로 설계하였다.
- 먹이먹기(*Eating*) : 비둘기들은 '내려앉기'를 완료한 후부터 '먹이먹기'로 상태 전이를 한다. 이 상태는 항상 지면에서 이루어지는 것으로 비둘기는 가장 먼저 배회하면서 먹이를 찾아 먹는다. 다음으로 찾아진 먹이에 대하여 '도착하기'를 실행한다. 이러한 와중에도 다른 비둘기와 충돌을 피해야 하므로 '분리하기'가 우선적으로 적용되도록 설계하였다.

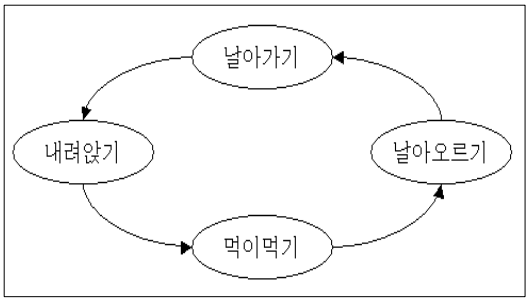
[표 1]은 앞에서 설계한 액션모델에 대하여 적용되는 힘들을 정리한 것이다. C_{sep}는 분리힘에 대한 가중치이며, C_{ali}는 정렬힘, C_{coh}는 결합힘, C_{arr}는 도착힘 및 C_{wan}은 배회힘에 대한 각각의 가중치를 표시하며, 각 힘들은 표시 순서대로 우선순위가 적용된다([그림 7]참조).

[표 1] 비둘기 무리 짓기의 액션 모델

액션모델	적용된 조종힘
Flying	$f_f = C_{sep}$*분리힘+C_{ali}*정렬힘+ C_{coh}*결합힘+C_{wan}*배회힘
Landing	$f_d = C_{sep}$*분리힘+C_{arr}*도착힘
Taking Off	$f_u = C_{sep}$*분리힘+C_{arr}*도착힘
Eating	$f_s = C_{sep}$*분리힘+C_{arr}*도착힘+ C_{wan}*배회힘

3.2 비둘기들에 대한 유한상태기계 설계

본 논문에서는 자연에서 무리를 지어서 날아다니는 비둘기들의 집단행동에 대한 액션모델을 '날아가기', '내려앉기', '날아오르기', '먹이찾기' 및 '먹이먹기'로 간략화 하였다. 이러한 액션모델들은 임의적으로 발생되는 것이 아니라, 어떤 이벤트에 따라 순서적으로 일어나야한다. 이러한 이벤트에 따른 상황변화를 가장 잘 제어할 수 있는 기술이 유한상태기계 기법이다. 본 논문에서는 앞서 정의한 액션모델을 제어하기 위한 기법으로 유한상태기계를 도입하였고, 각 액션모델을 유한상태기계에서 하나의 상태로 대응하여 설계하였다.

