Journal of Korea Game Society 2014 Oct; 14(5): 87-96
http://dx.doi.org/10.7583/JKGS.2014.14.5.87

게임 프로그래밍

에너지 모델 기반으로 한 몬스터의 자연스러운 행동 패턴 구현

이재문, 임승규
한성대학교 멀티미디어공학과
{jmlee, sklim}@hansung.ac.kr

Implementation of Natural Behavior Patterns of Monster based on Energy Model

Jae Moon Lee, Seong Kyu Lim
Dept. of Multimedia Engineering, Hansung University

요 약

게임 개발에서 사실성은 몰입감을 높이기 위해서 중요한 요소로 고려된다. 이를 위하여 본 논문은 RPG에서 몬스터들의 이동을 기존의 에너지 모델을 적용하여 '정글 마스터' 게임을 개발하였다. 이 게임의 주요 시나리오는 정글에서 동물들이 살아남기 위하여 서로 공격하는 것이다. 기존의 에너지 모델이 추격 몬스터와 도망 몬스터가 1:1인 환경인 반면에 본 논문에서는 게임을 보다 재미있게 하기 위하여 그것을 n:1로 확장하였다. 그 결과 에너지 모델이 실제 게임에서도 효과적으로 적용될 수 있음과 보다 자연스럽게 이동하는 몬스터들을 구현할 수 있음을 보였다.

ABSTRACT

In developing games, realism is considered as an important factor to increase immersion. To do this, the paper developed 'Jungle Master' game applying the conventional energy model to movement of monsters in RPG. The main scenario of the game is that animals attack each other in order to survive the competition in jungle. While chasing monster : fleeing monster is 1:1 in the conventional energy model, it is extended as n:1 in order to increase fun. As the results, this paper showed that the energy model can be effectively applied to the real game and monsters can be implemented so that they can move in natural.

※ 본 연구는 한성대학교 교내연구비 지원과제임.
Received: Sept. 15, 2014 Accepted: Oct. 13, 2014
Corresponding Author: Jae Moon Lee(Hansung University)
E-mail: jmlee@hansung.ac.kr

ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

많은 RPG(Role-Playing Game)에서 비논리적인 몬스터의 움직임을 종종 관찰할 수 있다. 이것은 대부분의 경우 몬스터의 움직임이 이론적 근거보다는 개발자의 경험에 근거하여 개발되기 때문이다. 이러한 게임은 사용자의 몰입감을 해칠 수 있는 요소가 될 수 있다. 본 논문에서는 이러한 몬스터의 움직임을 에너지 이론에 기반하여 개발함으로써 비논리적 움직임을 극복하는 사례를 보이고자 한다.

동물들의 먹이 사슬에 기초하여 자연스런 생태계를 시뮬레이션하는 많은 연구가 이루어져 왔다[5,7,8,9,11]. 본 논문의 정글 마스터 게임은 [8]에서 제시한 에너지 모델을 기반으로 한다. 정글 마스터 게임은 정글에서 다양한 동물들이 먹이 사슬에 기초하여 약육강식하는 게임이다. 이때 게임의 사실성을 증대하기 위하여 모든 몬스터의 움직임을 [8]에서 제시한 에너지 모델을 사용하여 제어한다. 본 논문에서 게임은 유니티3D[6]를 이용하여 구현되었다. 유니티3D는 멀티 플랫폼을 제공하나 본 게임은 스마트 폰에 최적화되도록 개발함으로써 사용자가 장소에 제한받지 않고 게임을 할 수 있도록 하였다.

2장에서는 [8]에서 제시한 에너지 모델을 소개하며, 3장, 4장에서는 정글 마스터 게임의 설계, 구현 내용을 각각 설명할 것이다. 5장에서는 구현된 게임에 대한 주요 장면과 문제점에 대하여 논하고 마지막 6장에서 결론을 논할 것이다.

2. 관련 연구

2.1 몬스터 이동에 대한 에너지 모델

RPG 게임에서 몬스터들의 사실감 있는 이동을 위한 다양한 연구가 되어 왔다[2]. 대표적인 연구 중의 하나가 에너지 모델 기반 몬스터의 추격-도망을 제어한 [8,9]이라고 판단한다. [8]에서는 몬스터의 보유 에너지를 기준으로 추격, 배회하기 등의 행동을 결정지었으며 행동에 따른 소모 에너지를 통하여 논리적인 움직임을 제안하였다.

