한국게임학회 논문지
제6권 제4호 2006년 12월

기계학습 기법을 사용한 캐릭터 제어 엔진의 설계 및 구현[1)]

이재문
한성대학교 멀티미디어공학과
jmlee@hansung.ac.kr

Design and Implementation of Engine to Control Characters By Using Machine Learning Techniques

Jae Moon Lee
Dept. of Multimedia Engineering, Hansung University

요 약

본 논문은 기계학습 기법을 이용한 게임 캐릭터를 제어하는 엔진을 설계하고 구현하는 것을 제안한다. 제안된 엔진은 실제 게임에서 상황 데이터를 추출하여 지식 데이터로 사용하므로 지능 캐릭터의 행동 패턴을 게이머들이 쉽게 인식하지 못하는 장점이 있다. 이를 위하여 상황 데이터를 추출하여 학습하는 모듈과 임의의 상황 데이터에 대하여 최적의 상황 제어를 판단하는 시험 모듈을 개발하는 것을 제안하였다. 구현된 엔진은 FEAR에 이식되고 Quake2 게임에 적용되었다. 또한 개발된 엔진의 올바른 동작과 효율성을 위하여 다양한 실험을 하였다. 실험으로부터 개발된 엔진이 올바르게 동작할 뿐만 아니라 제한된 시간 내에 효율적으로 동작함을 알 수 있었다.

ABSTRACT

This paper proposes the design and implementation of engine to control characters by using machine learning techniques. Because the proposed engine uses the context data in the rum time as the knowledge data, there is a merit which the player can not easily recognize the behavior pattern of the intelligent character. To do this, the paper proposes to develop the module which gathers and trains the context data and the module which tests to decide the optimal context control for the given context data. The developed engine is ported to FEAR and run with Quake2 and experimented for the correctness of the development and its efficiency. The experiments show that the developed engine is operated well and efficiently within the limited time.

Keyword : Game engine, AI, Datamining, Machine learning, similarity, dot product

1)본 연구는 2006학년도 한성대학교 교내연구비 지원과제임

1. 서론

컴퓨터 게임 산업이 국내외적으로 급격히 활성화되면서 게임 개발에 필요한 기술 개발이 크게 요구되고 있다[1, 2, 3].이러한 기술은 게임 개발의 생산성을 높이기 위하여 게임에 필요한 요소들로 모듈화되어 개발되고, 이러한 모듈을 엔진이라고 한다[1, 2]. 대표적인 게임 엔진으로 그래픽 엔진, 사운드 엔진, 물리 엔진 및 인공지능 엔진 등이 있다[1, 2]. 그래픽 엔진이란 게임에 필요한 그래픽을 랜더링하거나 애니메이션해 주는 엔진이고, 사운드 엔진은 소리에 대한 엔진이다. 물리 엔진이란 게임 요소에 들어가는 물리에 대한 다양한 계산을 해 주는 엔진이고, 인공지능 엔진이란 게임상에 존재하는 다양한 캐릭터에 대한 지능을 부여하는 엔진이다.

IT 강국으로 자부하는 국내 게임 산업의 경우 네트워크 게임, 모바일 게임 등 게임 산업이 크게 신장하고 있다[1]. 이러한 게임 개발 환경을 살펴보면 화려한 영상과 사운드 개발에 많은 노력을 기울어 그래픽 및 사운드 엔진의 개발에는 획기적인 발전을 가져왔다. 하지만 상대적으로 물리 엔진과 인공지능 엔진의 개발은 부진하여 현재에도 거의 초보 단계에 지나지 않는다[1]. 최근 컴퓨터 성능이 급속히 개선되고, 3D 게임 엔진 등이 개발되어 저렴한 가격에 상용화된 관계로 화려한 그래픽 위주의 게임은 거의 완성 단계라 할 수 있다. 이러한 상황에서 게이머들은 보다 자연스럽고 재미있는 게임을 요구하거나 자신과 비슷한 수준의 지능화된 NPC(Non-Player Character)를 요구하는 추세이다[3, 4]. 또한 네트워크 기술의 발전으로 네트워크를 통한 멀티 플레이 게임이 증가하면서 팀플레이 게임 등이 활성화 되어 일대일 게임보다는 다대다 게임으로 전향되고 있다[4]. 이러한 경우 팀을 형성하기 위하여 때로는 팀원으로 NPC를 사용하는 경우가 빈번해 지고 이것은 또다시 보다 지능적인 NPC를 요구하고 있다[3, 4]. 지능적인 NPC에 대한 하나의 확실한 방법이 인공지능 게임 엔진을 사용하는 것이다. 게임에서 발생하는 다양한 상황들에 대하여 인공지능의 기술을 적용하여 NPC가 임기응변적으로 대응할 수 있도록 하는 것이다.

본 논문은 기계학습 기법을 적용하여 NPC를 제어하는 엔진을 개발하는 것이다. 게임에서 요구하는 인공지능의 요소는 게임마다 요구 사항이 다르기 때문에 한 두 가지 기술적 요소를 적용하여 모든 요구 조건을 만족하는 인공지능 게임 엔진을 개발하는 것은 쉽지 않다[5]. 따라서 인공지능 게임 엔진은 가장 많이 사용되는 인공지능 기술로부터 시작하여 필요에 따라 다른 인공지능 기술이 더해져야 한다. 현재까지 대부분의 상용 게임 엔진에서 사용한 기술적 요소는 규칙기반시스템(Rule Based System), 유한상태기계(Finite State Machine)등 이다. 이것은 개발하기가 쉽고 성능이 비교적 뛰어나기 때문이다. 그러나 규칙이 고정되어 있기 때문에 시간이 지남에 따라 게이머가 그 규칙을 파악하게 되어 오히려 비 지능적인 NPC를 만들게 하는 단점이 있다[1, 3]. 본 논문에서 적용하고자하는 기법은 이러한 규칙기반시스템, 유한상태기계의 단점을 극복하는 것이다. 초기 시스템은 전문가에 의하여 정의된 규칙에 따라 반응하나 점차 게이머의 행위에 대한 데이터를 축적하고 이들을 마이닝하여 게이머의 행위에 적응하는 새로운 규칙을 생성하여 적용하자는 것이다.

