지능정보연구 제14권 제3호 2008년 9월(pp.59~76)

L-CAA : 행위 기반 강화학습 에이전트 구조

황종근
경기대학교 컴퓨터과학과
(wargen@kyonggi.ac.kr)

김인철
경기대학교 컴퓨터과학과
(kic@kyonggi.ac.kr)

본 논문에서는 실시간 동적 환경에 효과적인 L-CAA 에이전트 구조를 제안한다. L-CAA 에이전트 구조는 변화하는 환경에 대한 적응성을 높이기 위해, 선행 연구를 통해 개발된 행위기반 에이전트 구조인 CAA에 강화학습 기능을 추가하여 확장한 것이다. 안정적인 성능을 위해 L-CAA 구조에서는 행위 선택과 실행을 학습에 전적으로 의존하지 않고 학습을 보조적으로 이용한다. L-CAA에서 행위 선택 메커니즘은 크게 두 단계로 나뉜다. 첫 번째 단계에서는 사용자가 미리 정의한 각 행위의 적용 가능 조건과 효용성을 검사함으로써 행위 라이브러리로부터 실행할 행위들을 추출한다. 하지만 첫 번째 단계에서 다수의 행위가 추출되면, 두 번째 단계에서는 강화학습의 도움을 받아 이들 중에서 실행할 하나의 행위를 선택한다. 즉, 강화학습을 통해 갱신된 각 행위들의 Q 함수값을 서로 비교함으로써, 가장 큰 기대보상값을 가진 행위를 선택하여 실행한다. 또한 L-CAA에서는 실행 중인 행위의 유지 가능 조건을 지속적으로 검사하여 환경의 동적 변화로 인해 일부 조건이 만족되지 않는 경우가 발생하면 현재 행위의 실행을 즉시 종료할 수 있다. 그 뿐 아니라, L-CAA는 행위 실행 중에도 효용성이 더 높은 다른 행위가 발생하면 현재의 행위를 일시 정지하였다가 복귀하는 기능도 제공한다. 본 논문에서는 L-CAA 구조의 효과를 분석하기 위해, 대표적인 동적 가상환경인 Unreal Tournament 게임에서 자율적으로 동작하는 L-CAA 기반의 에이전트를 구현하고, 이를 이용한 성능 실험을 전개해본다.

논문접수일 : 2008년 05월 게재확정일 : 2008년 09월 교신저자 : 김인철

1. 서론

최근의 많은 에이전트 구조 연구들은 인터랙티브 컴퓨터 게임 혹은 실시간 로봇제어와 같이 예측하기 어렵고 동적 변화가 심한 환경에서의 에이전트 적응문제를 다루고 있다. 반응형 에이전트 구조(reactive agent architecture)를 발전시킨 행위기반 에이전트 구조(behavior-based architecture), 숙고형 에이전트 구조(deliberative agent architecture)와 반응형 에이전트 구조의 장점을 결합한 혼합형 에이전트 구조(hybrid agent architecture) 연구들은 복잡한 환경에서의 에이전트 적응문제를 해결해보기 위한 시도로 볼 수 있다 (Murphy, 2000). 이러한 에이전트 구조들은 에이전트 설계자가 환경을 예측하고 분석하여 그에 따른 행동을 미리 에이전트 내부에 기술해 놓는 방식을 사용한다. 미리 기술된 정책들은 설계자의 예측된 범위 내에서 동작하게 되며 에이전트가 환경 내에서 상호작용하는 동안 상황에 대한 행동정책은 변화하지 않는다. 그리고 실행 제어구조가 고정되어 있기 때문에 에이전트 설계자가 예측한 상황범위 내에서 제어 메커니즘이 동작한다. 불확실성과 변화가 심

* 본 연구는 2007학년도 경기대학교 학술연구비지원에 의하여 수행되었음.

한 환경에서 이러한 에이전트 구조들은 에이전트 설계자의 예측된 범위 내에서는 좋은 성능을 보일 수 있는 장점이 있는 반면 예측된 범위를 벗어난 상황이 발생할 경우 만족할 만한 성능을 기대하기 어렵다.

또 다른 접근 방법으로 고정된 정책을 사용하는 에이전트 구조의 제어 메커니즘과는 다른 시도로 에이전트의 적응성을 위해 오직 기계학습 기능에 의존하여 스스로 행위를 선택하고 실행하는 에이전트 구조들이 많은 연구자들에 의해 제안되었다. 이러한 방식의 에이전트 구조는 에이전트가 환경과 상호작용하는 동안 기계학습 알고리즘을 이용하여 에이전트 내부의 행동정책을 변화시키는 방식으로 동작한다. 이와 같은 학습 에이전트 구조의 경우, 기존의 에이전트 구조에 비해 다양한 행동양식을 보일 수 있고, 잘 학습된 에이전트의 경우 환경에 대한 적응성이 매우 뛰어나다. 그러나 순수 학습 에이전트 구조, 특히 강화학습 에이전트 구조의 경우, 학습에 긴 시간이 요구되고, 행위 선택과 실행이 전적으로 학습에 의존하기 때문에 학습이 충분치 않은 상태에서는 좋은 성능을 기대할 수 없다.

본 연구에서 제안하는 L-CAA 구조는 선행연구를 통해 개발된 CAA(Context-sensitive Agent Architecture) (Kim, 2005)를 확장한 에이전트 구조이다. 기존의 CAA 구조는 고정된 행동정책을 가진 행위기반 에이전트 구조이다. 즉, 설계자에 의해 CAA 에이전트의 행동 정책이 한번 결정되고 나면, 에이전트가 실행하는 동안은 학습을 통해 행동정책이 자율적으로 변경되지 않는다. 따라서 CAA 구조는 불확실성과 가변성이 높은 멀티 에이전트 환경에서처럼 사전에 미리 예측하지 못한 상황이 자주 발생할 경우, 이에 효과적으로 대처하기 어렵다는 단점이 있다. 본 논문에서 제안하는 L-CAA(Learning for CAA) 구조는 설계자의 지식과 기계 학습 알고리즘의 하나인 강화학습의 학습 결과를 조합함으로써 동적으로 변화하는 환경에서도 높은 성능을 얻을 수 있도록 하였다. CAA 구조와는 달리 L-CAA 구조에서는 사용자로 하여금 월드모델을 기초로 불리 언 상태변수들을 정의하도록 하였으며, 이 상태변수들을 이용하여 각 행위의 적용조건과 유지조건을 정의할 수 있도록 하였다. 본 논문에서는 L-CAA를 기초로 Unreal Tournament (Gerstmann, 1999) 게임 환경에서 자율적으로 동작하는 한 응용 에이전트를 구현하고, 이 응용 에이전트를 이용하여 L-CAA의 환경 적응능력을 분석하기 위한 실험을 전개한다.