본 논문에서는 추격 몬스터(포식자)를 i라 하며, 도망 몬스터(희생자)를 j라 한다. 또한 m_i, p_i, v_{mi}, v_i, a_i, E_i는 각각 몬스터 i에 대한 현재의 질량, 위치, 최대속도, 속도, 가속도, 에너지를 표시한다. 또한 본 논문에서 언급되는 질량과 에너지, 시간은 스칼라이고 위치, 속도, 가속도, 힘은 벡터이다. 표시를 간단히 하기 위해 스칼라와 벡터를 구분하기 위한 별도의 표시는 생략한다.

[8]에서는 몬스터들이 작은 거리($\triangle r$)를 이동하기 위해서 사용되는 소모 에너지와 일정 거리(d)를 이동하기 위하여 필요한 에너지를 제시하였다. 먼저 $\triangle t$ 시간 동안 작은 거리를 이동하기 위하여 필요한 에너지를 (eq. 1)과 같이 제시하였다.

$$E_u(i,j) = (f_p(i,j) + m_i\mu)(v_i + \frac{1}{2}\frac{f_p(i,j)}{m_i}\triangle t)\triangle t \quad \text{(eq. 1)}$$

(eq. 1)에서 μ는 마찰계수이며, $f_p(i,j)$는 작은 거리를 $\triangle t$ 시간동안 이동하기 위하여 가해지는 힘이다. 이 힘은 [1,2]에서 제안한 기본적 행동 모델에 의하여 구해진다. 또한 [8]에서는 몬스터간의 거리, 속도, 질량 등을 고려하여 추격 몬스터가 도망 몬스터를 잡기위한 필요에너지 $E_n(i,j)$를 물리적 이론을 바탕으로 다음과 같다고 제시하였다.

$$E_n(i,j) = [\frac{1}{2}m_i(v_{mi}^2 - v_{oi}^2) + m_i\mu\frac{v_{mi}d}{v_{mi}-v_{mj}} + m_i\mu\frac{\beta v_{mi}v_{mj}}{2(v_{mi}-v_{mj})}] \quad \text{(eq. 2)}$$

(eq. 2)에서 i, j는 추격 몬스터와 도망 몬스터를 의미하며, m_x, v_{mx}, v_{ox}는 각각 x의 질량, 최대속도, 초기 속도를 의미한다. 또한 d는 추격 직전 몬스터간 거리를 나타내며, β는 추격 몬스터와 도

망 몬스터가 정지 상태에서 최대 속도 v_{mi}, v_{mj}에 도달하는 시간을 각각 t_{mi}, t_{mj}라 할 때, $\beta = t_{mi} - t_{mj} (\geq 0)$이다.

2.2 유니티3D 게임 엔진

유니티3D는 게임 개발 도구이자 엔진이다. 인터랙티브한 3D 및 2D 게임 개발을 위한 직관적 도구와 신속한 워크플로가 완전히 통합된 저작 도구를 제공한다[6,10]. 강력한 물리엔진, 렌더링과 멀티 플랫폼을 지원함으로써 웹, 안드로이드, iOS 등에서 운영되는 게임을 한번에 개발할 수 있다.

유니티3D는 장면(Scene)단위로 게임에 필요한 객체를 트리 형태로 관리하며, 매 프레임마다 트리에 존재하는 모든 객체에 대하여 Update함수를 호출해 주기 때문에 개발자는 대부분의 기능을 이 함수 안에서 구현함으로써 쉽게 게임 개발을 할 수 있다. 본 논문에서 개발하고자하는 정글 마스터 게임도 유니티3D 게임 엔진을 사용한다.