2장에서는 개발될 엔진의 설계에 대하여 설명하고, 3장에서는 엔진의 구현에 대하여 설명한다. 4장에서는 개발된 엔진의 완성도 및 성능에 대하여 논하고 5장에서 결론을 논한다.

2. 기계학습 기술을 이용한 캐릭터 제어 엔진 설계

2.1 관련 연구

게임을 위한 인공지능의 연구는 다양한 분야에서 연구되어 왔다. 이러한 인공 지능 기술은 지능을 부여하는 방법에 따라 프로그램 코딩형 지능 기술, 기계학습형 지능 기술 및 진화 적응형 지능 기술이 있다[1]. 프로그램 코딩형 지능 기술은 규칙기반 지능기술, 유한상태기계 지능기술 등이 여기에 속하는 것으로 대부분의 경우 프로그램 코딩 단계에서 지능화 정도가 결정된다. 기계학습형 지능 기술은 프로그램 코딩형 지능 기술의 문제를 해결하기 위하여 개발된 것으로 실제 게임에서 일어나는 상황을 인식하고 분석하여 이러한 예제 상황으로부터 규칙을 생성하는 기법이다. 의사결정 트리나 신경망, 데이터 마이닝 기법 등이 대표적인 기술이다. 이 방법은 게임 실행 중에 동적으로 규칙을 생성하는 장점은 있으나 얻어진 결과가 항상 바람직한 규칙이라는 보장을 하기가 어렵고 모든 경우에 대비한 올바른 예

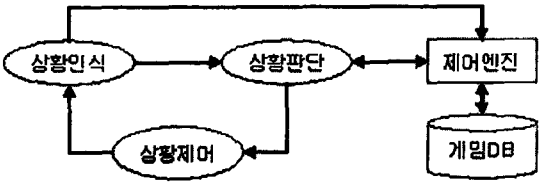
제를 준비하기가 어렵다는 단점이 있다. 진화 적응형 지능 기술은 유전자적 진화 알고리즘을 사용하는 기술이다[3, 4]. 자연계의 진화를 모방하여 선택, 교차, 돌연변이를 수행하여 문제에 적합한 해를 탐색한다. 대부분의 경우 하나의 전략을 염색체에 매핑하여 염색체를 진화함으로써 새로운 전략을 탐색할 수 있도록 하는 것이다. 본 논문에서는 기계학습형 지능 기술을 전투 게임에 적용하는 것이다.

현재 공개된 인공지능 게임 엔진 프레임워크로 FEAR(Flexible Embodied Animat aRchitecture, fear.sf.net)[6]가 있다. 본 논문에서 개발한 엔진을 FEAR 시스템에 이식하여 적용한다. 또한 Quake2 상에서 개발한 엔진의 성능을 측정할 수 있도록 하였다.

2.2 기계학습 기법을 이용한 캐릭터 제어 엔진 설계

2.2.1 제어 엔진의 구조

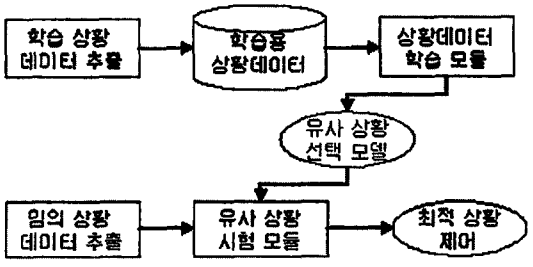
게임에서 NPC는 주변 환경에 정보를 감지하는 상황인식 기능과 주어진 상황에 대하여 의사 결정을 내리는 상황판단 기능 그리고 이러한 상황판단에 따른 행위를 제어하는 상황제어 기능을 가져야 한다. 이러한 NPC의 기능 중 의사 결정을 내리는 상황판단 기능을 제어 엔진에서 제공하는 것이 된다. 따라서 NPC는 게임 환경으로부터 주기적으로 상황을 인식하고 인식된 상황 데이터를 기반으로 제어 엔진에게 상황판단에 대한 질의를 하고 그 결과를 얻음으로써 적절히 상황에 대처하게 된다.



[그림1] NPC의 주기적 행위

그림 1은 NPC의 기능을 도식적으로 설명하고 있는 것이다. 그림 1에서 상황인식과 제어 엔진 사이의 라인은 제어 엔진이 기계학습형인 경우 상황판단을 위한 경우가 아니더라도 학습용 상황 데이터를 수집하기 위한 관계를 표시하고 있다. 즉, NPC는 주기적으로 상황인식을 하고 이로부터 얻어진 상황 데이터를 제어 엔진에 보내어 제어 엔진이 학습할 수 있도록 하여야 한다. 또한 제어 엔진은 기계학습에서 학습된 지식데이터를 저장관리 하기 위하여 게임 데이터베이스를 가지고 있다.