2. 에이전트 구조들

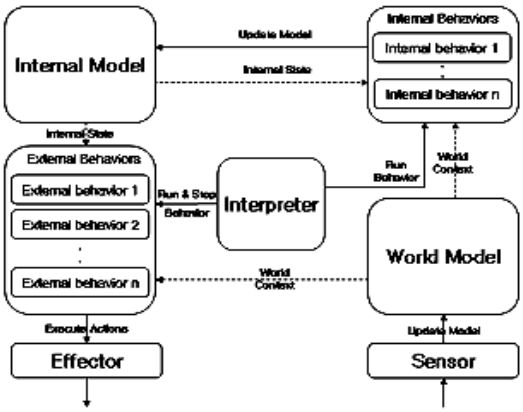
2.1 행위기반 에이전트 구조

행위기반 에이전트 구조는 지능 로봇이나 가상 에이전트 응용분야에 널리 이용되는 대표적인 에이전트 구조중의 하나이다. 행위기반 에이전트 구조는 서로 독립적인 프로세스인 행위들의 집합으로 구성되며, 에이전트의 실행제어가 행위 별로 모듈화 되어 있다. 각 행위는 인식부(SENSE)와 동작부(ACT)의 쌍으로 이루어져 있다. 따라서 외부 환경에 대한 인식이 행위 독립적이고, 전역적인 월드 모델을 중심으로 집중되어 있지 않고 각 행위 별로 특성화, 지역화 되어 있다. 또 각 행위의 동작부는 실제 환경과의 상호작용을 나타내는 단위 행동들을 포함하고 있다. 따라서 각 행위는 외부 환경의 특정한 변화만을 감지하는 자신의 인식에 따라 독립적으로 활성화되며, 이들 중 선택된 행위는 동작부에 기술된 자신만의 단위 행동들을 실행함으로써 환경을 변화시킨다. 행위기반 에

이전트 구조에서는 행위의 독립성과 병렬성이 보장되며, 행위 선택을 위해 복잡한 추론 메커니즘이 필요치 않다. 환경으로부터 입력된 센서정보에 따라 적용 가능한 각 행위들이 활성화되면, 이들에 대해 단순한 중재 메커니즘이 적용되어 수행할 행위를 빠르게 선택한다. Rodney Brooks가 제안한 포함구조(subsumption architecture)(Brooks, 1986)와 Ron Arkin이 제시한 운동 에너지장(potential field)(Arkin, 1998)은 행위기반 에이전트 구조의 좋은 예이다. 많은 행위기반 에이전트 구조에서 에이전트의 행위는 실행상황에 따라 동적으로 선택되지만, 행위 선택 방식 즉, 행동 정책(behavior policy)은 에이전트 설계 당시 설계자에 의해 한번 결정되면 실행 중에는 변경되지 않는 경우가 대부분이다. 그리고 행위기반 에이전트 구조에서 선택된 행위는 행위가 완전히 종료되기 전까지 행위가 중지되거나 다른 행위로 대체되지 않는다. 따라서 짧은 순간에 주변 정황이 복잡하게 변화하는 환경에서 수행의 길이가 긴 행위를 수행할 경우 빠르게 주변 정황에 대처 할 수 없는 문제점을 가지고 있다.

2.2 강화학습 에이전트 구조

강화 학습 에이전트 구조에서는 미리 정의된 행동정책에 기반하여 행위를 수행하지 않는다. 에이전트는 행위를 선택하여 경험해 보고 난 뒤에 얻어지는 보상을 기초로 행동정책을 학습해 나간다. 이 구조에서는 에이전트 설계자가 에이전트의 명확한 행동정책을 미리 정의해두지 않아도 에이전트가 학습을 통해서 스스로 행동정책을 알아낼 수 있기 때문에 알려지지 않은 환경에 대한 에이전트의 적응성이 매우 뛰어나다. 강화 학습 에이전트 구조로는 Dyna 구조(Sutoon, 1991), Q-Con 구조(Lin, 1993), CLARION 구조(Sun, 1997), PIQLE 구조(Decomite, 2005) 등이 있다. 그러나 일반적으로 순수 강화 학습 에이전트 구조들은 학습에 긴 시간이 요구되고, 충분히 학습되기 이전에는 현재 환경에 맞는 적응성을 가진 행동정책을 기대하기가 어렵다.

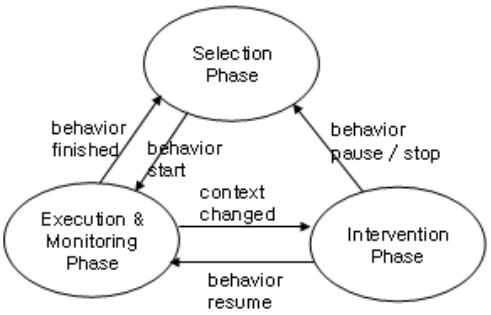


<그림 1> CAA 에이전트 구조

2.3 CAA 에이전트 구조

CAA(Kim, 2005)는 에이전트의 모든 부분을 Java 언어로 구현할 수 있는 행위기반 에이전트 구조이며, 변화하는 상황에 맞추어 효과적으로 행위들의 실행을 제어할 수 있는 기능을 제공한다. <그림 1>은 CAA 에이전트 구조를 보여주고 있다 CAA는 월드 모델(world model), 내부 모델(internal model), 내부 행위(internal behavior), 외부 행위(external behavior), 센서(sensor)와 실행기(effector), 그리고 인터프리터(interpreter) 등으로 구성된다. CAA 구조에서 월드 모델은 외부 환경의 상태를 나타내고, 반면에 내부 모델은 행위 모드나 히스토리(history), 목표와 같은 에이전트의 내부 상태를 나타낸다. CAA 구조에서 내부 행