3. 정글 마스터 게임 설계

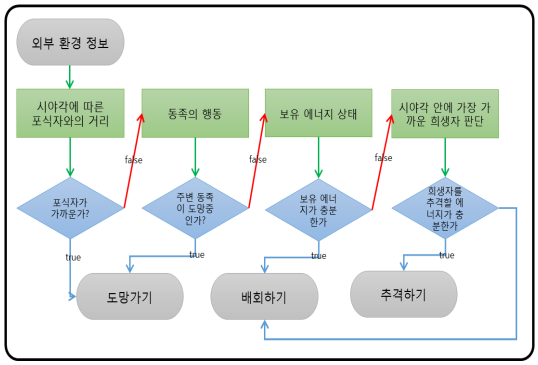
3.1 게임 시나리오

정글 마스터 게임 장르는 시뮬레이션 분야의 생존게임이다. 게임에 등장하는 몬스터들은 곰, 늑대, 멧돼지, 여우, 사슴, 토끼로 총 6가지 종류가 있으며 각각의 몬스터들은 생존을 위해 매 시간마다 추격 또는 도망 등의 알맞은 행동을 선택해야 한다. 사용자도 곰, 멧돼지, 여우 등의 캐릭터 중 하나를 선택하여 자신의 캐릭터보다 약한 몬스터를 잡아먹거나 강한 몬스터를 만났을 시 도망가는 행동을 하여 생존해야 한다. 몬스터 사냥에 성공하게 되면 몬스터의 생태계 등급에 따라 에너지를 회복하며, 생존에 필요한 아이템과 생존점수 보상 등이 실시간으로 이뤄진다. 최종적으로 게임은 보유 에너지가 0이 될 때 종료되며 생존 점수에 따라 사용자의 게임 랭킹이 결정된다.

각각의 몬스터들의 인공지능은 다음과 같이 행동을 결정하여 움직인다. 몬스터들이 매순간 첫 번째로 판단하는 것은 주변의 환경정보를 파악한 후에 알맞은 행동을 취하는 것이다. 주변 환경정보라 함은 현재 몬스터의 가까운 시야각 내에 자신을 해칠만한 추격 몬스터가 있는지, 현재 보유 에너지 상태 등을 파악한다. 그리고 이 정보를 통하여 추격하기, 도망가기, 배회하기[1,2,3,4] 등의 알맞은 행동을 결정한다. 그리고 행동결정에 따라 조종힘을 발생시켜 원하는 방향으로 몬스터가 움직이도록 한다. 이 때 발생된 조종힘으로 인해 에너지가 소모되며 이를 반영하여 보유에너지가 지속적으로 삭감 된다.

3.2 몬스터의 의사 결정 우선순위

몬스터들은 정글에서 생존하기 위하여 때로는 다른 몬스터들을 추격하여야 하며, 때로는 도망가야 한다. 본 논문에서 개발한 게임에서 몬스터는 우선순위에 따라 의사결정을 하게 되는데 이것은 동물들의 실제 행동 패턴을 최대한 적용하기 위함이다.



[Fig. 1] Priority of decision for pursuit and flee

본 논문에서 설계한 의사결정 우선순위는 [Fig. 1]과 같다. 의사 결정에 있어 추격 몬스터에 대한 위험 요소를 가장 먼저 고려한다. 이 때 첫 번째로 몬스터의 시야각 내에 있는 추격 몬스터를 거리에 따라 판단하였으며 시야각은 몬스터가 바라보는 전

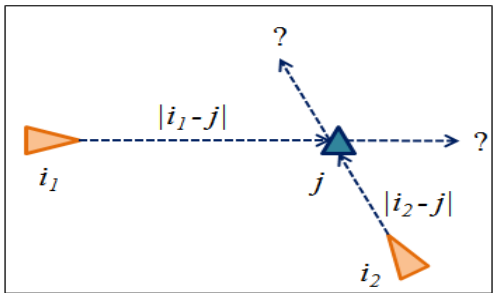
방으로 제한하였다. 두 번째 조건에서는 동물들의 집단성을 반영하며 동족이 도망 의사결정을 한 경우 위협상황을 인지하고 도망가기 의사결정을 하도록 하였다.

세 번째와 네 번째는 먹이 사슬의 하위 계층몬스터들을 추격하는 의사 결정 요소들이다. 세 번째 의사결정에서는 몬스터 자신의 보유 에너지를 판단한다. 에너지가 충분하다면 배회하기를 통해 시야각만을 바꿔가며 휴식을 취할 것이다. 이것은 주변의 추격 몬스터가 다가옴을 경계하기 위함이다. 마지막 네 번째 의사결정은 보유에너지가 충분지 않고 배고픈 상태일 때 시야각 내에 가장 가까운 도망 몬스터를 탐색고 도망 몬스터를 추격할 수 있는 에너지가 충분한가를 판단한 뒤에 추격을 시도하는 것이다. 만약 추격할 수 있는 충분한 에너지가 없다면 배회하기를 통해 시야각을 바꾸며 최적의 도망 몬스터를 탐색한다. 이 같은 4단계의 단계별 상황판단을 토대로 몬스터가 합리적인 의사결정을 내리도록 설계하였다.