최근 데이터 마이닝 기술은 데이터베이스 및 인공 지능에 대한 접목 기술로 눈부신 발전을 하여 왔고 그 결과 의사 결정 시스템 등에 많이 사용되고 있다[7, 8]. 또한 이러한 데이터 마이닝 기술을 인공지능 게임 기술에 적용하려는 시도도 일어나고 있다[1]. 데이터 마이닝의 대표적 기술 중의 하나인 기계학습은 일반적으로 학습단계, 시험단계의 두 단계로 구분된다. 학습단계는 잘 정제된 학습 데이터를 이용하여 학습하는 단계를 의미하며, 시험 단계는 하나의 시험 데이터에 대하여 학습 결과를 이용하여 현재의 시험 데이터에 대한 적절한 의사 결정을 하는 것이다. 예를 들어 게임에서 NPC는 자신의 주변 환경에 대한 상황인식과 주어진 환경에서 NPC의 행위에 의하여 얻어진 결과(예를 들어 전투 게임에서 상대방의 데미지 정도 등)를 학습 데이터로 제어 엔진에 축적하고, 제어 엔진은 이렇게 축적된 데이터를 이용하여 학습하게 된다. 이렇게 학습된 이후부터 NPC는 상황판단이 필요한 경우 현재의 상황에 대한 상황 데이터를 제어 엔진에 보내게 되고, 제어 엔진은 현재의 상황 데이터와 가장 유사한 상황 데이터를 추론하여 그때의 적에 대한 예상 결과(데미지 정도 등)를 돌려주게 된다. NPC는 이러한 데이터를 받아 적절한 상황제어를 하게 되는 것이다.



[그림2] 제어 엔진의 구조도

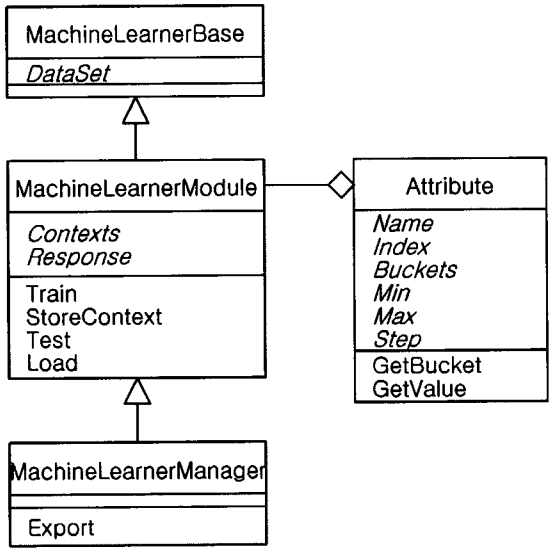
그림 2는 이러한 기계학습 기능을 갖는 제어 엔진의 구조이다. 먼저 이 엔진은 학습을 위하여 다수의 상황 데이터를 요구하는데 이것은 그림 2에서 학습 상황 데이터 추출 모듈에 의하여 수집된다. 이렇게 수집된 학습 상황 데이터를 이용하여 주어진 모델(예를 들어 kNN, NaiveBayes[7, 8] 등)에 따라 학습된다. 이러한 학습의 결과는 유사 상황 선택 모델을 구성하게 된다. 이러한 일련의 과정을 학습 단계라 한다. 다음 임의의 상황 데이터에 대하여 유사 상황 선택 모델을 이용하여 최적의 상황 제어를 판단하는 것은 그림 2에서 유사 상황 시험 모듈에 의하여 실행된다. 이러한 과정을 시험 단계라 한다. 학습 단계는 학습용 상황 데이터가 적당히 수집된 후 한번만 실행되지만, 시험 단계는 최적 상황 제어가

필요한 모든 경우 실행되어야 하므로 매우 자주 실행될 것이다. 따라서 이 단계의 효율적 구현도 매우 중요하다.

2.2.2 제어 엔진의 설계

기계학습 기술을 사용한 캐릭터 제어 엔진은 학습 데이터를 받아서 스스로 학습한 후, 이를 바탕으로 주어진 데이터에 대하여 상황 판단을 하는 것으로 본 논문의 핵심 개발 내용이다.

이러한 제어 엔진은 그림 3과 같이 MachineLearnerBase, MachineLearnerModule, MachineLearnerManager, Attribute로 나뉘어 설계되었다. 설계의 가장 큰 관점은 제어 엔진의 구조를 제공하여 향후 다양한 기계학습 방법에 의하여 쉽게 확장 가능하도록 설계하는 것이다. 그림 3에서 화살표는 객체 다이어그램에서 상속관계를 나타내며, 한쪽 끝이 마름모인 라인은 HAS-A 관계를 표시한다. 다음은 각 객체의 설계 내역이다.



[그림3]데이터 마이닝 엔진 구조의 설계

① MachineLearnerBase 객체

제어 엔진을 구성하는 최상위 객체이다. 이 객체는 제어 엔진에 외부의 데이터를 필요로 하는 경우 외부로부터 읽은 데이터를 저장관리 하는 객체이다. 예를 들어 기계학습에 의한 상황판단 기능이 개발되는 경우 게임 기획자는 게임에 대한 여러 상황들을 분류하여 이에 대한 데이터를 게임에 적용하고자 할 것이다. 이러한 데이터를 저장하는 객체이다.

