게임에서의 인공지능 기술

이 만 재*

● 목 차 ●

1. 서 론

초기의 컴퓨터 게임은 그래픽, 사운드 모든 면에 제약이 있어 인공지능에 대해서는 게임을 개발하는 프로그래머의 상식 수준에 의존하는 게임이 개발되었다. 게임에 있어 인공지능이 중요한 역할을 하기 시작한 것은 1990년대 후반부터이다. 그래픽이나 사운드 기능이 일정 수준에 도달하자 게이머들은 보다 자연스럽고 재미있는 게임을 요구하였으며 인공지능이 이의 돌파구가 되고 있다. 1997년의 경우 게임에 있어 인공지능에 사용되는 CPU의 비율은 5% 이하이었으나 2000년의 경우 25%의 CPU가 인공지능 처리에 사용되고 있다. 또한 1997년의 경우 인공지능 담당 프로그래머가 있는 게임 프로젝트는 25%에 불과하였으나 2000년의 경우 80%의 게임 프로젝트에서 인공지능을 담당하는 전담 프로그래머가 포함되어 있다는 결과를 보인다.

게임에서 사용되는 인공지능 기법은 FSM(Finite State Machine)과 같은 전통적인 기법으로부터 A-Life와 같은 특수한 게임에 사용되는 기법과 같은 다양한 기법이 사용된다. 본고에서는 게임에서 사용되는 인공지능 기법을 먼저 방법론에 따라 설명하고 국내에서 가장 대중적인 FPS(First Person Shooter), RPG(Role Playing Game), RTS(Real Time Strategy)의 세 가지 장르별 사례를 설명하도록 한다.

* 한국정보통신대학교 공학부 교수

2. 게임의 인공지능 기법

2.1 게임에서 인공지능의 역할

게임에서 인공지능의 역할은 여러 가지가 있다. 그 중 가장 중요한 것은 게임의 상대 역할을 수행하는 것이다. 이 경우 게임의 핵심은 인공지능과의 대결이다. 이 경우 인공지능은 게이머에게 무조건 이기는 것이 목적이 아니라 유사한 수준의 상대 역할을 수행한다. 다음으로 인공지능은 게임의 보조자 역할을 수행하기도 한다. RPG의 경우에 주인공의 보조역할을 하는 캐릭터의 인공지능을 말하며 때로는 게임 초보자를 이끌어 주는 역할을 하기도 한다. 세 번째로 NPC의 역할을 담당한다. RPG 게임에서 특정한 술집에 들어갔을 경우 그 술집에 있는 다른 손님이나 술집의 주인 역을 말한다. 이 경우 해당 NPC는 주인공과 상호작용을 하며 실제로 그 NPC가 지능적인 객체라는 인식을 게이머에게

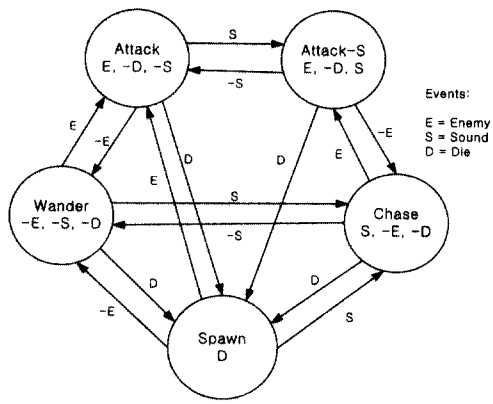
주어야 한다. 마지막으로 인공지능은 애니메이션 동작의 제어를 담당한다. 예를 들어 야구 게임의 경우 공을 1루에 던진다고 했을 경우 공을 던지는 외야수와 1루수의 움직임에 어떠한 애니메이션을 적용해야 하는 판단은 인공지능이 담당한다.

게임에서 가장 자주 등장하는 현실적인 문제는 현재의 위치에서 목표물까지 가는 경로를 찾는 것이다. 전략게임에서 목적지를 마우스로 알려주면 가장 지름길을 찾아 해당하는 장소로 이동해야 하는 문제로 전략 게임 뿐만 아니라 거의 모든 게임에 등장한다. 이에 대한 해법으로 A*라는 알고리즘이 사용된다. A* 알고리즘은 경로찾기 문제에서 비교적 빠른 시간에 목적지까지의 최적의 경로를 찾아준다. 그러나 목표물까지 가능 도중에 있는 다리가 폭파되거나 중간에 적이 길을 차단하는 경우와 같이 지형이 바뀌는 경우에는 A* 만으로 모든 경로 찾기 문제를 해결할 수 없으며 추가적인 방법이 요구된다. 경로찾기는 인공지능 문제 중에서 비교적 많은 연구가 진행된 분야이며 대부분 A* 알고리즘과 다른 해법을 함께 이용하여 해결한다.