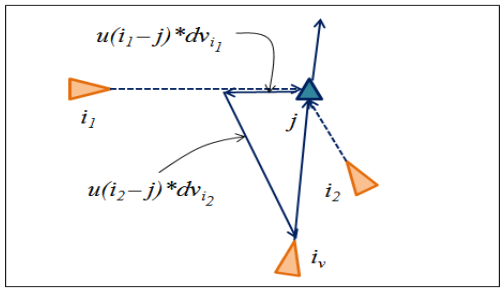
3.3 몬스터의 도망 알고리즘

일반적으로 추격 몬스터가 추격을 할 때 도망 몬스터는 추격 몬스터의 반대 방향으로만 도망간다. 이것은 [8]에서 추격 몬스터에게 필요한 에너지 계산에서도 도망 몬스터는 추격 몬스터의 반대 방향으로만 도망간다는 것을 가정하였다. 그러나 현실 또는 게임 세계에서 이러한 가정은 추격 몬스터와 도망 몬스터가 1:1인 경우에만 올바르게 동작하고, 두 마리 이상의 추격 몬스터가 존재하는 경우 문제가 발생한다.

[Fig. 2]는 두 마리의 추격 몬스터 i_1, i_2가 도망 몬스터 j 주변에 있는 상황이다. 여기서 $|i_1-j|$과 $|i_2-j|$는 추격 몬스터와 도망 몬스터사이의 거리를 나타낸다. 이러한 상황에서 몬스터 j는 몬스터 i_1으로부터 멀어져야 할지 몬스터 i_2로부터 멀어져야 할지 갈팡질팡 하는 모습을 보인다. 이런 경우 도망 몬스터 j는 추격 몬스터 i_1과 i_2로부터 동시에 멀어지는 힘을 발생 시켜야 한다. 이러한 문제를 해결하는 다양한 방법이 있으나, 본 논문에서는 다음과 같은 방법으로 문제를 해결하였다.



[Fig. 2] Problem for a monster to flee from multiple chasers



[Fig. 3] Location of a virtual chaser

[Fig. 2]와 같이 두 마리의 추격 몬스터와 한 마리의 도망 몬스터가 있는 경우를 고려하자. 그림에서와 같이 $|i_1-j| \geq |i_2-j|$이라고 한다면 도망 몬스터 j는 추격 몬스터 i_1보다 추격 몬스터 i_2를 더 많이 고려하여 도망가야 할 것이다. 본 논문에서는 이에 필요한 힘을 일관성 있게 발생시키기 위하여 여러 추격 몬스터들을 하나의 가상 몬스터로 추상화하고 j를 이 가상 몬스터 i_v로부터 멀어지는 방향으로 도망가도록 설계했다. [Fig. 3]는 이러한 가상 몬스터의 위치를 계산하는 방법에 대한 하나의 예이다. 여기서 추격 몬스터는 i_1, i_2, ..., i_k, ..., i_m으로 표시되며 m은 개수이다. [Fig. 3]에서는 추격 몬스터가 2개인 경우($m=2$)의 예이다. $u(i_k-j)$를 벡터 i_k-j의 단위 벡터라 하고, dv_{i_k}를

$(1-\frac{|i_k-j|}{\sum_{k=1}^{m}|i_k-j|})\sum_{k=1}^{m}|i_k-j|$라 하자. 이 경우 가상의 추격 몬스터 i_v는 [Fig. 3]와 같이 계산된다. [Fig. 3]의 경우를 일반화 시키면 다음과 같은 가상의 추격 몬스터의 위치와 방향을 구할 수 있다.