위는 월드 모델의 환경정보와 내부 모델에 저장된 내부 상태를 참고하여 내부적인 의사결정을 수행하는 역할을 한다. 그리고 이와 같은 의사결정은 내부 모델에 저장되어 있는 현재의 목표나 행위모드의 갱신으로 표현된다. 이에 반해, 외부 행위는 실제 실행기(effector)를 통해 외부 환경을 변화시키는 행위를 말한다. 하나의 내부 행위는 내부 모델과 월드 모델을 참고하여 행위의 적용 가능성을 검사하는 적용 가능 메소드(applicable method), 행위의 효용성을 계산하는 효용성 메소드(utility method), 실행할 단위동작 들을 포함한 실행 메소드(run method) 등으로 기술되고, 외부 행위는 여기에 주기적으로 실행중인 행위의 유지 가능성을 검사하는 유지 가능 메소드(maintainable method)가 추가로 기술된다. 하나의 객체로 표현되는 각 행위는 실행 상태에 따라 생성(created), 대기(ready), 수행(run), 실패(failed), 종료(finished) 등의 서로 다른 상태들을 가진다. CAA 구조의 인터프리터는 월드 모델과 내부 모델을 참고하여 상황에 적합한 행위를 선택, 실행, 감시, 중지하는 역할을 수행한다. 이를 위해 인터프리터는 각 행위의 수행 가능 메소드를 호출하여 수행 가능성을 검사하고, 수행 가능한 행위들의 효용성 메소드를 이용하여 이들 중 효용성이 가장 높은 행위를 선택한다. 일단 하나의 행위가 선택되면, 그 행위의 실행 메소드를 호출해줌으로써 해당 행위가 수행될 수 있도록 해준다. 하나의 행위가 실행되고 있는 동안 인터프리터는 주기적으로 해당 행위의 유지 가능 메소드를 호출하여 환경 변화에 따른 행위 유지 가능성을 검사한다. 행위의 유지 가능성 검사를 통해 만약 더 이상 현재의 행위를 계속 실행하는 것이 의미가 없다고 판단되면, 실행 중인 행위를 즉시 중지하고 상황에 적합한 새로운 행위를 선택한다. 따라서 CAA 에이전트의 행동정책은 에이전트 설계자가 미리 정해주는 각 내부 행위 및 외부 행위의 적용 가능 메소드와 유지 가능 메소드에 의해 결정된다고 볼 수 있으며, 이러한 행동정책은 에이전트가 동작하는 동안 동적으로 변경되지는 않는다.



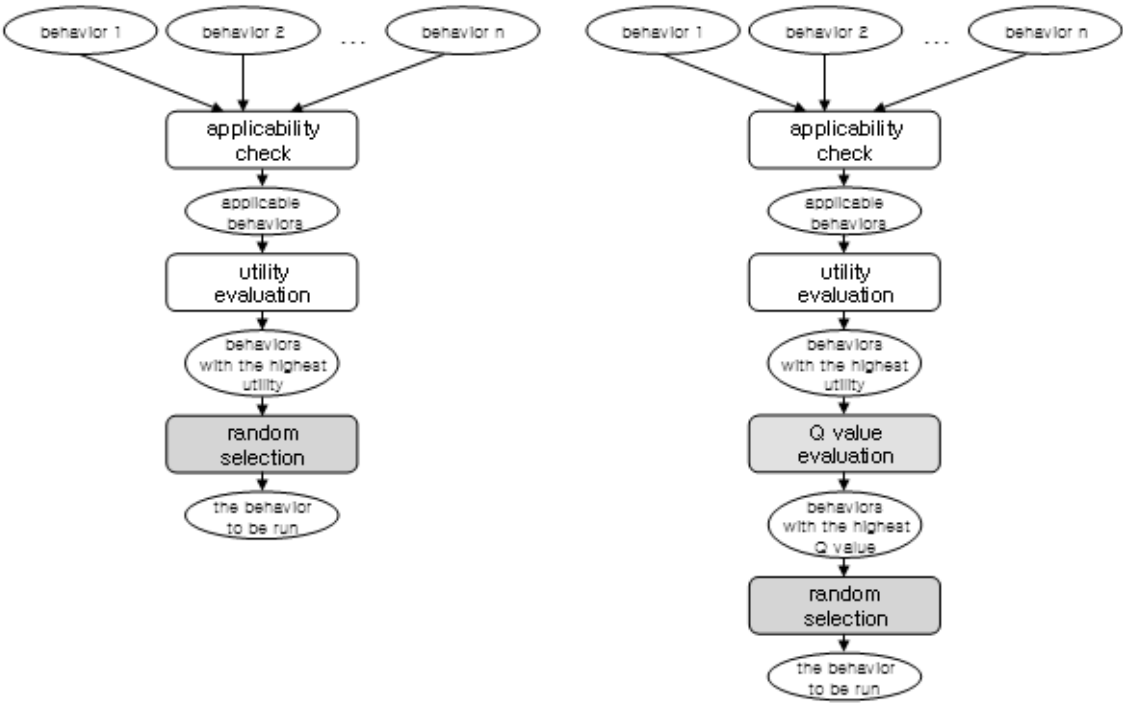
<그림 2> L-CAA의 행위 실행 주기

3. L-CAA 구조의 설계

3.1 행위 실행 주기

L-CAA는 CAA 에이전트 구조를 확장한 에이전트 구조이다. 따라서 L-CAA는 CAA와 같은 행위 기반에이전트 구조에 기초하고 있다. L-CAA는 일정한 행위 실행 주기를 통해서 외부환경과 상호작용한다. <그림 2>는 L-CAA의 행위 실행 주기를 나타내고 있다. L-CAA의 행위 실행 주기는 행위 선택 단계, 행위 실행 및 감시 단계, 행위 간섭 단계의 3단계로 구성된다. 행위 선택 단계는 외부 환경의 상황을 감지하여 수행 가능한 행위들을 가려내고 가장 효용성이 높은 행위를 선택 수행 시키는 단계이다. <그림 3>은 L-CAA와 CAA의 행위 선택 메커니즘을 비교한 것이다. <그림 3>에서 보듯이 행위 선택 단계에서 다음에 실행할 행위 선택을 위해 L-CAA 구조는 CAA 구조와 마