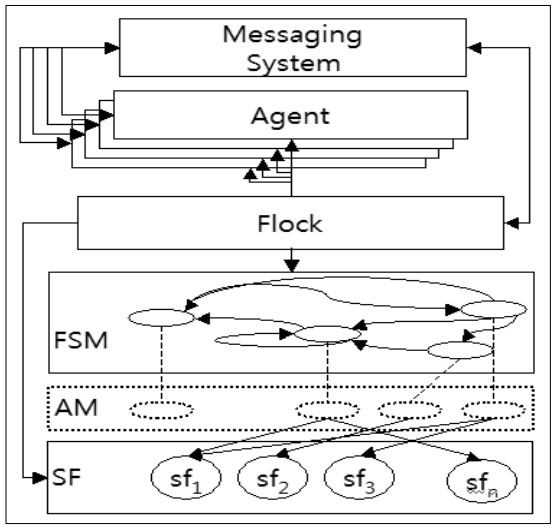


[그림 1] 액션모델에 대한 상태 다이어그램

[그림 1]은 비둘기들의 액션모델에 대한 상태 다이어그램이다. 간단히 표시하기 위하여 상태 전이를 위한 이벤트는 생략하였다. 비둘기들은 통상적으로 '날아가기'를 한다. 그러한 상태에서 피곤하거나 허기를 느끼는 경우 지면위에 '내려앉기'를 할 것이다. 내려앉은 상태에서 휴식을 취하거나 '먹이찾기'를 한다. 먹이를 찾은 경우 '먹이먹기'를 할 것이다. 충분한 휴식이나 먹이를 먹은 후 비둘기는 '날아오르기'를 하고 또다시 '날아가기'를 할 것이다. 유한상태기계는 매우 유연한 확장성을 제공한다. 예를 들어 만약에 비둘기들을 쫓아다니는 '매'가 있고, 날아가는 순간에 '매'가 달려들면 도망을 가야하므로 '도망가기' 액션모델이 필요하고, 이 경우 [그림 1]에서 '도망가기' 상태를 추가하면 된다.

3.3 비둘기 집단행동에 대한 시스템 설계

본 논문에서는 하나의 비둘기 무리만 시뮬레이션하는 시스템을 설계하고 구현하였다. 즉, 다수의 무리가 존재하여 무리간 상호작용, 이합집산 행동은 배제하였다. 설계된 시스템은 비둘기 각각을 관리하는 *Agent* 모듈, 전체 비둘기 무리를 관리하는 *Flock* 모듈, 무리의 액션모델을 관리하는 *AM*과 *FSM* 모듈, 조종힘을 관리하는 *SF* 모듈 및 에이전트간 또는 무리와 에이전트간 통신을 위한 *Messaging System* 모듈로 구성되었다.



[그림 2] 개발 시스템의 모듈 및 모듈간의 관계도

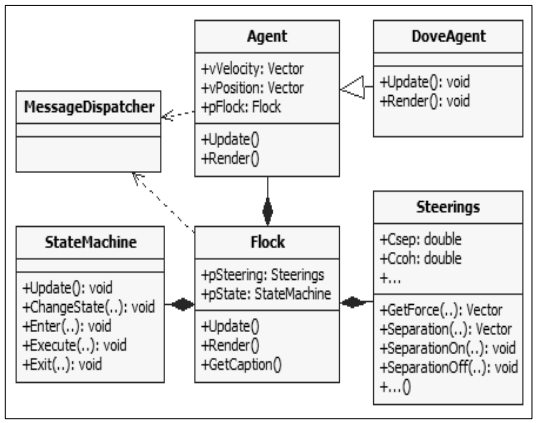
[그림 2]는 본 논문에서 개발하는 시스템의 모듈 및 모듈간의 관계도이다. *Agent* 객체는 무리에 속한 다수의 비둘기에 해당하므로 다수 존재하며, *Flock* 객체, *FSM* 객체, 및 *SF* 객체는 오직 한 개만 존재한다. 이것은 앞서 언급하였듯이 단지 하나의 무리만 시스템에 존재하기 때문이다. *AM* 모듈은 실제 설계 및 구현에서는 *FSM* 모듈에 포함시켰다. *AM* 모듈의 각 구성요소와 *FSM*모듈의 구성요소가 1:1로 대응하기 때문에 두 모듈을 하나의 모듈로 구성함으로써 시스템을 간략화할 수 있었다.

4. 비둘기 집단행동의 구현 및 시뮬레이션

4.1 시스템의 주요 클래스 구현

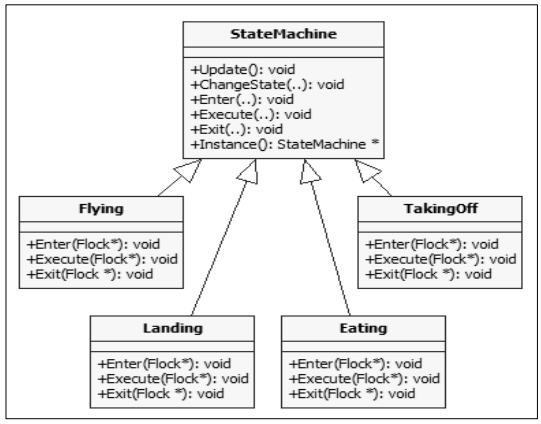
본 논문에서 설계한 시스템을 윈도우즈상에서 Visual Studio 2005, C++ 언어 및 STL(standard template library)을 사용하여 개발하였다. 또한 빠른 개발을 위하여 공개 3D 그래픽 엔진인 Ogre(OGRE:Object-oriented Graphic Rendering Engine)[10]을 사용하여 개발하였다.