② MachineLearnerModule 객체

이 객체는 제어 엔진의 핵심 객체이다. 다양한 상황 데이터를 저장하고 있으며, 이러한 상황 데이터를 적절히 분류하는 학습 기능과 학습된 지식을 바탕으로 특별한 상황에 대하여 응답하는 기능을 갖고 있다. 이 객체는 함수 멤버로 Train, Increment, Test, Load가 있다. Load는 시스템 외부로부터 주어지는 데이터를 읽어들어 이 객체를 초기화 하는데 사용된다. 외부로부터 주어지는 데이터는 학습하여야 할 상황 데이터의 구조(Attribute)와 각 구조별 퀀팅화에 대한 정보를 담고 있다. 외부 데이터는 XML로 설계 하였다. 함수 Increment는 상황 데이터 축적 인터페이스 모듈로부터 유효한 상황 데이터가 발생 시 이를 MachineLearnerModule 객체가 저장하도록 하는 함수이다. 이 함수는 두 가지 모드를 제공하여 상황 데이터가 지속적으로 축적하도록 할 수도 있으며, 일정 이상의 상황 데이터가 축적되면 신뢰도가 낮은 상황 데이터를 제거할 수 있도록 한다. 신뢰도가 낮은 상황 데이터를 선택하는 것도 쉬운 문제는 아니다. 함수 Test는 패턴탐색 인터페이스 모듈로부터 특정한 상황 데이터를 입력받아 가장 유사한 기존의 상황 데이터를 탐색하는 기능이다. 이 기능은 Train 함수의 기능과 더불어 인공지능의 수준을 나타내는 것이 된다. 데이터 멤버로 Contexts와 Response를 가지고 있다. Contexts는 상황 데이터의 구조를 관리하는 데이터 멤버이다. 즉 앞에서 설명한 Attribute별 해당 객체를 가지고 있어 Train, Test에서 퀀팅화 하는데 사용한다. Response는 상황 데이터에 대한 응답 데이터에 대한 구조를 정의하고 있으며, 역시 응답 데이터에 대한 퀀팅화에서 사용된다. 새로운 방법의 기계학습형 제어 엔진을 개발하고자 하는 경우 이 객체로 부터 상속 받으면 된다.

③ MachineLearnerManager 객체

제어 엔진을 외부 객체가 사용하고자 하는 경우 이 객체를 생성 또는 참조하여 사용한다. 이 객체는 자체적으로 함수 멤버 Export를 가지고 있는데 이 함수는 인터페이스 모듈과 제어 엔진을 연결 시켜서 인터페이스 모듈이 이 엔진을 사용할 수 있도록 하는 기능이다.

④ Attribute 객체

이 객체는 다양한 상황 데이터(Context)를 정량화하고

이를 퀀텀화 하는 객체이다. 즉 상황 데이터는 일종의 아날로그 데이터로 이를 몇 가지 가능한 값으로 퀀텀화 한다. 예를 들어 게임에서 거리에 대한 상황 데이터는 픽셀 수로 나타나지만 인공지능을 적용하는 경우에는 "가까운 거리", "중간 거리", "아주 먼 거리" 등 3가지 종류이면 충분하다. 함수 멤버 GetButket는 아날로그 값을 주면 퀀텀화된 값을 리턴하는 함수이며, GetValue는 퀀텀화된 값을 주면 그 값을 기준으로 가장 가까운 아날로그 값을 계산하는 함수이다. 이 객체는 최소(Min), 최대(Max), 간격(Step)을 입력받아 (Max-Min)/Step 단계만큼 퀀텀화 한다. Attribute 객체는 Name과 Index 두개의 데이터 멤버를 가지고 있는데 이들은 각각 대응하는 상황 데이터의 이름과 이 상황 데이터의 순서를 저장하는데 사용된다.

3. FEAR 플랫폼에서 기계학습 기법을 이용한 캐릭터 제어 엔진 구현

3.1 공개 소스 FEAR 플랫폼

하나의 독창적인 AI 게임 엔진을 개발 하였다고 할지라도 개발된 엔진을 적절한 게임에 적용하여 그 성능을 시험하는 것이 쉬운 일이 아니다. 대부분의 게임 소프트웨어의 소스가 공개되지 않은 관계로 이 일은 더욱 어렵다.

FEAR(Flexible Embodied Abimat aRchitecture)[6]는 이러한 관점에서 개발된 AI 엔진을 적용하여 시험하기에 아주 적합한 플랫폼이다. 이는 AI 게임 엔진의 개발을 위한 기본 프레임웍과 다양한 컴포넌트를 제공하는 공개 소스 소프트웨어 플랫폼이다. 따라서 FEAR를 이용하여 AI 게임 기법을 개발하고자 하는 경우 단순히 AI 모듈만 개발하여 추가함으로써 기존의 NPC에 추가된 AI 게임 기법을 적용할 수 있고, 이에 따라 게임을 실행할 수 있다. 특히 FEAR는 Quake2와 연동되어 있어 개발한 AI 게임 기법에 따라 Quake2를 동작시킬 수 있는 장점이 있다[6]. 따라서 FEAR를 이용하여 AI 게임 기법을 개발하는 경우 시험에 들어가는 많은 노력을 줄일 수 있다.

3.2 제어 엔진의 구현

본 논문에서 제안한 엔진은 FEAR 상에서 구현되었다. 구현에 대한 주요 내용은 FEAR가 Quake2와 연동하므로 Quake2에 적합한 상황 데이터를 정의하고, 정의된 상황 데이터를 축적하여 학습하는 것과 임의의 상황 데이터에 대하여 현재의 상황을 시험함으로써 가장 적절한 상황 제어를 하도록 하는 것이다.