찬가지로 먼저 설계자의 의도대로 각 행위의 적용 가능 조건을 검사하여 실행할 행위들을 가려낸다. 그리고 만약 다수의 적용 가능 행위들이 존재할 경우에는 각 행위의 효용성(utility)을 평가하여 효용성이 가장 높은 행위를 가려낸다. 이 단계까지는 CAA의 행위 선택 메커니즘과 동일하다. 그러나 만약 행위의 효용성으로도 우열을 가릴 수는 없는 경우에는 CAA 구조는 그 중에서 임의의 행위를 하나 선택하여 수행하는 반면, L-CAA 구조는 강화 학습으로 학습된 각 행위의 가치함수인 Q값을 비교하여 Q 함수 값이 가장 우수한 행위를 선택한다. 행위 선택 단계를 통해서 하나의 행위가 선택되어 수행되면 행위 선택 단계는 행위 실행 및 감시 단계로 전환된다. 행위 실행 및 감시 단계는 외부 환경의 변화를 지속적으로 관찰하면서 현재 수행중인 행위를 계속하는 것이 적합한가를 판단하는 단계이다. 즉, 이 단계에서는 현재 수행 중인 행위의 유지 조건이 여전히 만족되는지, 효용성이 더 높은 다른 행위가 발생하였는지 등을 매우 짧은 주기로 지속적으로 검사한다. 만약 상황 변화에 의해 현재 수행중인 행위에 간섭이 필요하다고 판단되면 행위 실행 및 감시 단계에서 행위 간섭 단계로 전환된다. 행위 간섭 단계에서는 더 높은 효용성을 지닌 다른 행위의 우선적 처리를 위해 현재 수행 중인 행위에 수행 대기(pause)를 요청하거나 수행 중지(stop)를 요청한다. 기존의 CAA 구조에서는 하나의 행위가 수행되고 있는 동안에 효용성이 더 높은 다른 행위가 발생하여도 이를 우선 처



(a) CAA구주 (b) L-CAA 구조

<그림 3> 행위 선택 메커니즘의 비교

리하기 위한 메커니즘을 제공하지 못하였다. 즉, CAA 구조에서는 현재 수행중인 행위가 유지 조건을 만족하는 동안에는 이 행위의 수행을 잠시 멈추고 대기하거나 중지시킬 수 없었다. 이에 반해, L-CAA에이전트 구조에서는 실행 중인 행위의 수행 대기(pause)와 복귀(resume) 기능을 지원하여 행위 수행 방식을 향상시켰다. 일단 수행 중이던 행위가 대기상태로 전환되면 그 행위 보다 더 높은 효용성과 Q 함수 값을 갖는 행위를 선택하기 위해 다시 행위 선택 단계로 전환된다. 수행중이던 행위가 성공적으로 종료(finish)되면, 수행을 멈추고 대기 중인 행위가 있는지 검사하여 그 행위로 복귀한다.

3.2 강화학습

L-CAA 구조에서는 기존의 CAA 구조에 강화학습 기능을 새로 추가함으로써 에이전트 스스로 경험을 통해 실행 중에도 자신의 행동전략을 변경할 수 있도록 하였다. 일반적으로 강화 학습을 위한 여러 종류의 학습 알고리즘이 있지만, L-CAA 구조에서는 환경에 대한 별도의 모델을 요구하지 않으면서도 수렴성이 뛰어난 Q학습 알고리즘을 채용하였다. <그림 4>는 L-CAA 구조에 적용된 Q학습 알고리즘을 나타내고 있다. L-CAA 구조에서는 <그림 4>의 Q학습 알고리즘과 같이 ε-그리디 행동 선택 함수(ε-greedy action selection function)를 채택한다. ε-그리디 행동 선택 함수는 언제나 Q 함수 값이 가장 큰 행동만을 선택하는 그리디 행동 선택 함수에 비해, 새로운 경험을 촉진하기 위해 의도적으로 임의 동작(random action)을 선택하는 방식의 하나이다. ε-그리디 행동 선택 함수에서는 (1-ε) ($0 < \varepsilon < 1$)의 확률로는 Q 함수 값이 가장 큰 행동을 선택하지만, ε의 확률로는 임의 동작을 선택할 수 있도록 허용한다. L-CAA 구조에서는 각각의 외부 행위(external behavior)를 Q학습의 기초가 되는 단위 행동(action)으로 간주하였다. L-CAA 구조는 Q학습 알고리즘에 따라 상태 s에서 하나의 외부 행위 a를 선택하여 실행을 완료하고 나면, 그 결과로 주어지는 새로운 상태 s'과 보상 값 r을 기초로 가치함수인 Q(s, a)의 값을 새로이 갱신한다. 갱신된 Q 함수 값은 L-CAA의 내부 테이블에 저장 보관되며, 인터프리터가 새로운 행위를 실행하기 위해 행위 선택 단계를 수행할 때 참조된다.

```
Initialize Q(s, a) arbitrarily
Set ε (0 < ε < 1)
Repeat (for each episode) :
 Initialize s
 Repeat for each step of episode :
      select action a (argmax_a Q(s, a)) with
      probability 1-ε
      or random select action with probability ε
      Take the select action a
      Observe reward r, next State s'
      Update Q(s, a) :
          Q(s, a) ← (1-α)Q(s, a)+α[r+γmax_a,
        Q(s', a')-Q(s, a)]
 Update s ← s'
until s is terminal
```