$$i_v = j + \sum_{k=1}^{m}(u(i_k-j)\times dv_{i_k}) \qquad \text{(eq. 3)}$$

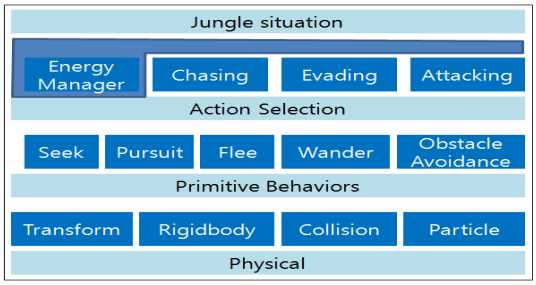
상기와 같이 다수의 추격 몬스터를 하나의 추격 몬스터로 추상화함으로써 (eq. 1)에 기초한 도망을 적용할 수 있다.

4. 정글 마스터 게임 구현

4.1 정글 마스터 게임 구현 환경

본 논문의 게임은 스마트 폰을 타겟으로 개발되었으며, 개발 환경은 Windows 기반 PC에서 Unity 4.2 엔진(C#, Android Jelly Bean, 테스트 장비 : Pantech Vega N°6)[6]을 사용하였으며 이 엔진은 멀티 플랫폼을 지원한다.

게임에서 몬스터의 모든 움직임을 제어하는 클래스는 monobehavior을 상속받은 MovingEntity이다. 모든 몬스터는 이 클래스의 객체를 갖고 있으며 매 순간마다 이 객체를 참조하여 주변 환경에 따른 판단과 행동결정, 그리고 조종힘 등을 관리하게 된다. [Fig. 4]는 유니티3D를 이용하여 구현한 내용이다. 각 몬스터는 추격, 도망, 공격이라는 액션을 선택할 수 있도록 했다. 추격, 도망, 공격은 게임 환경에 따라 찾아가기, 추격하기, 도망가기, 방황하기, 장애물 피하기 등의 기본 행동들[1,2,3]을 선택하고 이들을 통하여 필요한 조종힘을 계산한다. 필요한 조종힘을 구한 후, 유니티3D에서 제공하는 물리 기능을 이용하여 새로운 위치를 결정도록 구현하였다.



[Fig. 4] Game architecture in Unity3D

4.2 몬스터의 초기 에너지 설정

게임 초기에 몬스터들을 설정하기 위해서는 각 몬스터별 초기 에너지가 필요하다. 초기 에너지는 (eq. 2)에 근거하여 구했다. 임의의 몬스터에 대한 초기 에너지를 계산하기 위하여 이 몬스터와 가장 가까운 최하위 계층 몬스터를 선정하고 이를 도망 몬스터로 간주하여 (eq. 4)와 같이 계산했다.

$$E_{st}(i,j) = E_n(i,j)\times 10\times rand(0.7,1) \qquad \text{(eq. 4)}$$

(eq. 4)에서 i는 $E_{st}(i,j)$를 부여받는 몬스터이며, j는 주변의 가장 가까운 최하위 계층 몬스터이다. (eq. 4)에서 $E_n(i,j)$ 계산에 필요한 다양한 파라미터 값들은 [Table 2]의 값을 적용했다. 특히, d는 유니티 거리 40으로 고정하였는데 이것은 스마트폰에서 추격 몬스터와 도망 몬스터를 한 화면에서 볼 수 있도록 하기 위함이다. (eq. 4)에서 10을 곱한 이유는 추격 몬스터가 도망 몬스터를 추격하여 포획에 성공하기 위해서는 추격 과정에서 나타나는 다양한 장애물 등을 피해야 하므로 (eq. 2)의 에너지만으로는 대부분의 경우 포획에 실패하기 때문이다. 마지막으로 랜덤 함수를 곱한 이유는 각 몬스터마다 체력에 대한 개별 특성을 부여하였다. 예를 들어 현실세계의 같은 종의 사슴이라도 체력이 좋은 개체가 존재하는 반면에 약한 개체도 존재한다. 이점을 반영하여 랜덤 변수를 두어 초기 에너지를 부여했다.