3.2.1 상황 데이터 구현

상황 데이터는 게임 플레이어가 스크린을 통하여 게임 상황을 인식 하듯이 제어 엔진이 상황 데이터를 분석하여 현재의 게임 상황을 분석할 수 있어야 한다. 따라서 상황 데이터는 특별한 엔진에 따라 정해지는 것이 아니라 동일한 엔진에 대해서도 게임의 종류에 따라 다양한 상황 데이터가 적용될 수 있어야 한다. 본 논문에서는 이를 XML로 설계함으로써 상황 데이터를 유연하게 적용할 수 있도록 하였다. XML을 사용하는 장점은 XML은 다양한 데이터에 대하여 유연한 구조를 제공하므로 복잡한 데이터 구조일지라도 쉽게 데이터 구조를 설계할 수 있다. 또한 최근 XML 사용의 활성화로 DOM, XSLT등 XML을 이용하는 API가 다양한 형태로 제공되고 있기 때문에 XML 데이터를 이용한 프로그램이 매우 쉬운 것이 특징이다[6, 9]. 그림 4는 상황 데이터에 대한 XML의 DTD를 보이고 있다. 이것은 FEAR[6]에서 제공하는 상황 데이터 구조에 기반 하였다.

```
<!DOCTYPE knowledge [
  <!ELEMENT knowledge (MachineLearner)>
  <!ELEMENT MachineLearner (context+,response)>
  <!ELEMENT context EMPTY>
    <!ATTLIST context name CDATA #REQUIRED>
    <ATTLIST context type (real|int|bool) "real">
    <!ATTLIST context min CDATA #REQUIRED>
    <!ATTLIST context max CDATA #REQUIRED>
    <!ATTLIST context step CDATA #REQUIRED>
  <!ELEMENT response EMPTY>
    <!ATTLIST response name CDATA #REQUIRED>
    <!ATTLIST response type (real|int|bool) "real">
    <!ATTLIST response min CDATA #REQUIRED>
    <!ATTLIST response max CDATA #REQUIRED>
    <!ATTLIST response step CDATA #REQUIRED>
]>
```

[그림4]상황 데이터 구조에 대한 DTD

그림 4에서 볼 수 있듯이 이 XML의 최상위 엘리멘트는 Knowledge이며 자식으로 하나의 MachineLearner를 갖는다. MachineLearner 엘리멘트는 다수의 Context 엘리멘트와 하나의 Response 엘리멘트를 갖는다. Context 엘리멘트는 name, type, min, max, step의 속성을 갖는다. 이러한 속

성들을 근거로 제안한 엔진에서는 이들을 Attribute 객체를 만든다. Response 엘리멘트도 동일한 속성을 갖는다.

```
<?xml version="1.0" encoding="UTF-8"?>
<knowledge>
  <MachineLearner>
    <context name="distance" type="real" min="0" max="270" step="90"/>
    <context name="health" type="int" min="0" max="102" step="34"/>
    <context name="restriction" type="real" min="0" max="270" step="90"/>
    <context name="weapon" type="int" min="0" max="10" step="1"/>
    <response name="fitness" type="real" min="-100" max="100" step="25"/>
  </MachineLearner>
</knowledge>
```

[그림5]구현된 XML 데이터베이스

그림 5는 그림 4의 설계에 따라 구성된 Quake2를 위한 상황 데이터 구조이다. 이것에서 상황 데이터는 5가지의 속성으로 구성된다는 것을 알 수 있다. 여기서 distance는 적과의 거리를 나타내며 health는 현재 캐릭터의 건강 상태를 의미한다. 또한 restriction은 적을 공격하기에 불합리한 공간적 제약 사항을 수치화 하는 데이터이다. weapon은 현재의 상황에 따라 사용한 무기를 나타내고 있다. response로써 fitness는 현재의 상황에서 얻은 결과를 표기한다. 이러한 상황 데이터를 얻는 함수는 FEAR에서 제공한다. distance에서 min=0, max=270, step=90이라는 의미는 거리의 최소, 최대 값은 각각 0, 270이며, 이들은 90이라는 간격으로 3등급으로 나누어 진다는 것을 의미한다. 즉, "0~89", "90~179", "180~269" 각각은 하나의 등급에 해당하는 것으로 논리적인 의미는 "0~89"는 가까운 거리, "90~179는 보통 거리, "180~269"는 먼 거리를 나타내는 것이다.

3.2.2 상황 데이터 학습

상황 데이터 학습은 앞에서 정의된 MachineLearnerModule 객체의 Train 함수로 상황 데이터에 NPC의 행위에 대한 결과를 저장하여 효율적으로 유사 상황 판단을 할 수 있도록 데이터를 가공하는 단계이다. 본 논문에서는 Quake2에 대하여 상황 데이터로 〈distance, health, restriction, weapon〉과 fitness를 선택하였다. 이는 주어진 환경 〈distance, health, restriction〉에서 특정 무기(weapon)을 사용한 결과 fitness와 같은 결과를 얻었다는 것을 의미한다. 즉, 향후 NPC가 유사한 환경이 발생하는 경우 앞에서 사용한 무기를 동일하게 사용하는 경우에 유사한 fitness를 얻는다는 것을 의미한다. 여기서 fitness는 적에 대한 데미지(demage)로 하였다.

상황 데이터의 학습은 유사 상황을 잘 표현하도록 데이터를 가공하는 것이다. 본 논문에서는 상황 데이터를 벡터화하고 이를 단위 벡터로 가공하는 것으로 하였다. 이 경우 임의의 상황 데이터를 역시 단위 벡터화하면 두 상황 데이터 간 유사도는 kNN[7, 8] 등 기계학습 방법에서의 유사성 계산에 많이 사용되는 벡터의 내적을 사용함으로써 효율적으로 두 상황 데이터간 유사성을 계산할 수 있도록 하였다.