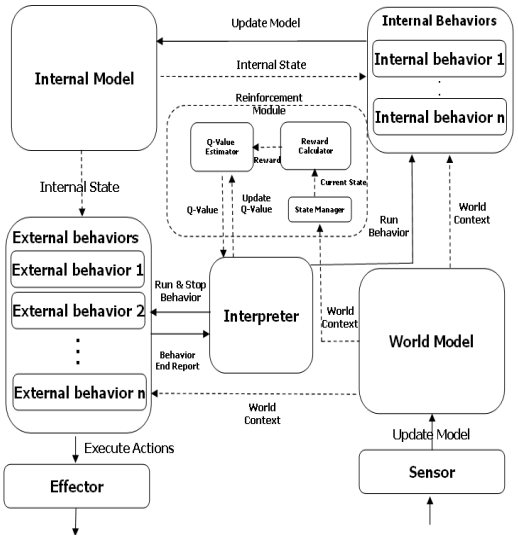
<그림 4> Q학습 알고리즘

3.3 구성요소

L-CAA 구조에서는 Q학습 기반의 적응적 행위 선택을 지원하기 위해 Q학습에 필요한 구성 요소들을 가진 강화학습 모듈을 새로 추가하였다. Q학습을 위한 상태관리기 (state manager), 보상계산기(reward calculator), 가치평가자(value estimator) 등이 새로 추가되고 인터프리터의 행동정

책 결정에 Q학습의 결과를 참고하도록 변경되었다. <그림 5>는 L-CAA 구조의 주요 구성 요소들을 보여주고 있다. L-CAA 구조에서는 에이전트 설계자가 응용 환경에 맞게 미리 상태를 결정할 수 있는 상태 변수들을 정의해준다고 가정한다. 센서 입력정보에 따라 월드 모델이 갱신되면, 상태관리기(state manager)는 다양한 객체들의 속성과 속성 값들로 표현된 월드 모델상의 정보를 기초로 각 상태 변수들의 불리 언 값을 결정함으로써 현재 상태를 판단하는 역할을 수행한다. 이와 같이 상태관리기에 의해 주어지는 명시적인 상태 정보는 L-CAA 구조 안에서 이루어지는 Q학습의 중요한 기초가 된다. 보상계산기(reward calculator)는 행위에 의해 변경된 외부 환경상태를 기준으로 보상 값을 계산하는 역할을 한다. 가치평가자(value estimator)는 보상계산기가 제공하는 각 행위에 대한 보상 값을 기초로 그 행위의 가치함수인 Q값을 갱신하고 갱신된 Q값을 저장하는 역할을 수행한다. 또 인터프리터의 요청이 있으면 실행 가능한 행위의 Q 함수 값을 인터프리터에게 제공하는 역할을 수행한다. 그러면 인터프리터는 실행 가능한 행위들의 Q 함수 값과 에이전트 설계자가 제공한 효용성 값을 참고하여 앞서 설명한 행위 선택과정을 수행한다.

한편, L-CAA 구조의 외부행위는 CAA 구조에 비해 크게 다르지 않으나 행위의 적용 가능 메소드(applicable method), 효용성 메소드(utility method), 유지 가능 메소드(maintainable method) 등이 모두 상태관리기에 의해 결정되는 명시적인 상태 표현을 참조하도록 변경되었다. 순수 학습 에이전트 구조의 낮은 성능 문제 때문에 L-CAA에서는 학습에 의해서 얻어진 학습의 결과를 기존 CAA가 가지고 있던 행동 선택 메커니즘과 혼합하여 사용한다. L-CAA의 인터프리터(interpreter)는 기



<그림 5> L-CAA 에이전트 구조

존 CAA와 마찬가지로 에이전트 설계자가 각 행위의 적용 가능 메소드와 효용성 메소드, 유지 가능 메소드를 통해 설정 해놓은 행위정책에 기초하여 매 순간 실행할 행위를 선택함으로써 시행착오의 위험 부담을 줄이면서 보다 안정적이고 높은 성능을 얻도록 하였다. 하지만 보다 적응성을 높이기 위해 경험을 통해 얻어진 Q학습 결과를 행위 선택 메커니즘의 마지막 단계에 도입하였다. 즉, L-CAA의 인터프리터는 효용성이 동일한 다수의 적용 가능 행위들이 있을 때는 이들 중 가치함수인 Q값이 가장 우수한 행위를 선택한다. 그리고 실행 중인 보다 효용성이 더 높은 행위들이 발생할 경우에도 가치평가자의 도움을 받아 각 행위들의 효용성을 계산하고 가장 효용성이 높은 행위를 선택하여 수행한다. 따라서 L-CAA의 인터프리터는 에이전트 설계자가 특정상황에서 효용성이 동일한 다수의 행위들이 경쟁하도록 설계했을 경우에는 학습의 결과를 보조적으로 이용하여 가장 효용성이 높은 행위를 선택할 수 있다.

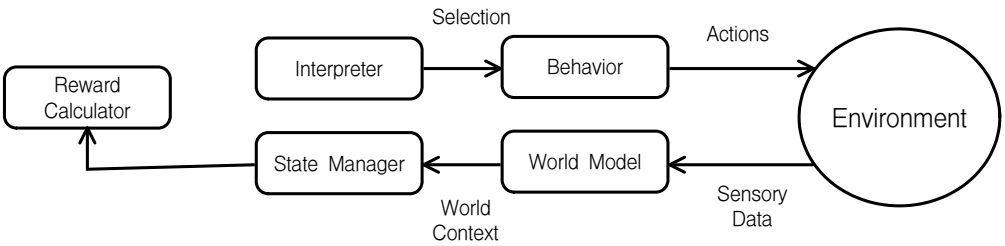
3.4 월드 모델과 상태관리기

CAA 구조에서는 환경을 구성하는 다양한 객체들의 조합으로 월드 모델의 정보를 표현하였다. 일반적으로 복잡한 실 세계 문제에 강화학습을 효과적으로 적용하기 위해서는 환경변화를 잘 표현할 수 있는 유한 개의 상태 변수들을 미리 정하고 이 상태 변수 값들의 변화에 따라 상태들을 명시적으로 나누고 표현하는 것이 필요하다. L-CAA 구조에서는 에이전트 설계자가 미리 응용 환경에 맞는 몇 개의 불리 언 상태 변수들을 정의해준다고 가정하였다. 이 상태변수들은 실시간으로 변화되는 월드 모델 정보에서 추출한 중요한 특징(feature)들을 나타낸다. 이러한 상태변수들의 집합을 이용함으로써 실 세계의 복잡한 환경상태를 간결하게 표현할 수 있다. 센서입력을 통해 월드 모델이 갱신되면 상태관리기는 정해진 상태변수들의 값을 새로이 계산함으로써 환경의 현재 상태를 결정한다. 또한 상태관리기는 언제나 특정 행위가 실행되기 이전 상태와 실행 이후 상태를 함께 저장하였다가 제공함으로써 행위 실행 이후 가치평가자의 Q 함수 값 갱신을 돕는다.