4.3 몬스터의 도망/추격 알고리즘 구현

본 논문에서 제어하는 추격 몬스터와 도망 몬스터는 매 순간마다 조종힘을 계산하고, 계산된 조종힘에 따라 새로운 위치로 이동한다. 또한 추격 몬스터의 경우 필요한 에너지를 축적하기도 하고, 반대로 축적된 에너지를 사용하여 도망 몬스터를 추격하기도 한다. 다음 [Fig. 5]은 도망 몬스터를 제어하는 알고리즘이다. [Fig. 5]에서 *SearchNearestPursuer*는 추격 몬스터를 찾는 함수이다. 앞에서 설명 하였듯이 추격 몬스터는 여럿이 존재할 수 있다. 따라서 본 게임의 구현에서 *SearchNearestPursuer*는 이러한 다수의 추격 몬스터들을 하나의 가상 몬스터로 추상화하는 과정을 포함하고 있다. [Fig. 6]은 다수의 추격 몬스터가 추격할 시에 도망 몬스터가 최적의 경로로 도망가기 위해 가상 추격 몬스터의 위치를 구하는 알고리즘을 보여준다. 도망 몬스터가 추격을 피해 도망을 성공할 때, 즉 도망가기 행동 이후 다른 행동을 결정 할 때 가상 추격 몬스터 배열이 초기화 되는 점은 표현을 간단히 하기 위해 생략했다.

```
Update_Prey(){
    //시야각 내 가장 가까운 추격 몬스터 탐색
    pursuer = SearchNearestPursuer(EntityLevel)
    // 안전거리 내에 추격 몬스터가 없을 시 배회하기 결정
    if(pursuer==null){
        SetOnWander()
        return;
    }
    // 안전거리 내에 추격 몬스터가 있을 시 도망가기 결정
    if(distance(pursuer) < getDangerArea() ){
        SetOnEvade(pursuer)
        return;
    }
    // 주변 동료가 도망간다면 도망가기 결정
    GameObject[] SameSpecies
        = GameObject.FindObjectsWithTag (EntityLevel);
        for(i=0;i<SameSpecies.Count(); i++){
            if(Distance(j) < GetSameSpeciesArea()){
                SetOnEvade(j.GetPursuer())
                return;
            }
        }
}
```

[Fig. 5] Evading Algorithm

```
SearchNearestPursuer(int EntityLevel){
    //근처에 있지 않은 추격 몬스터를 제거
    for(i=0; i<PredetorThreat.Count(); i++){
        if(Distance(j)> SafeArea() ){
            PredetorThreat.RemoveAt(i--)
        }
    }
    if(PredetorThreat.Count()==0)    return null;

    // 추격 몬스터를 탐색
    for(i=EntityLevel+1; i<=MaxLevel; i++){
        GameObject[] ALLPredetors
          = GameObject.FindObjectsWithTag (EntityLevel);
        for(j=0;j<ALLPredetors.Count(); j++){
            if(Dot(j)>=0 && Distance(j) < SafeArea()
                    (&& !IsDuplication())){
                PredetorThreat.add(j);
            }
        }
    }
    // 가상 추격몬스터 생성 및 eq (3) 계산
    virtualPursuer= new VirtualPursuer();
    for(i=0; i<PredetorThreat.Count(); i++){
        VectorScalarRatio
          =(1-(Distance(j)/ TotalDistance());
        VirtualPursuer.position
          += Dot(j)*T TotalDistance()*VectorScalarRatio
    }
    return virtualPursuer;
}
```

[Fig. 6] Evading Algorithm for Multiple Chasers

```
Update_Pursuer(){
    // 배고픔 체크
    if(IsHungry() == false){
        SetOnWander()
        return;
    }
    // 추격중이면
    if(prey != null){
        E(this) -= E_u(i, j) // 식 (1) 계산 및 적용
        Pursuit(i, j)
        return;
    }

    // 주변 하위 계층 몬스터 탐색
    prey = SearchNearestPrey(EntityLevel)
    if(prey==null){
        SetOnWander()
        return;
    }

    E_n(prey) = NeedEnergy(prey) // 식 (2) 계산
    if(E(this) >= E_n(prey)){
        setOnPursuit(this,prey)
    else{
        SetOnWander()
        return;
    }
}
```

[Fig. 7] Chasing Algorithm

몬스터 추격 알고리즘은 도망 알고리즘에 비하여 비교적 단순하다. [Fig. 7]은 몬스터의 추격에 대한 알고리즘이다. 먼저 배고픈 상태를 확인하고, 가장 가까운 먹이 몬스터를 찾는다. 보유 에너지가 초기 에너지의 0.5배보다 적으면 배고픈 상태로 설정하였다. 그리고 먹이 몬스터를 추격하는데 필요한 에너지를 계산하고, 이를 자신이 가진 에너지와 비교하여 추격 유무를 결정한다.