3.2.3 최적의 상황 판단 알고리즘

최적의 상황 판단은 설계에서 설명한 MachineLearnerModule 객체의 Test 함수이다. 구현된 알고리즘은 다양한 상황 데이터를 학습 데이터로 하여 지식을 축적하고 특정 상황 데이터에 대하여 학습 데이터에 축적된 데이터 중 가장 큰 유사도(similarity)를 가지는 상황 데이터를 탐색하여 그 상황에서의 응답 값을 기초로 현재의 상황 데이터에 대한 응답 값을 기대하는 방법이다. 예를 들어 전투 게임에서 캐릭터는 다양한 무기를 가지고 있다. 이러한 상황에서 적을 만나는 경우 캐릭터는 어떤 무기를 사용하여야 할지 순간적으로 결정하여야 한다. 대부분의 저급 인공지능을 가진 캐릭터는 가장 성능이 강력한 무기를 선택하여 적을 공격하게 하는 것이 대부분이다. 그렇지만 때로는 이러한 선택은 적은 물론 자기 자신에게도 상당한 피해를 줄 수 있다. 예를 들어 아주 근거리에 있는 적에 대하여 적당한 방해물도 없는 상태에서 수류탄을 무기로 선택하여 던졌다면 적과 마찬가지로 자신도 동일한 피해를 입게 될 것이다. 따라서 다양하고 강력한 많은 무기를 가질지라도 주어진 상황 데이터에 따라 적절한 무기를 선택하는 것이 필요하다. 본 논문에서는 이러한 경우에 캐릭터가 가지고 있는 모든 무기에 대하여 현재의 상황 데이터를 가지고 가장 유사한 과거의 상황 데이터를 탐색하고 탐색된 상황 데이터에 해당하는 응답이 가장 높은 무기를 선택한다. 그림 6은 구현된 최적 상황 판단 알고리즘이다.

```
/* 처음에는 무기가 선택되지 않음을 표시 */
maxResponse=0; selectedWeapon= NULL;

/* 상황 데이터를 추출 */
GetContext(Data);

/* 가용한 모든 무기에 대하여 주어진 상황 데이터로 */
/* 최대의 response를 주는 weapon을 선택 */
foreach weapon in (weapons){
  response= GetResponse(Data, weapon);
  if(response > maxResponse){
    maxResponse= response;
    selectedWeapon= weapon;
  }
}
```

[그림6]최선의 무기를 선택하는 방법

3.2.4 유사 상황 판단 알고리즘

그림 6에서 함수 GetResponse는 현재의 상황 데이터와 이 상황에서 무기를 사용할 때 얼마나 그 응답이 효과적일지를 계산하여 주는 함수이다. 함수 GetResponse 이하 if-문은 단순히 최대의 응답을 가지는 무기를 선택하는 과정을 나타내고 있다.

그림 7은 현재의 상황 데이터를 기반으로 예측되는 응답값을 계산하는 GetResponse에 대한 알고리즘이다. 이 알고리즘은 현재 주어진 상황 데이터와 무기를 벡터화하여 학습 데이터(TrainedDataSet)에 있는 모든 상황 데이터와 벡터의 내적을 구한 후 가장 큰 내적의 값을 가지는 상황 데이터를 선정하고 그때의 응답값(response)을 리턴하는 것이다. 여기서 두 벡터 $\vec{a}=(a_1, a_2, ..., a_n)$, $\vec{b}=(b_1, b_2, ..., b_n)$로 표현할 때 두 벡터 $\vec{a}, \vec{b}$의 내적 $(\vec{a}.\vec{b})$은 다음과 같이 계산한다.

$$\vec{a}.\vec{b}=\sum_{i=1}^{n} a_i * b_i, \quad // \text{ 두 벡터의 내적}$$

여기서 벡터 $\vec{a}, \vec{b}$는 단위 벡터로 $|\vec{a}|=|\vec{b}|=1$이다. 그림 7에서 내적 계산을 하는 코드는 중간 부분의 foreach에 해당한다.

```
response GetResponse(Data, weapon)
{
  maxSimilarity= 0;       // 최기화 maxSimilarity
  maxResponse= 0;         // 초기화 maxResponse
  Data.append(weapon); // weapon을 Data에 포함

  /* 모든 학습 데이터와 비교 */
  foreach unitData in (TrainedDataSet){
    /* 유사성 계산 similarity= a.b= Σ ai*bi */
    similarity= 0;
    foreach (ai, bi) in (Data, unitData){
      similarity += ai*bi;
    }
    /* 최대 유사성 선택 */
    if(similarity > maxSimilarity){
        similarity = maxSimilarity;
        maxResponse= unitData.response;
    }
  }
  return maxResponse;
}
```

[그림7] 가장 유사한 상황 탐색 알고리즘

TrainedDataSet					similarity	
distance	health	restriction	weapon	fitness	s_1	s_2
2(0.52)	1(0.26)	3(0.77)	1(0.26)	50	0.98	0.93
2(0.47)	1(0.24)	3(0.71)	2(0.47)	40	0.97	0.98
2(0.63)	2(0.63)	1(0.32)	1(0.32)	40	0.90	0.88
2(0.55)	2(0.55)	1(0.28)	2(0.55)	60	0.88	0.92