3.5 행위와 보상계산기

L-CAA 구조에서는 각 행위 객체를 구성하는 적용 가능 메소드, 효용성 메소드, 유지 가능 메소드 등이 모두 불리 언 상태변수 값의 벡터로 표현된 현재 상태를 참조하여 반환 값을 결정하도록 수정되었으며, 대기 메소드, 복귀메소드, 중지 메소드 등이 추가되었다. L-CAA 구조에서는 CAA의 행위들에 비해서 행위 자체가 가지는 자원의 관리와 사용 권한이 대폭 상향되었다. 기존 CAA 구조에서는 인터프리터의 강제적인 개입으로 행위가 종료, 수행 되었으나 이와 같은 방법은 행위의 부정확한 종료 및 자원사용의 동기화에 많은 어려움을 발생시켰다. L-CAA 구조에서는 행위의 수행과 종료가 인터프리터의 요청에 의해서 처리되도록 수정, 변경되었다. 인터프리터는 행위 선택 단계 및 행위 간섭 단계에서 행위객체 자체에 수행의 종료와 수행을 행위 스스로가 요청을 받은 후 독립적으로 자체 프로세스를 바탕으로 수행된다. 인터프리터는 요청을 전달 한 후에 행위가 안전하게 모든 자원과 선점 권을 반환 할 때까지 대기 해줌으로써 안정적으로 행위가 수행 가능하도록 변경 되었다.

L-CAA 구조에서는 행위의 상태를 실행(run), 대기(wait), 완료(complete), 중지(stop) 4가지로 수정, 변경하였다. 인터프리터는 행위 실행 및 감시 단계에서 행위의 상태를 주기적으로 관찰하며. 행위가 완료 된 시점을 기준으로 상태 관리기를 통해서



<그림 6> 보상 계산 메커니즘

```
Set previous_state = StMgr.GetCurrentState();
Set emergent_behavior = null;
Loop
      Set state = StMgr.GetCurrentState();
      Set current_behavior = FindRunningBehavior(state);
  // 현재 실행 중인 행위가 없거나 실행을 완료하였을 때
  // 보상값을 이용하여 Q 함수 값을 갱신, 적용 가능한 행위들 중 Q 함수 값과 효용성(utility)
// 의 합이 최대인 행위를 선택하여 실행
      if (current_behavior == null ||
          current_behavior.Completed()) then
             Set reward = RC.CalculateReward(state);
             QE.UpdateQValue(previous_state, current_behavior, reward);
             Set best_value = -∞
             for each behavior BehaviorList
                 if (behavior.Applicable(state)) then
                       Set q_value = QE.GetQValue(state, behavior)
                       Set utility  =  behavior.CalculateUtility(state)
                       if (best_value < (q_value + utility)) then
                             Set current_behavior = behavior;
                             Set best_value = q_value + utility;
             Set previous_state = state;
             current_behavior.Run();
  // 현재 실행 중인 행위보다 우선 순위가 더 높은 행위가 있을 때
  // 현재 실행 중인 행위를 잠시 멈추고 우선 순위가 더 높은 행위를 실행한 후 복귀
    Else if (emergent_behavior != null) then
             current_behavior.Suspend();
             Set previous_state = state;
             emergent_behavior.Run();
             while (!emergent_behavior.Completed()) wait;
             Set state = StMgr.GetCurrentState();
             Set reward = RC.CalculateReward(state);
             QE.UpdateQValue(previous_state,emergent_behavior,reward);
             Set emergent_behavior = null;
             current_behavior.Resume();
  // 현재 실행 중인 행위의 유지 가능성을 검사
  // 우선 순위가 더 높은 행위가 있는지 검사
   Else if (current_behavior.Maintainable()) then
             Set best_value = -∞
             for each behavior BehaviorList
                 if (behavior.Applicable(state)) then
                 Set q_value = QE.GetQValue(state, behavior)
                 Set utility = behavior. CalculateUtility(state);
                 if (best_value < (q_value + utility)) then
                       Set emergent_behavior = behavior;
                       break;
  // 현재 실행 중인 행위가 더 이상 실행을 유지할 수 없을 때
 // 실행을 중단
   Else
          current_behavior.Stop();
          Set current_behavior = null;
End
```

<그림 7> 인터프리터의 실행주기 의사코드

얻어진 현재 상태를 이용하여 보상메커니즘을 호출한다. <그림 6>은 보상계산기의 보상 계산 메커니즘을 표현한 것이다. 보상계산기는 행위의 결과를 기준으로 행위를 평가한다. 행위로 인해 변경된 외부 환경 상황은 센서를 통해 월드 모델에 전달되고 전달 된 환경 정보는 상태관리자에 의해 표현된 상태로 변경된다. 얻어진 상태를 기준으로 보상계산기는 상태를 평가하고 평가된 상태를 기준으로 행위를 평가한다. 행위의 결과에 대한 평가는 에이전트 설계자의 의도에 따라 재정의가 가능하도록 되어 있다. 그리고 행위가 실행 상태에 있을 경우에는 보상계산기의 행위 평가 메커니즘은 호출되지 않는다. 대기 상태는 인터프리터의 요청에 의해 행위 대기 메소드가 수행 된 뒤 현재 행위가 실행 대기상태로 전환 됐을 때의 상태를 나타낸다. 중지 상태는 행위의 유지조건이 만족하지 않았을 때 행위 객체가 수행 중지 메소드를 호출하면서 가지고 있던 자원과 선점 권을 반환하는 작업을 수행 할 때의 상태이다.