[그림 3]은 [그림 2]의 모듈들에 대하여 구현된 클래스 관계도이다. 하나의 *Flock* 객체는 여러 개의 *Agent* 객체를 가지고 있으며, 한 개씩의 *StateMachine*(FSM) 객체와 *Steerings*(SF)객체를 가지고 있다. *StateMachine* 객체가 오직 *Flock* 객체에 대한 유한상태기계이므로 *Agent* 객체는 *Flock* 객체의 상태에 지배를 받게 된다. 즉, 이것은 비둘기들의 무리에서 '한 마리의 비둘기는 무리의 상태에 지배를 받는 것이지 별도의 상태를 유지하는 것은 아니다'라는 것을 의미한다. 또한 비둘기 객체인 *DoveAgent*는 *Agent* 객체로부터 상속받도록 하였다. *MessageDispatcher*(Messaging System)는 에이전트간 또는 에이전트와 *Flock*간 통신이 필요한 경우 1:1 또는 1:N 통신을 지원한다.



[그림 3] 개발 시스템에 대한 클래스 관계도

시스템은 매 순간마다 Ogre의 장면 그래프에 대하여 *Update*와 *Render*를 호출하며, 이 함수들은 각각 *Flock::Update*와 *Flock::Render*을 호출하면서 동작한다. *Flock::Update*에서는 *StateMachine::Update*와 각 *DoveAgent*에 대하여 *Agent::Update*를 호출하게 되고, *Agent::Update*에서는 *Flock*을 통하여 *Steerings::GetForce*를 호출하여 자신의 새로운 위치와 방향을 정하게 된다. *StateMachine::Update*에서는 각 상태에 대하여 *StateMachine::Execute*를 호출하게 되고, 여기서 각 상태별로 상태 전이에 대한 조건을 체크한다.



[그림 4] FSM 모듈에 대한 클래스 설계

4.2 유한상태기계의 구현

[그림 4]는 [그림 1]의 상태 다이어그램에 대한 클래스 설계 다이어그램이다. *StateMachine*이라는 상위 클래스를 구현하고 이를 상속받아 4개의 상태를 클래스로 구현하였다. *StateMachine::Update*는 *Flock::Update*로 부터 매 프레임마다 호출되는 함수이다. *StateMachine::Enter*은 새로운 상태로 전이하는 경우에 처음 한번만 호출되며, *StateMachine::Exit*는 현재의 상태를 벗어나기 직전에 한번 호출되는 함수이다. *StateMachine::Execute*는 *StateMachine::Update*에서 호출되는 함수이다. 따라서 이 함수를 통하여 이 상태에서 매 프레임마다 실행되어야 하는 일을 실행한다. [그림 4]의 주요 구현 기법은 [4]를 참고하였다.

```
Flying::Enter(Flock *flock){
  // '날아가기' 상수 설정
  flock-->pSteering-> C_sep = 4.0;
  flock-->pSteering->S C_ali = 8.0;
  flock-->pSteering-> C_coh = 1.0;
  flock-->pSteering-> C_wan = 1.0;
  // '날아가기' 조종힘 계산함수 설정
  flock-->pSteering->SeparationOn();
  flock-->pSteering->AlignmentOn();
  flock-->pSteering->CohesionOn();
  flock-->pSteering->WanderOn();
}
Flying::Execute(Flock *flock){
  if(flock->GetCaptain()->FeelHungry()){
    flock->pState->
        ChangeState(flock, Landing::Instance());
  }
}
Flying::Exit(Agent *agent){
  flock->pSteering->SeparationOff();
  flock->pSteering->AlignmentOff();
  flock->pSteering->CohesionOff();
  flock->pSteering->WanderOff();
}
```

[그림 5] '날아가기' 상태 클래스의 구현

[그림 5]는 '날아가기' 상태에 대한 *Flying::Enter*와 *Flying::Execute*에 대한 의사코드이다. *Flying::Enter* 함수에서는 이 상태에 들어가기 전에 필요한 조종행동과 이에 대한 상수를 설정하고, *Flying::Execute*에서는 지속적으로 대장(Captain) 비둘기의 피로도를 측정하여 필요한 경우 상태 전이를 한다. *Flying::Exit*에서는 다음 상태를 위하여 현재 상태에서 사용하였던 모든 조종행동을 제거한다. '내려앉기', '날아오르기' 및 '먹이먹기' 상태도 [그림 5]와 유사하게 구현되었다.