또한 인공지능 기능을 사용하여 등장 캐릭터의 지능적인 행동을 구현한다. 게이머는 게임에서 상대편이 서툰 플레이를 하는 것을 원하지 않는다. 비록 그러한 경우를 이용해서 게임에 이겼다고 해도 그 즐거움이 크지 않기 때문이다. 따라서 게임의 상대편은 지능적인 행동을 하게 만들어야 한다. 예를 들어 전략 게임에서 몇 개의 유니트가 한 쪽 방향에서 다리를 건너려 하고 또 일부 유니트는 다리 반대쪽에서 다리를 건너려 한다면 실제로 양 편 모두 다리에서 움직이지 못하는 경우가 있다. 실생활에서 일어나지 않는 이러한 상황은 인공지능 처리가 미숙할 경우에 발생하기 쉽다. 이 경우에는 교통경찰의 역할을 담당하는 기능이 인공지능 내부에 포함되어 있어야 한다.

2.2 FSM

FSM(Finite State Machine)은 현재 가장 널리 사용되는 인공지능 처리 방식이다. (그림 1)에서 표시한 바와 같이 여러 개의 상태로 나누어지면 캐릭터의 현재 상태(state)에 따라 외부에 대처하는 방식이 결정된다. 외부의 상황이 변화하게 되면 이에 따라 state가 변화된다.



(그림 1) FSM의 기본 개념

그림에서 보는 바와 같이 캐릭터는 특정한 상태에 있을 경우에는 항상 같은 방식으로 행동한다. 예를 들어 Wander state에서 적이 발견되는(E) 상황이 발생할 경우 Attack state로 이동하게 되며 이 state 에 있는 동안은 계속 같은 행동을 한다. 만약 이 state에서 플레이어가 죽게(D) 되면 Spawn state로 바꾸며 적이 보이지 않는(-E) 경우에는 다시 Wander state 상태로 변하게 된다.

FSM 모델은 이해하기가 쉽고 프로그램으로 구현하기 쉽다. 단순하게 switch () 구문을 사용하면 가능하다. 다음은 FSM을 의사코드(pseudo code)로 구현한 일부를 보여준다.

```
switch(creature_state)
  case STATE_ATTACK:
                플레이어 쪽으로 이동
```

```
                20%의 확률로 포 발사
case STATE_RETREAT:
                플레이어와 반대방향으로 이동
```

FSM은 특별히 뛰어난 인공지능을 필요로 하지 않는 대부분의 게임에서 사용된다. FSM을 사용할 경우 state 수가 늘어날 경우에는 switch() 문장을 사용하는 대신 array 구조를 사용하여 해당 루틴에 빠르게 접근할 수 있도록 하는 기법이 사용된다. FSM의 단점은 state 수가 늘어나면 state diagram을 정리하기가 어렵고 state 변화를 가능하게 하는 외부 센서 입력 루틴이 급속도로 복잡해지며 이를 사용한 캐릭터의 행동을 예측하기가 쉽다는 점이다. 이러한 문제를 부분적으로 해결하기 위해 하나의 state를 몇 개의 substate로 나누어 해결하기도 한다.

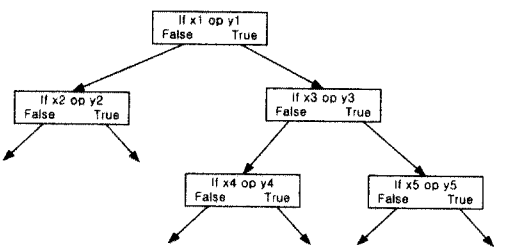
2.3 Fuzzy State Machine (FuSM)

FSM의 원리는 이해하기 쉽고 구현도 어렵지 않아 널리 사용된다. 그러나 FSM을 게임의 상태편으로 적용한 경우 게임 진행 중 유사한 경우를 여러 번 겪고 나면 인공지능의 행동을 예측할 수 있게 된다. 이는 FSM의 기본 원리가 같은 상태의 경우에는 외부에 대한 반응이 같기 때문에 일어나는 현상이다. 예를 들어 적에게 특정거리까지 근접하면 적이 뒤로 후퇴한다는 것을 알게 되면 이를 이용하여 게임을 진행시키고 상대편의 행동을 예측할 수 있게 된다.