3.6 가치평가자와 인터프리터

가치평가자는 인터프리터에게 행위 선택에 이용할 각 행위의 Q 함수 값을 제공한다. 각 행위의 Q 함수값은 해당 행위의 수행 결과에 따라서 얻어지는 보상 값을 토대로 갱신된다. 가치평가자는 식 (1)에 따라 상태 s에서 실행 가능한 행위 a의 Q 값을 갱신한다. 식 (1)의 보상 값 r은 보상계산기로부터, 현재 상태 s와 다음 상태 s'은 상태관리기로부터 각각 제공되며, 학습비율 γ와 감쇠인자 α 는 각각 에이전트 설계자가 설정한 값을 이용한다.

$$Q(s, a) \leftarrow (1 - \alpha)\, Q(s, a) + \alpha\,[r + \gamma \mathrm{Max}_{a'}\, Q(s', a') - Q(s, a)] \quad (1)$$

L-CAA 구조의 인터프리터는 앞서 설명한 행위 실행 주기를 기초로 설계되었다. 인터프리터의 실행 주기는 다음과 같은 <그림 6>의 의사코드(pseudo code)로 다시 표현할 수 있다. <그림 7>에 기술된 인터프리터의 실행주기를 살펴 보면, 행위 선택 단계에서 적용 가능 메소드를 통해서 행위들을 검사하고 설계자의 의도가 반영된 효용성 메소드와 학습으로 얻어진 Q 값을 이용하여 행위를 선택하는 것을 알 수 있다. 설계자의 의도가 반영된 효용성 값이 뚜렷한 차이가 날 경우에는 효용성 값에 따라 실행할 행위가 결정되지만 그렇지 못한 경우에는 Q학습의 결과에 따라 행위가 선택된다. 그리고 현재 실행 중인 행위의 유지 가능 메소드를 주기적으로 호출하여 유지조건이 만족되지 못할 경우, 즉각적으로 그 행위의 실행을 중지한다. 하지만 현재 실행 중인 행위의 유지조건이 만족되는 경우에도 더 높은 효용성과 Q 값을 가진 행위가 발견 되면, 실행 중인 행위를 수행 대기(pause)상태로 둔 채 새로 발견된 행위를 우선적으로 선택하여 실행하는 것을 알 수 있다.

<그림 8> Unreal Tournament(UT) 게임

4. 응용 에이전트

4.1 컴퓨터 게임 환경

본 연구에서는 3차원 인터랙티브 컴퓨터 게임인 Unreal Tournament (UT) 게임(Gerstmann, 1999)에서 자율적으로 행동하는 지능형 캐릭터 에이전트, 즉 NPC(Non-Player Character)를 L-CAA 구조를 이용하여 구현해 보았다. <그림 8>은 NPC가 전투를 벌이는 UT게임의 한 장면을 보여주고 있다. 네트워크를 통해 원격으로 UT게임의 NPC를 제어하기 위해 USC와 CMU 대학에서 공동으로 개발한 Gamebots 시스템(Kaminka, 2002)을 이용하였다. Gamebots 시스템은 UT 게임을 이용해 지능형 에이전트를 구현하고 실험할 수 있도록 개발된 연구용 테스트베드이다. Gamebots 시스템은 지능형 에이전트들인 보트 클라이언트(bot client)들과 Gamebots 서버로 구성된다(Marshall, 2001). Gamebots 서버는 원격의 보트 클라이언트들에게 실시간 센서정보를 전달하고, 각 보트 클라이언트가 내린 행동명령을 받아 실행하는 역할을 한다. Gamebots 시스템은 환경의 변화, 센서정보와 행동 명령의 전달, 그리고 에이전트의 의사결정이 0.1초 단위로 이루어지는 실시간 동적 환경이며, 다수의 에이전트가 공존하며 환경에 영향을 미치는 멀티 에이전트 환경이다.

4.2 응용 에이전트의 구현

본 연구에서는 L-CAA 구조를 이용해서 Unreal Tournament Game에서 지원하는 팀 도미네이션 게임을 자율적으로 수행하는 지능형 에이전트 L-CAA UTBot을 구현하였다. 팀 도미네이션 게임은 숨겨진 세 개의 거점을 먼저 찾아내어 점령한 뒤 이것을 길게 방어하는 하는 팀이 승리하는 멀티 에이전트 게임이다. 먼저 L-CAA UTBot을 구현하기 위해 <표 1>과 같은 외부 행위를 설계하고 구현하였다. <표 1>의 행위들은 팀 도미네이션 게임에 사용되는 일반적인 행위들로 구성되어 있다. 탐색, 거점공격, 아이템 탐색, 일반적인 공격과 같은 행위들은 팀 도미네이션 게임을 수행하기에 충분하다. 한편, 효율적인 팀 도미네이션 게임을 위해서 행위들의 수행 가능 조건과 유지 조건에 사용되는 상태변수들을 설계하였다. <표 2>는 불리 언 형태로 표현된 상태변수들을 나타낸다. 설계된 상태변수들은 <표 1>의 외부 행위들의 적용 가능 조건과 유지 조건을 표현하는데 사용되었다. L-CAA 상태 관리기는 실시간으로 Gamebots 서버로부터 전송되는 메시지를 통해서 상태변수들을 갱신한다. 갱신된 상태들은 에이전트의 현재 상태를 나타내는데 사용되며 인터프리터는 현재 상태를 기준으로 행위들을 제어한다.

응용 에이전트에 적용된 학습 보상설계는 <표 3>과 같다. 본 연구에서 설계한 응용 에이전트는

<표 1> 팀 도미네이션 게임을 위한 외부행위들

행위기호	행위이름	설 명
B1	Explore	맵을 탐색
B2	Dominate	거점 공격
B3	Power_Up	아이템 탐색
B4	Offensive Attack1	접근 하면서 무기 1로 사격
B5	Offensive Attack2	접근 하면서 무기 2로 사격
B6	Defensive Attack1	멀어지면서 무기 1로 사격
B7	Defensive Attack2	멀어지면서 무기 2로 사격
B8	Standing Attack1	제자리에서 무기 1로 사격
B9	Standing Attack2	제자리에서 무기 2로 사격

전투 능력향상에 학습을 이용하기 위해서 모든 보상은 전투와 관련되게 설계하였다. 그리고 Q-학습에 필요한 상태변수는 <표 2>의 모든 상태를 사용할 경우 2^7 = 128개의 상태가 표현 가능하게 된다. 하지만 모든 상태변수가 전투 행동에 필요한 요소가 아니기 때문에 상태변수 중 장애물, 적과의 거리, 탄환의 수, 체력 4가지 상태변수들만을 이용하여 2^4 = 16개의 서로 다른 상태를 Q학습에 이용하였다.