4.3 조종힘의 계산

조종힘 계산은 매 프레임마다 각 비둘기에 대하여 계산되어야 하는 힘이다. *Agent::Update* 함수는 매 프레임마다 *Flock::Update*로부터 호출되어 비둘기의 조종힘을 계산하여 새로운 위치와 방향을 계산한다. [그림 6]은 *Agent::Update* 함수의 의사코드를 보이고 있다.

```
Agent::Update(AgentList neighbor, float time){
  // 조종힘 계산
  vForce=pFlock->pSteering->
                        GetForce(this, neighbor);

  // f=ma에서 가속도(a)를 구함
  vAccel = vForce / GetMass();
  vVelocity = GetVelocity();

  // v= v_o+at, p=p_o+vt 계산
  vVelocity += vAccel * time;
  vPosition += vVelocity * time;

  // 속도, 위치와 방향의 재갱신
  SetVelocity(vVelocity);
  SetPosition(vPosition);
  SetOrientation(vVelocity.normalise());
}
```

[그림 6] 조종힘을 이용한 위치와 방향의 갱신

가장 먼저 *Steerings::GetForce*를 호출하여 에이전트에 미치는 힘을 계산하고, 그 힘을 이용하여 가속도($a=f/m$)를 계산한다. 이 가속도를 이용하여 속도($v=v_o+at$)와 위치($p=p_o+vt$)를 계산한다. 이렇게 계산된 속도와 위치는 다음 프레임을 위하여 저장될 뿐만 아니라 이 에이전트의 *Ogre SceneNode*에 새로운 위치와 방향 벡터를 갱신하여야 한다. 이것은 시스템 렌더링이 Ogre에 의하여 실행되기 때문이다.

함수 *Steerings::GetForce*는 에이전트의 조종힘을 계산하는 함수이다. 보통 하나의 에이전트에 하나의 조종힘만 적용되는 경우는 없다. 예를 들어 '날아가기'의 경우 분리힘, 정렬힘, 결합힘 및 배회힘을 적용하되 적절히 가중치를 적용하여 원하는 '날아가기'를 시뮬레이션 한다. 이때 최종적인 조종힘을 계산함에 있어서 고려하여야 하는 사항은 에이전트에 주어지는 최대힘의 한계가 있어야 한다. 즉, 하나의 에이전트가 받을 수 있는 힘의 최대치는 제한되어야 한다.

```
Vector Steerings::GetForce(Agent *agent,
                           AgentList neighbor){
  Vector3 tForce=0.0, force;

  if (bSeparation){ // 분리힘 계산
    force= C_sep*Separation(m_agentList);
    if(tForce+force >= 최대힘) return tForce;
    tForce += force;
  }

  if (bAlignment){ // 정렬힘  계산
    force= C_ali*Alignment(m_agentList);
    if(tForce+force >= 최대힘) return tForce;
    tForce += force;
  }

  if (bCohesion){  // 결합힘  계산
    force= C_coh*Cohesion(m_agentList);
    if(tForce+force >=최대힘) return tForce;
    tForce += force;
  }

  if (bArrive){  //도착힘 계산
    force = C_arr*Arrive();
    if(tForce+force >= 최대힘) return tForce;
    tForce += force;
}

  if (bWander){  //배회힘 계산
    force = C_wan*Wander();
    if(tForce+force >= 최대힘) return tForce;
    tForce += force;
  }
  return tForce;
}
```