FSM에 퍼지(Fuzzy)이론을 접목한 FuSM의 경우에는 입력과 출력에 Fuzzy 함수를 적용하여 일정한 랜덤 기능을 적용하여 동일한 외부 상황에도 다른 출력을 얻을 수 있도록 하는 방법이다. 이러한 FuSM을 사용하면 상대편의 행동을 예측하기 어렵게 되어 보다 현실적인 게임을 즐길 수 있다. 예를 들어 적과의 거리를 Near(200m), Close(500m), Far(700m) 의 선택을 확률적으로 하는 Fuzzy 함수로 정의할 수 있다. 이 경우 적과의 거리가 550m이면 약 0.7 정도의 확률로 Close 로 정의하고 0.3 의 확률로 Far라고 정의하게 되며 이러한 입력에 따라 결과를 처리하는 방법이다.

2.4 Decision Tree

앞서 설명한 FSM은 state 수가 많아지면 표현력의 한계가 생긴다. 따라서 state 수가 많아지더라도 사용할 수 있는 방법이 필요하며 이 중 가장 널리 사용되는 방식으로 Decision Tree가 있다. Decision Tree는 현재의 상태에 따라 행할 기능을 tree 형태로 나누어 해당하는 행동을 하도록 하는 것을 말한다. 미리 정해진 state가 없기 때문에 매 프레임마다 게임 로직을 처리해야 한다는 부담이 있다. 그러나 tree 구조를 잘 선택하면 게임 로직에 사용되는 시간을 최소한으로 줄일 수 있기 때문에 앞서 설명한 FSM과 같이 널리 사용된다. 이 역시 if... then...else 루틴을 반복적으로 사용하여 구현하면 되기 때문에 프로그래머의 이해가 빠르고 코딩에 어려움이 없다.



(그림 2) Decision Tree의 기본 개념

Decision Tree의 경우 state 수의 제한은 없으나 그러나 비교적 적은 수의 환경변수를 사용할 때 편리하다. 또한 결과로 얻어지는 결과가 하나 또는 아주 적은 수의 값만을 사용하는 데 적합하다. 예를 들어 적에게 접근한다면 접근하는 방법은 가장 빠른 속도로 접근하거나 전혀 이동하지 않는 경우의 두 가지만을 표시하게 된다. 따라서 해당되는 명령을 다음과 같이 구분하는 문제로 바뀌게 된다.

Walk => <forward, backward, stop>

Turn => <left, eight, none>

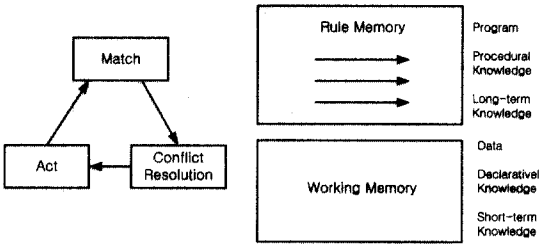
Run => <yes, no>

또한 한번 정해진 decision tree는 계속 사용되기 때문에 게임의 진행에 따라 얻어진 지식을 활용할 수 없어 플레이어에 의해 인공지능이 담당하는 캐릭터의 행동예측이 가능하다는 FSM과 유사한 단점을 갖게 된다. 이를 피하기 위해서는 최종 출력에 있어 랜덤 함수를 사용하여 FuSM과 같은 변화를 주는 방법을 사용하기도 한다.