5. 구현 결과

5.1 구현된 게임의 주요 장면

일반적으로 동물의 종족 개체수를 보면 피라미드 구조로 이뤄져 있으며 만약 생존 경쟁의 상위 개체수가 증가하게 되면 생태계의 규칙이 깨지게 되어 생태계가 파괴되게 된다. 이와 같은 이유로 게임이 안정적으로 유지되기 위해서는 몬스터의 개체 수 또한 논리적 구조로 이뤄져야만 한다. 본 논문에서 개발한 정글 마스터에서 몬스터는 총 6가지로 곰, 늑대, 멧돼지, 여우, 토끼, 사슴 등으로 이뤄져 있다. 또한 먹이 사슬의 상위/하위 계층 관계는 곰 > 늑대 > 멧돼지 > 여우 > 사슴, 토끼 순이며 개체 수는 상위 계층 몬스터에 대하여 1:2 비율을 유지하도록 하였다. [Table 1]에서와 같은 규칙으로 개체 수를 유지하였으며 개체수를 지속적으로 확인하는 것이 아니라 시간의 일정 간격을 두어 각 몬스터의 개체수를 판단하여 부족한 개체수를 동족근처에 다시 생성되도록 하였다.

[Table 1] Type and number of monsters

Monster	bear	wolf	wild pig	fox	deer	rabbit
Number	1	2	4	8	16	16

[Table 2] Values of parameters in (2) and (4)

parameters	bear	wolf	wild pig	fox	deer	rabbit
m_x	10.0	8.5	7	3.5	2.5	1
v_{mx}	10.0	8	7	6	5	5
etc	$v_{ox}=0,\ \beta=1,\ d=40$					

[Table 2]는 (eq. 2)와 (eq. 4)에서 사용된 파라미터들에 대한 설정 값들이다. 이러한 값들은 스마트폰에서 게임하기에 가장 적합한 값들을 다양한 실험을 통하여 가장 적절하다고 판단되는 값들이다.

(a) Starting to chase (b) Failure to chase

(c) Starting to evade (d) Starting to evade with peers

[Fig. 8] Screen shots of 'Jungle Master' game

본 논문에서 개발한 게임은 유니티3D에서 제작되었고, 안드로이드 폰에서 동작하도록 최적화 되었다. [Fig. 8]는 구현된 게임의 주요 이미지들이다. [Fig. 8](a)는 곰이 토끼를 발견하고 추격을 시작하기 직전의 상황에 대한 장면이다. 에너지 부족 상태는 배고픔을 느끼는 상태이며 몬스터는 자신보다 약한 도망 몬스터를 탐색하여 추격을 시도한다. [Fig. 8](b)는 추격 중에 주변 환경의 영향으로 모든 에너지를 다 사용하여 에너지가 고갈된 상태여서 추격을 포기하는 장면이다. [Fig. 8](c)는 혼자서 놀고 있는 토끼가 위험 거리 내에서 곰을 발견하고 도망을 시작하는 장면이고, [Fig. 8](d)는 여

러 마리의 토끼가 존재하는 상황에서 곰이 출현하자 동시에 도망을 시작하는 장면이다.

5.2 문제점 및 개선 방안

게임 개발을 하며 몇 가지 문제점이 발견되었으며 이들을 개선해야 할 필요성이 있다. 첫 번째로 개체 수에 따라서 연산량이 비례하여 늘어난다는 점이다. Android Jelly Bean Pantech Vega N°6로 테스트를 해보았을 때 개체수가 50마리가 되면 눈에 띄게 프레임 수가 줄어드는 것을 확인 할 수 있었다. 이의 원인은 과다한 렌더링 시간으로 파악되며, 해결책으로는 카메라에 보이지 않는 개체에 대해서는 유니티3D의 장면에서 일시적으로 제거하는 기법이 필요하다.