[표 1] 학습된 상황데이터의 예

표 1은 그림 6과 7에 대한 간단한 예이다. 표 1은 4개의 학습된 상황 데이터 〈distance, headth, restriction, weapon〉로 구성된 *TrainedDataSet* 를 보이고 있다. 〈distance, headth, restriction, weapon〉에서 괄호 밖의 값은 단위벡터를 만들기 전의 값이고 괄호안의 값은 단위벡터로 만든 후의 값이다. 이러한 상황 데이터가 있는 상태에서 캐릭터가 직면한 현재의 상황에 대한 〈distance, headth, restriction〉 값이 각각 〈2, 1, 2〉이라고 하고 이때 캐릭터가 가진 무기가 1, 2만 있다고 가정하자. 이 경우 GetRespone는 (〈2,1,2〉, 1) 그리고 (〈2,1,2〉, 2)의 매개변수로 두 번 호출된다. Data.append(weapon)을 실행하고, 이들을 단위 벡터화 하면 각각 〈0.63, 0.32, 0.63, 0.32〉, 〈0.55, 0.28, 0.55, 0.55〉로 된다. 이들에 대하여 벡터의 내적을 계산한 결과는 표 1에서 s_1, s_2와 같다. 따라서 무기 1에 대해서는 첫 번째 상황 데이터가 가장 유사하여 그 상황의 fitness인 50이 리턴 되고, 무기 2에 대해서는 두 번째 상황 데이터에 대한 fitness인 40이 리턴 된다. 이러한 결과로 캐릭터는 무기 1을 선택하게 되는 것이다.

3.2.5 상황 판단에 대한 시간 복잡도

본 절에서는 NPC가 상기 알고리즘에 따라 무기를 선택할 때 소요되는 시간 복잡도를 계산하기로 한다. w를 NPC가 소유하고 있는 무기의 수라 하고, 그림 7에서 *TrainedDataSet* 의 요소의 수를 N이라 하면 시간 복잡도 $T(w,N)$는 다음과 같이 표현된다.

$$T(w,N)= C*w*N= O(wN)$$

여기서 C는 상수이다. 그림 6에서 각각의 무기에 대하여 GetResponse를 호출하여야 하며, GetResponse는 *TrainedDataSet* 에 있는 모든 요소와 벡터의 내적 계산을 하여야 하므로 계산 비용은 무기의 수와 *TrainedDataSet* 에 있는 요소의 수에 비례하게 된다. 따라서 식 1와 같은 복

잡도를 얻을 수 있다. 식 1로 부터 상황 판단에 필요한 시간이 *TrainedDataSet* 의 요소 수에 따라 선형적으로 변할 것을 예측할 수 있다.

4. 실험 및 성능 측정

4.1 실험 환경

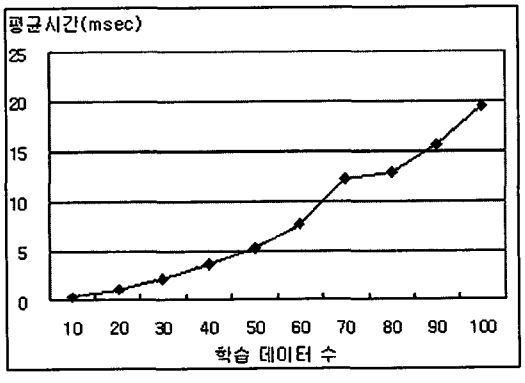
개발된 엔진이 올바르게 동작하는지와 얼마나 효율적으로 동작하는지를 측정하기 위하여 실험을 하였다. 실험을 위한 하드웨어는 펜티움 4 3.0GHZ에 1GB 메모리로 구성된 개인용 컴퓨터를 사용하였다. 소프트웨어 환경은 Quake2와 FEAR 플랫폼이 설치되었고 본 논문에서 개발한 제어 엔진이 탑재되었다.

4.2 실험 결과

Quake2와 FEAR 플랫폼은 게이머간 전투 게임을 할 수 있는 완벽한 하나의 게임 소프트웨어이다. 더하여 많은 종류의 캐릭터를 제공하며, 이러한 캐릭터에 인공지능을 더할 수 있도록 다양한 유틸리티를 제공한다. 실험은 이러한 환경에서 개발한 제어 엔진을 탑재하여 이 엔진을 가진 캐릭터의 동작에 대한 올바름과 효율성을 측정하는 것이다.

실험의 첫 번째 목적은 개발된 제어 엔진 올바르게 동작하는지에 대한 검증이다. 실험은 개발된 제어 엔진에 따라 동작하는 Quake2의 캐릭터와 게이머 사이의 실제 전투로 실행되었다. 그림 8은 인공 지능을 가진 캐릭터와 게이머 사이에 일어나는 전투에 대한 시연 장면을 보이고 있다. 그림에서 왼쪽은 개발한 제어 엔진에 따라 동작하는 캐릭터에 대한 화면이며, 오른쪽은 게이머에 의하여 제어되는 캐릭터에 대한 화면이다. 초기에 캐릭터와 게이머는 서로 다른 공간에서 출발하여 각각 상대방을 찾아 나선다. 그러한 과정에서 게임 공간상에 존재하는 다양한 무기를 취득한다. 그런 후 상대방이 일정 거리 이내 접근하면 서로 무기를 사용하여 전투를 하게 된다. 게이머의 경우에는 적을 찾아 다니면서 게이머의 판단에 따라 무기를 수시로 바꾸는 형태이나, 캐릭터의 경우 제어 엔진의 판단에 의하여 발사 직전 순간적으로 무기를 선택한다. 즉 현재의 상황 데이터를 추출하여 기존의 학습된 데이터로부터 가장 적절한 무기를 선택하도록 동작하는 것이다. 실험 결과 개발된 엔진은 구현된 알고리즘에 따라 올바르게 동작하는 것을 확인할 수 있었다. 게이머와 캐릭터 사이의 승률은 게이머의 숙련 정도에 다소 의존적이었다.