2.5 규칙 기반 시스템

규칙 기반 시스템 (Rule Based System)은 전통적인 인공지능 학문에 기반을 둔 기법이다. Allen Newell과 Herb Simon에 의해 처음으로 제시된 개념으로 많은 규칙을 나열하고 현 상황에 해당하는 규칙을 골라내어 이 규칙에 따른 결과를 얻는 것을 말한다. 그림 3에서 보는 바와 같이 모든 규칙을 rule memory 에 보관하고 현재 상태를 working memory에 유지하여 working memory와 matching 되는 rule 을 찾아내어 이를 실제로 적용한다. 규칙이 적용되면 상태의 변화에 따라 working memory 의 값이 변화하며 이를 반복적으로 수행한다

게임의 경우 규칙 기반 시스템은 Baldur's Gate에 사용된 Infinity 엔진에 처음으로 사용된 것으로 알려졌으며 이에 대해서는 뒤에 다시 설명한다. 규칙 기반시스템의 경우 복잡한 판단이 가능하기 때문에 앞서 제시된 FSM이나 Decision Tree에 비해 높은 지능을 구현할 수 있다. 그러나 규칙의 수가 늘어날 경우 현 상태를 표시하는 state에서 해당되는 규칙을 찾아내는 matching 기능에 많은 시간이 소요된다. 복잡한 캐릭터의 조합으로 이루어지는 RPG게임의 경우에는 규칙기반 시스템을 요구한다. 그러나 단순한 지능만으로도 게이머에게 충분한 액션 게임의 경우에는 사용되지 않는다.



(그림 3) 규칙 기반 시스템의 동작원리

2.6 Planning

Planning은 현재의 상황에서 목표하는 상황에 도달하기 위해 여러 단계를 거치는 경우 이를 미리 계획하는 것을 말한다. Planning이란 이러한 계획을 만들어 내는 것을 말하며 전략게임에 있어서 건물의 건축이나 병사의 양성 순서(build order)를 정하는 경우에 사용된다.

Planning은 state-space search 방식을 사용하여 해결한다. State space search란 현재의 initial state 에 특정한 operation을 가할 경우 goal state에 도달할 수 있다는 여러 개의 operation 을 나열하고 이러한 operation 의 조합에서 목표하는 goal 을 찾기 위해 수행할 operation 의 순서를 정하는 방식이다. 게임의 경우 상대방이 있기 때문에 planning은 계획된 대로 진행되지 않는 경우가 많다. 미리 정해진 대로 건설하려는 건물이나 병사가 적에 의해 파괴되면 plan 결과는 새로 정리해야 한다. Planning 에 사용되는 operation은 앞서 설명한 규칙 기반 시스템과 유사한 점이 있다. 최종 목표인 goal 을 가능하게 하는 operation의 순서를 정리한 결과가 plan이 되기 때문이다. 일부 실시간 전략게임의 경우에는 Planning 을 처리하기 위해 스크립트 언어를 사용한다.

3. 장르 별 인공지능 적용

3.1 액션 게임의 인공지능

액션 게임은 Quake, Unreal과 같은 FPS(First

Person Shooter)가 주를 이루고 있고 Tomb Raider와 같은 TPS(Third Person Shooter), Thief 와 같은 FPS(First Person Sneaker)와 같은 변형된 형태의 게임이 있다. FPS는 한 사람의 주인공은 대상으로 하고 있기에 인공지능 기능이 단순하였으나 최근 팀의 협력을 필요로 하는 게임이 등장함과 함께 보다 복잡한 인공지능 기능을 요구하고 있다.

일반적으로 액션 게임의 인공지능은 행동(behavior), 이동(movement), 애니메이션, 전투(combat)의 4 가지 controller로 나누어 구현된다. 이중 가장 하위계층인 애니메이션 기능은 주어진 애니메이션 시퀀스를 상황에 적합하게 변형시키는 기능으로 지능이 필요한 기능은 아니다. 이동기능은 이동할 공간이 그리 넓지 않을 경우에는 비록 속도 면에서 비효율적일 지라도 A*와 같은 전통적인 알고리즘을 사용한다. 경우에 따라서는 전 지역의 이동은 global controller가 지역적인 이동은 A*를 사용하는 방식을 채택하기도 한다. Combat Controller의 미리 정해진 Camp. Joust, Ambush 와 같은 전투방식의 선택을 담당하며 상위의 controller인 Behavior controller의 메시지를 받아 처리한다. Behavior Controller는 가장 상위 개념의 인공지능 기능을 수행하며 앞서 말한 FSM 기법을 사용하여 처리하는 경우가 대부분이다.

액션 게임의 인공지능은 C 언어를 사용하는 대신 C와 유사한 스크립트 언어를 사용하는 경우가 대부분이다. 이는 C언어로 구현하기에 불편한 네트워크 기능과 인공지능 기능을 동시에 해결하기 위한 수단으로 볼 수 있다. Unreal 엔진의 경우에는 FSM 구현에 편리한 state 기능을 기본적으로 제공한다. 또한 다양한 NPC의 구현에 필요한 객체의 특성을 상속하는 기능을 제공하고 있어 기본적인 NPC의 행동모델을 일부만 변형하여 사용할 수 있도록 하는 기능을 제공한다.