두 번째는 맵 크기의 제한이다. 도망 몬스터의 움직이는 반경을 제한하지 않는다면 시간이 지날수록 몬스터들이 맵의 외곽 쪽으로 몰리는 현상이 일어날 수 있음을 알아내었다. 그러므로 도망 몬스터가 움직일 수 있는 반경을 제한하여야만 이 같은 문제점을 해결 할 수 있을 것이다. 이러한 문제도 큰 맵을 작은 맵으로 분할하고, 카메라에 보여지는 맵만 동적으로 장면에 삽입하는 방법이 있으나, 맵의 정교한 연결 등 실제적으로는 많은 어려움이 있다.

세 번째 실제 동물들은 무리를 짓는 습성이 있는데 본 논문에서 다루는 게임에서는 이를 다루지 못하였다. 무리성을 추가 시킨다면 훨씬 더 논리적인 움직임을 갖는 몬스터를 볼 수 있을 것이다. 이것은 추격에서의 협업 등과 함께 고려되어야 할 사항이다.

6. 결 론

본 논문에서는 자연세계의 생태계와 유사한 생존게임을 개발하여 기초논문이 얼마나 게임개발에 실효성이 있는지를 검증하였고 그 결과 게임 내에서 사용자뿐만 아니라 몬스터간의 생존을 위한 행동들을 통해 자연 생태계와 유사함을 확인할 수 있었다. 앞으로 게임뿐만 아니라 자연 생태계와 유사하도록 시뮬레이션 하는 분야에도 적용될 수 있음을 알 수 있었다.

향후 연구로써 앞서 문제점으로 제시되었던 문제들을 해결하는 것이다. 특히, 다수의 개체들이 존재하는 공간에서 무리를 지어 다니는 동물들에 대하여 추격 및 도망시 서로의 목적을 달성하기 위하여 협업에 대한 연구는 게임의 사실성과 흥미 증대를 위하여 반드시 필요한 연구라고 판단한다.

ACKNOWLEDGMENT

This research was financially supported by Hansung University.

REFERENCES

[1] Reynolds, C. W., “Flocks, Herds, and Schools: A Distributed Behavioral Model”, SIGGRAPH, 21(4), pp. 25-34, 1987.
[2] Mat Buckland, “Programming Game AI by Example”, ISBN 1556220782, Wordware Publications, 2005.
[3] Jae Moon Lee, “An Efficient Algorithm to Find k-Nearest Neighbors in Flocking Behavior”, Information Processing Letters, pp. 576-579, 2010.
[4] Xiaoyuan Luo, Shaobao Li, Xinping Guan, “Flocking algorithm with multi-target tracking for multi-agent systems”, Pattern Recognition Letters, pp. 800-805, 31, 2010.
[5] Vladimir Zhdankin and J. C. Sprott, “Simple predator-prey swarming model”, PHYSICAL REVIEW E 82, 056209-1～7, 2010.
[6] Sue Blackman, “Beginning 3D Games Development with Unity”. Wiki Books, 2012.
[7] Zhang XC, Sun G-Q, Jin Z, “Spatial dynamics in a predator - prey model with Beddinggton - DeAngelis functional response”, Phys. Rev.

E 85, 2012.
[8] Jae Moon Lee, Young Mee Kwon, “A Model of Pursuing Energy of Predator in Single Predator-Prey Environment”, Journal of Korea Game Society, v.13, no.1, pp.41-48, 2013.
[9] Jae Moon Lee, “Analysis of Behaviour of Prey to avoid Pursuit using Quick Rotation”, Journal of Korea Game Society, v.13, no.6, pp.27-34, 2013.
[10] C. J. Lim, Won Dae Han, Jeong Yun Guen, “Educational Game Making-Tool Development using Unity3D Engine: Birth of Game”, Journal of Korea Game Society, v.14, no.1, pp.29-38, 2014.
[11] Weiren Zhu, Haibin Duan, “Chaotic predator - prey biogeography-based optimization approach for UCAV path planning”, Aerospace Science and Technology, V. 32, Issue 1, pp.153 - 161, 2014

이 재 문 (Lee, Jae Moon)

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

관심분야 : 기계학습, 게임프로그래밍, 감성컴퓨팅

임 승 규(Lim, Seong Kyu)

2014년 한성대학교 멀티미디어공학과 재학

관심분야 : 게임프로그래밍, 인공지능