[그림8]캐릭터와 게이머의 전투에 대한 시연 장면



[그림9]상황 판단 평균 측정 시간

실험의 두 번째 목적은 제한된 시간 내에 개발된 시스템이 제대로 동작하는 가를 검증하는 것이다. 컴퓨터 게임이 비록 현실과는 많은 차이가 있으나 전투 게임과 같은 대전 게임에서는 실시간성 응답이 매우 중요하다. 따라서 제한된 시간 내에 상황 데이터로부터 지능 데이터를 추출하여 캐릭터에게 전달할 수 있어야 한다. 본 논문에서 개발한 시스템은 학습용 상황 데이터 수에 많은 의존을 가진 시스템이다. 즉, 제안한 제어 엔진에 축적된 데이터 수가 증가하면 다양한 상황 데이터에 대하여 데이터를 준비하고 있는 것이므로 보다 정확한 예측을 주는 반면, 데이터 처리를 위한 시간이 많이 필요하므로 속도가 늦어진다. 반대로 축적된 데이터 수 가 적은 경우 비록 속도는 빠를지라도 학습 데이터가 부족하므로 정확한 예측을 기대하기란 힘들어 진다. 그림 9는 학습 데이터 수에 따른 상황 판단 알고리즘(그림 6)의 실행 시간을 측정한 것이다. x축은 학습 데이터 수(그

림 7에서 *TrainedDataSet* 에서 데이터의 수)를 나타내고 y축은 5개의 무기를 가졌을 때 평균 실행 시간을 나타낸다. 그림 9에서 볼 수 있듯이 x축의 값이 증가할 수로 거의 선형적으로 실행 시간이 증가하는 것을 볼 수 있는데 이것은 식 1로부터 예측한 결과와 동일하다. 실험 결과 최대 20msec, 평균 10msec 내에 모든 인공지능에 대한 계산을 완료 한다는 것을 의미한다. 이것은 하나의 프레임을 재 생성하는데 소요되는 시간이 최소 33msec(1초/30frames)임을 고려할 때 적절한 시간이라고 판단한다.

5. 결 론

본 논문에서 데이터 마이닝의 대표적인 기법인 기계학습 기술을 사용한 게임 캐릭터 제어 엔진을 개발하는 것을 제안 하였다. 개발된 엔진은 실제 게임 상황에서 상황 데이터를 추출하여 지식 데이터로 사용하기 때문에 지능 캐릭터의 행동 패턴이 게이머들에게 노출되지 않는 장점이 있다. 이러한 관계로 캐릭터의 행동 패턴이 쉽게 노출되는 규칙 기반시스템, 유한상태기계 등의 인공지능 기술보다는 진보된 기술이라 할 수 있다. 또한 개발된 엔진은 다양한 데이터 마이닝 기법 적용을 위한 확장성 제공하도록 개발되었다. 따라서 향후 게임의 장르별 적합한 데이터 마이닝 기술이 필요한 경우 현재 개발된 시스템을 확장하여 쉽게 적용할 수 있다. 개발된 엔진은 sourceforge.net에서 제공하는 공개용 소프트웨어인 FEAR에 이식되고 Quake2 게임에 실제로 적용되어 개발된 시스템이 올바르게 동작하는지 실험하였으며 실험으로부터 올바르게 동작할 뿐만 아니라 상황 데이터가 축적됨에 따라 지능의 수준이 높아짐을 알 수 있었다.

향후 추가적인 연구로는 개발된 제어 엔진이 얼마나 효과적으로 동작하는지를 측정하여야 하고 또한 이를 더욱 개선하기 위한 방법이 무엇인지 찾아내는 것이다. 현재 부분적으로 개발된 제어 게임 엔진의 효과를 측정 중에 있다. 또한 전투 게임을 위한 보다 정밀한 상황 데이터에 대한 연구도 필요하다.

참고 문헌

[1] 우종식, 신동일, 임충재, 김풍민, "게임 엔진 제작과 기반 기술", 국가과학기술위원회, 2002

[2] Mark Deloura, "Game Programming Gems", 2001

[3] 이면섭, 조병헌, 성영락, 정성훈, 오하령, "유전자 알고리즘을 이용한 대전형 액션게임의 지능캐릭터 구현", 정보처리학회논문지B, Vol. 12, No. 3, pp 329-336, 2005.

[4] 권오광, 박종구, "유전 알고리즘과 신경망을 이용한 RPG 게임 캐릭터의 제어", 한국게임학회논문지, 제6권제2호, pp 13-21, 2006.

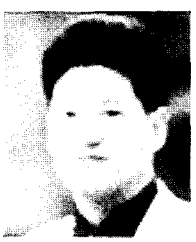
[5] 이재문, "전투 게임을 위한 기계학습형 AI 게임 엔진 개발", 하계 한국게임학회 학술발표대회, pp 193-198, 2006

[6] http://fear.sourceforge.net/docs

[7] Sebastiani F., "Machine learning in automated text categorization," ACM Computing Surveys, 34(1), pp.1-47, 2002.

[8] J.M. Lee, R. Calvo, "Scalable document classification", Intelligent Data Analysis: An International Journal, Vol. 9, No. 4, pp 365-380, 2005.

[9] 윤진현, 이재문, "SVG의 애니메이션을 이용한 슈팅 게임 설계 구현", 한국멀티미디어학회 추계학술발표대회, 2005



이 재 문 (Jae Moon Lee)

1986년 한양대학교 전자공학과 졸업(학사)
1988년 한국과학기술원 전기및전자공학과 졸업(석사)
1992년 한국과학기술원 전기및전자공학과 졸업(박사)
1994 ~ 현재 한성대학교 멀티미디어공학과 교수
관심분야 : 데이터베이스, 기계학습, 게임프로그래밍 등.