3.2 RPG 게임의 인공지능

RPG 게임은 Diablo와 같은 액션 RPG와 Baldur's Gate와 같은 전통 RPG로 나눌 수 있다. Infinity 엔진은 Bioware 사에서 개발한 RPG게임 전용 엔진이다. AD&D (Advanced Dungeons and Dragons)라는 TSR 사의 전통 게임을 기반으로 한 것으로 Baldur's Gate 시리즈, Icewind Dale, Planescape와 같은 게임에 사용되어 왔다.

액션 게임의 경우에는 네트워크 기능과 같은 게임 시스템 전체에 대한 내용을 다루는 데 반해 Infinity 엔진에 사용되는 AICompile이라는 스크립트 언어는 인공지능 기능만을 다루고 있다. AICompile 언어는 IF <condition> THEN <response> END로 표현하는 여러 개의 규칙 중 하나를 선택하도록 하는 규칙 기반 시스템(Rule Base System) 구조를 가지고 있다. 이러한 문법을 사용한 예는 다음과 같다.

```
IF
 Class(LastAttackerOf(Myself(),MAGE)
   HPGT(Myself,50)
THEN
  RESPONSE #80
   Attack(LastAttackerOf(Myself),MELEE)
  RESPONSE #40
                Help()
                RunAway()
END
```

예에서는 자신을 공격한 적이 Mage 클래스이고 자신의 건강이 50 이하일 경우 80/(80+40)의 확률로 Melee방식으로 적을 공격하거나 40/(80+40)의 확률로 구원을 요청하고 도망가는 것을 표현하고 있다. 위의 예에서 볼 수 있듯이 특정 상황에서 캐릭터의 동작을 쉽게 표현할 수 있는 장점을 갖고 있다. 여기서 condition은 여러 다른 상황을 표시할 수 있기

때문에 RPG 게임에서 자주 등장하는 NPC(Non Player Character)와의 대화 표현에도 사용된다. 이렇게 얻어진 Response는 캐릭터의 메모리에 보관되어 별도로 취소되지 않는 한 그 결과를 진행하도록 되어 있다. 이는 많은 계산을 하여 얻어진 결과를 가능한 한 지속적으로 사용하기 위함이다.

RPG게임의 특징 중 하나는 등장하는 캐릭터가 수명의 PC(Player Character)와 수십 내지 수천의 NPC로 매우 많다는 것이다. 이 모든 캐릭터의 인공지능을 담당하기 위해서는 3차원 그래픽 기능에서 사용되는 LOD(Level Of Detail) 기법을 사용한다. 예를 들어 Neverwinter Nights의 경우에는 <표 1>과 같이 인공지능 기능을 구별하여 처리한다.

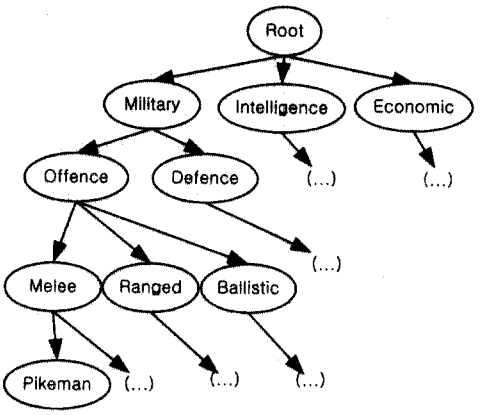
<표 1> Neverwinter Nights의 LOD 구현

LOD	설명	CPU 할당
1	PC(Player Character)	60%
2	PC와 상호작용하는 NPC	24%
3	PC와 50m 이내의 NPC	10%
4	PC와 같은 지역내에 있는 NPC	4%
5	PC와 다른 지역에 있는 NPC	2%

3.3 실시간 전략 게임의 인공지능

실시간 전략(RTS) 게임의 경우에는 상대편을 대신하는 전략적 목표를 설정하는 기능과 일시적으로 선택된 그룹의 전술적인 행동, 그리고 그룹에 속하는 개별 유닛의 단위 시간의 행동을 담당하는 기능의 인공지능이 필요하게 된다. 따라서 RTS 게임의 인공지능에서는 게임에서 플레이어가 활용할 수 있는 자원을 어떻게 할당을 결정하는 것이 중요한 문제이다. 일반적인 RTS 게임에서 플레이어는 다음과 같은 자원의 할당문제를 풀게 된다.