[그림 7] 우선순위가 적용된 조종힘 계산

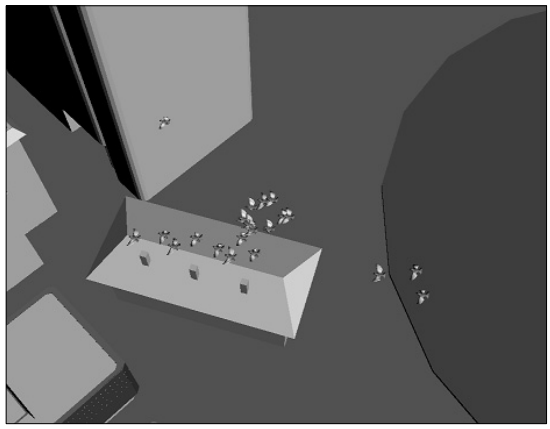
본 논문에서는 우선순위가 높은 조종힘부터 계산함으로써 최대힘의 한계를 제어하였다. 가장 우선순위가 큰 조종힘으로 '분리하기'가 적용되었으며, 다음으로 '정렬하기', '결합하기', '도착하기' 및 '배회하기'가 순서적으로 적용되었다. 예를 들어 '먹이먹기' 상태의 힘은 [표 1]에 따라 '분리하기', '도착하기' 및 '배회하기' 힘들이 순차적으로 적용된다. [그림 7]은 이에 대한 의사코드이다. 라인 4에서 *bSeparation* 변수의 값은 *Steerings::SeparationOn*, *Steerings::SeparationOff*에 의하여 설정된다. 마찬가지로 *bAlignment, bCohesion, bArrive* 및 *bWander*도 관련 *XxxOn, XxxOff* 함수에 의하여 설정된다.

4.4 시뮬레이션

비둘기의 무리 짓기에 대한 시뮬레이션은 상기와 같이 설계 및 구현으로만 완성되는 것은 아니다. 자연스러운 집단행동에 대한 시뮬레이션을 위해서는 수많은 시뮬레이션을 통하여 [표 1]에서 제시한 다양한 상수에 대한 값을 찾아야 한다. 이러한 상수들은 각각 독립적인 것이 아니므로 하나의 값은 다른 하나의 값에 영향을 주기 때문에 많은 시행착오를 거쳐 가면서 적절한 값을 찾아야 한다. [표 2]는 다양한 시뮬레이션을 통하여 가장 자연스러운 집단행동을 보여주는 값들이다.

[표 2] 시뮬레이션을 통하여 얻은 계수값

상태	C_{sep}	C_{ali}	C_{coh}	C_{arr}	C_{wan}
Flying	4.0	8.0	1.0	N/A	1.0
Landing	4.0	N/A	N/A	1.0	N/A
TakingOff	4.0	N/A	N/A	1.0	N/A
Eating	4.0	N/A	N/A	1.0	1.0



[그림 8] '날아가기' 상태의 시뮬레이션 장면

이러한 계수의 값은 조종힘을 계산하는 프로그램에 의존하기 때문에 값 그 자체로는 아무런 의미가 없다. 단지 상대적인 값으로 어느 힘이 다른 힘에 비하여 얼마나 많이 적용되는지를 알 수 있는 것이다. 예를 들어 분리힘은 모든 상태에 다 적용되지만 그들의 가중치 값들은 약간씩 다르다. 또

한 *Landing* 상태에서는 분리힘과 도착힘이 적용되는데 이들의 비가 4:1이라는 것을 알 수 있는 것이다. [그림 8]은 완성된 시스템에서 도심의 빌딩과 공원사이로 '날아가기'를 하고 있는 비둘기 무리의 시뮬레이션 장면이다.