예를 들어 (그림 4)에서 전체 전략을 Military 40%, Intelligence 30%, Economics 30%로 정했을 경우 현재 할당된 자원이 어떻게 배분되고 있나를 하부 유니트의 자원 배분으로 판단한다. 구체적으로 현재 Military-Offence-Melee에 10%의 자원을 할당



(그림 4) 실시간 전략 게임의 기술 트리

하려는 전략을 세웠을 경우 Pikeman의 수가 부족하며 이를 증가시켜야 한다. 따라서 이러한 판단 구조를 구현하기 위해서는 Planning system을 구현하여야 한다. 다만 RTS 게임에 사용되는 Planning System은 개개인의 캐릭터의 일부로 사용되기보다 하나의 플레이어의 인공지능을 구현하는 기능에 비중을 두어야 한다.

Ensemble Studio에서 개발한 Age of Empire II는 대표적인 RTS게임이며 인공지능 처리를 위해 스크립트 언어를 사용한다. Infinity 엔진의 If <condition> then <action>과 유사하지만 다른 표현 방식을 사용한다.

```
(defrule
  (current-age == castle-age)
  (building-type-count castle == 1)
=>
  (set-goal GOTO_NEXT_AGE TRUE)
  (chat-local-to-self "Trying to get to the
    Imperial Age")
  (set-goal CAN_ATTACK FALSE)
  (disable-self)
)

(defrule
```

```
    (food-amount <= 400)
    (can-buy-commodity food)
    (goal GOING_FOR_WONDER FALSE)
    (goal GOTO_NEXT_AGE FALSE)
=>
    (set-goal RESOURCE_STATE
        RESOURCE_STATE_NEEDFOOD)
    (chat-local-to-self "Need Food!")
)
```

사용되는 reserved word 중 goal은 전략 게임에 있어 전략목표를 지정하는 boolean 변수 값을 표시한다. 이를 지정하기 위해서는 set-goal이라는 reserved word를 사용한다. 전략게임에 있어서 goal은 매우 중요한 의미를 차지한다. 게임의 목적이 적을 섬멸하는 경우와 먼저 특정 건물을 건설하는 경우 이기기 위한 전략은 아주 다르기 때문이다. 따라서 하부 유닛은 전략적인 목표를 기본 입력으로 지능을 발휘하여야 한다.

4. 최근 동향

지금까지 대표적인 게임 장르의 인공지능 구현 기능을 기술하였다. 여기서는 최근에 새롭게 등장하는 인공지능 기술 추세를 설명한다. 최근 게임은 온라인 기능이 강조되며 그 결과로 참여하는 인원간의 팀워크를 강조한다. 이에 기반하여 인공지능에도 팀 인공지능이 중요한 이슈로 등장하고 있다. 팀 인공지능을 군대조직의 운영에 기원을 두고 있다. 부대장은 전체적인 상황을 판단하고 이에 따라 해당 부대원에 명령을 내리며 부대원은 그러한 명령에 기반하여 자기 자신의 다른 입력을 고려하여 판단한다. 이러한 인공지능 방식은 명령 계층을 2 또는 3 계층으로 나누어 처리한다. 3 계층의 예를 들면 제일 높은 수준은 전략(strategy) 수준, 다음은 작전(operation) 수준, 마지막은 전술(tactics) 수준을 사용한다. 이러한 개념이 등장하기 이전의 1인칭 액션 게임은 전술 개념만을 사용하였으며 이러한 발전이 이루어졌다.

LOD(Level of Detail)이라는 단어는 인공지능에서 사용되기 이전에 3D 그래픽에서 원경에 있는 오브젝트의 표현을 단순화하여 전반적으로 게임의 속도를 높이는데 사용한 용어이다. 같은 개념을 인공지능에 적용한 것을 LODAI라 하며 현재 스크린에 보이는 캐릭터의 인공지능은 보다 구체적인 알고리즘을 사용하며 보이지 않는 캐릭터의 인공지능은 보다 단순한 알고리즘을 사용하는 것을 말한다.