5. 결 론

본 논문은 무리 짓기 기술을 이용하여 비둘기 떼들에 대한 집단행동의 시뮬레이션에 대해 연구한 것이다. 대부분의 기존 연구가 무리에 대하여 하나의 상태에 대한 자연스러운 행동에만 초점이 맞추어져 연구된 반면에 본 논문에서는 이를 확장하여 무리들이 다양한 상태를 가질 수 있도록 상태별 집단행동을 자연스럽게 시뮬레이션 하는 것이다. 비둘기 무리의 액션모델을 '날아기기', '내려앉기', '날아오르기' 및 '먹이먹기'로 정의하여 각 액션모델별로 자연스러운 집단행동을 나타날 수 있는 조종행동을 설계하였다. 각 액션모델을 효율적으로 관리하기 위하여 유한상태기계 기술을 도입하여 설계하였다. 설계된 시스템은 Ogre 그래픽 엔진을 이용하여 구현되었다. 구현된 시스템을 통하여 다양한 실험하였고, 이 결과로 부터 각 액션모델에 포함된 단독 또는 집단 조종행동에 대한 적절한 가중치를 찾아서 제시하였다.

본 논문은 자연에서 나타나는 다양한 집단행동 중의 하나인 비둘기 무리에 대한 시뮬레이션을 연구한 것이다. 향후 연구 과제로는 이를 확장하여 다양한 다른 무리의 집단행동에 적용하는 것이다. 이 경우 선택된 무리의 액션모델은 그 무리의 특성에 따라 다시 정의되어야 한다. 그러나 액션모델을 관리하는 유한상태기계 및 시스템 구조는 재사용될 수 있을 것이다. 또 다른 향후 연구 과제로는 다수의 무리들에 대한 상호작용에 대한 연구이다. 즉, 비둘기 그룹, 비둘기 그룹과 다른 종류의 그룹간에 그룹행동에서 발생하는 상호 영향력에 대한 연구로 본 연구의 결과를 확장하는 것이다.

참고문헌

[1] Reynolds, C. W., "Flocks, Herds, and Schools: A Distributed Behavioral Model", SIGGRAPH, 21(4), pp. 25-34, 1987.

[2] Reynolds, C. W., "Steering Behaviors For Autonomous Characters", Proceedings of Game Developers Conference, pp. 763-782, 1999.

[3] Iain D. Couzin, Jens Krause, Richard James, Graeme D. Ruxton and Nigel R. Franks, Collective Memory and Spatial Sorting in Animal Groups, J. theory Biol., 2002, pp. 1-11.

[4] Mat Buckland, "Programming Game AI by Example", ISBN 1556220782, Wordware Publications, 2005.

[5] N. Bell1, Y. Yu1 and P. J. Mucha2, "Particle-Based Simulation of Granular Materials", Eurographics/ACM SIGGRAPH Symposium on Computer Animation, 2005.

[6] 이재문, "이전 k 개의 가장 가까운 이웃을 이용한 무리 짓기에 대한 공간분할 방법의 개선", 한국게임학회 논문지, 제9권 제2호, 2009.

[7] Jae Moon Lee, Se Hong Cho, Rafael A. Calvo, "A Fast Algorithm for Simulation of Flocking Behavior", IEEE Consumer Electronics Society's Games Innovation Conference, London, 2009.

[8] 구태훈, 김진호, 이석규, 이재문, 정인환, "비둘기들의 집단행동 시뮬레이션", 한국게임학회 추계학술발표대회, 2009.

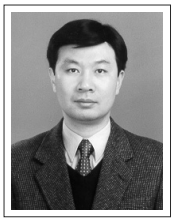
[9] http://www.red3d.com

[10] http://www.ogre3d.org/docs.

이 재 문 (Lee, Jae Moon)

1986 한양대학교 전자공학과(학사)
1988 한국과학기술원 전기및전자공학과(석사)
1992 한국과학기술원 전기및전자공학과(박사)
1994-현재 한성대학교 공과대학 멀티미디어공학과 교수

관심분야 : 데이터베이스, 기계학습, 게임프로그래밍



조 세 홍 (Cho, Sae Hong)

1983.2 연세대학교 3학년 수료
1991 (미)캘리포니아 주립대학교 CS 졸업 (학사)
1996 (미)애리조나 주립대학교 (CSE, 석사)
1999 (미)애리조나주립대학교 (CSE, 박사)
1999.9-2002.2 대구대학교 공과대학 정보통신학부 전임강사
2002.3-현재 한성대학교 공과대학 멀티미디어 공학과 교수

관심분야 : 멀티미디어 응용, 가상현실, 가상교육, 게임제작, 디지털 콘텐츠