영향 맵이란 게임에 등장하는 오브젝트가 각각 주변에 미치는 영향을 가지고 있어 이 영향권에 접근하는 캐릭터는 이를 계산하여 그 결과를 행동에 반영하자는 아이디어이다. 일부 게임에서 사용되는 것으로 알려져 있고 "심즈" 개발자인 윌 라이트는 이를 추후에 사용하겠다는 의사를 밝힌 바 있다.

2001년 GDC에서는 인공생명의 아이디어가 여러 곳에서 시도되고 있음을 볼 수 있다. 이전에 신경망 이론이나 유전자 이론 등 다른 새로운 인공지능 기법이 시도되었으나 성공적이지 못한 것에 비해 인공생명 기술을 사용할 수 있다는 판정을 받고 있다. Half-Life와 Unreal의 경우 flocking 이론을 사용하여 게임에 등장하는 몬스터의 움직임을 구현한 것으로 발표되고 있다. Flocking 이론은 "Age of Kings", "Homeworld"의 군중 이동(Group Movement)과 "심즈"에서 욕구의 발생과 이를 해결하는 과정에 사용되고 있다.

5. 결 론

우리는 그 동안 인간이 어떻게 지능을 표현하고 발전시키는지 또는 인간과 유사한 기계를 어떻게 만들 수 있는 지에 대해 연구를 지속시켜 왔다. 게임이라는 가상세계는 인공지능의 좋은 실험장이고

또한 연구결과를 상업화 할 수 있는 좋은 분야이다. 그 동안 우리는 게임이라면 단순한 프로그램이며 특별한 새로운 기술이 필요하지 않다고 여겨 연구의 대상으로 생각지 않았다. 그러나 현재 상업적인 게임에서 구현된 인공지능의 수준은 구현이라는 입장에서 본다면 대학이나 연구기관에서 발표되는 연구결과를 앞서고 있다. 특히 Unreal 같은 게임 엔진으로 구현된 3차원 가상세계는 연구기관에서 이를 활용하여 새로운 연구분야를 확장해 나가고 있는 실정이다.

게임이라는 산업이 구현에 비중을 두고 있기 때문에 연구결과는 논문으로 발표되기보다는 실제 상품으로 발표되는 면이 있다. 따라서 이 분야의 연구는 논문으로 발표된 내용이 많지 않으며 논문에 깊이 있는 내용이 담겨 있지 않은 경우도 있다. 앞으로 컴퓨터 관련 학계의 많은 참여가 기대된다.

(본문 내용의 일부는 2002년 대한민국 게임백서에 발표한 내용입니다)

참고문헌

[1] Steve Rabin, AI Game Programming Wisdom, Charles Rivers Media, 2002.

[2] Mark DeLoura, Game Programming Gems, Charles River Media, 2000.

[3] Mark DeLoura, Game Programming Gems 2, Charles River Media, 2001.

[4] Steven Woodcock, "Game AI: the state of the Industry", Game Developer, Aug. 2000, pp24-28, 2000.

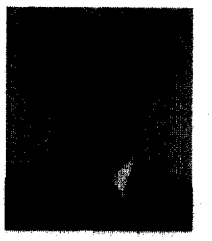
[5] Steven Woodcock, "Game AI: the state of the Industry 2000-2001", Game Developer, Aug. 2001, pp36-44, 2001.

[6] John E. Laird, "Using a Computer Game to Develop Advanced AI," IEEE Computer, July 2001, pp 70-75. 2001.

[7] Michael Lewis, Jeffrey Jacobson, "Game Engines in Scientific Research," Communications of the ACM, Jan. 2002, Vol45. No.1, 2002.

[8] Stuart Russel & Peter Norvig, Artificial Intelligence: A Modern Approach, Prentice Hall. 1995.

저자약력



이 만 재

1973년 한국과학기술원 전자계산실 연구원
1982년 스탠포드 대학 전기공학 석사
1986년 텍사스 오스틴 대학 컴퓨터 공학 박사
1986년 한국 전자통신연구소 책임연구원
1989년 솔빛미디어 대표이사
1995년 숙명여대 정보방송학과, 전산학과 교수
1998년 아주대학교 미디어학부 교수
2002년 한국정보통신대학교 공학부 교수
관심분야 : 컴퓨터 게임, 디지털 미디어
e-mail : manjai@icu..ac.kr