<표 2> 불리 언 상태변수들

상태 변수	설 명
Visible_Enemy	적이 보이면 True, 아니면 False
Close_Enemy	상대방과 거리가 30이하면 True, 아니면 False
Obstacle_Between	적과 나 사이에 장애물이 있으면 True, 아니면 False
Enough_Health	체력이 50이상이면 True, 아니면 False
Enough_Ammo	탄약이 30발 이상이면 True, 아니면 False
Enough_Exploration	찾아낸 경로점이 절반 이상이면 True, 아니면 False
Enough_Dom_Point	점령중인 거점이 2개 이상이면 True, 아니면 False

<그림 9>는 구현된 L-CAA UTBot의 사용자 인터페이스를 보여준다. L-CAA UTBot의 효과적인 사용을 위해서는 강화 학습을 위한 다양한 파라미터의 설정이 필수적이다. [그림 8]의 사용자 인터페이스는 사용자가 L-CAA UTBot의 강화 학습을 위한 학습율(learning rate)과 감쇠율(discount rate)을 설정 할 수 있는 기능을 제공하고 있다. 또 사용자 인터페이스는 L-CAA UTBot의 이동 경로와 이동 행위를 실시간으로 추적할 수 있

<표 3> 평가 요소 별 보상

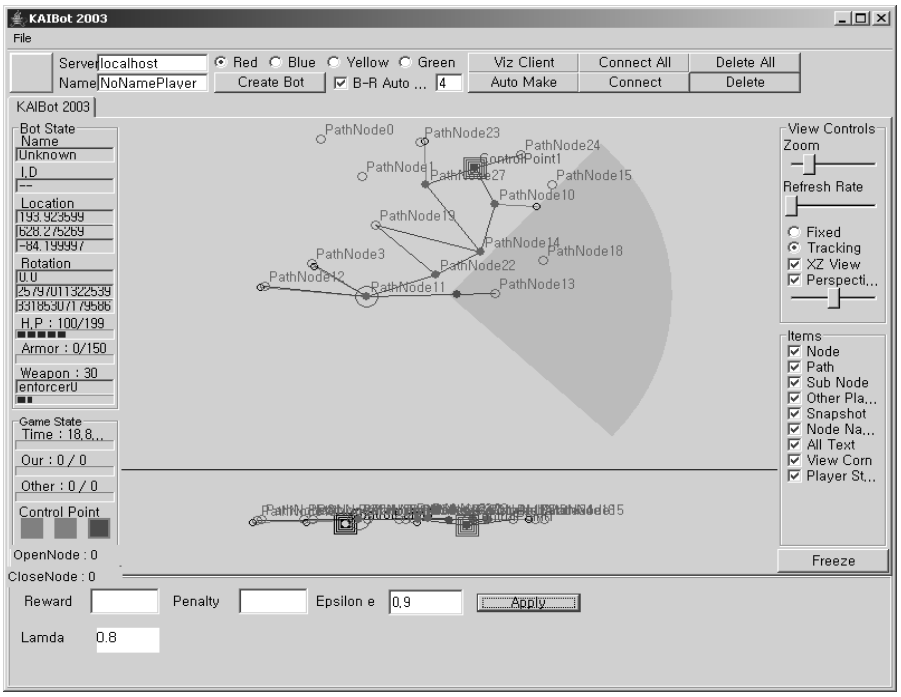
평가 요소	보상기준 1	보상기준 2	보상기준 3
체력	손상 없음 3점	50이하의 손상 1점	50점 이상 혹은 사망 0점
상대편 제압	상대편 제압 1점	없음	없음
탄환	탄환 X 0.1 씩 감점 (기본 2점)	없음	없음

고, 월드 맵(world map)과 같은 내부 상태 정보의 변경을 모니터링 할 수 있는 기능을 제공하고 있다. 또한 사용자 인터페이스는 다수의 L-CAA UTBot를 생성하여 실행시키거나 중지, 삭제시킬 수 있는 멀티 에이전트 제어 관리기 기능을 제공한다.

5. 실험

5.1 실험 목적 및 방법

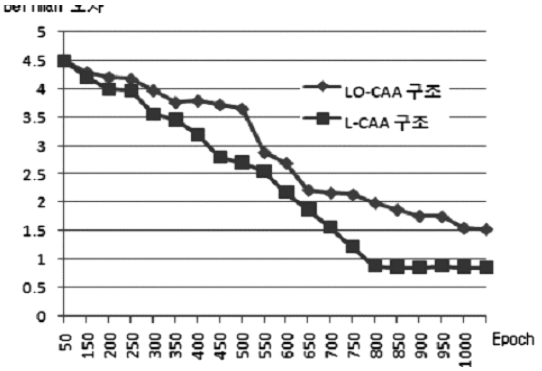
본 논문에서 제안한 L-CAA 구조와 이 구조에 기초해 개발된 응용 에이전트인 L-CAA UTBot의 학습 수렴성과 게임 수행 능력을 분석하기 위한 실험을 전개하였다. 실험을 위해 기존의 CAA 구조에 기초한 CAA UTBot팀과 오직 강화 학습에만 의존하여 행동을 결정하는 LO-CAA UTBot 팀을 별도로 구현하여 팀 도미네이션 게임상에서 비교 실험을 전개하였다. 첫 번째 실험은 L-CAA UTBot의 학습 수렴 성을 분석하는 실험이었다. 이 실험을 위해 LO-CAA UTBot와 L-CAA UTBot 2가지 에이전트를 일정 시간 동안 게임 환경에서 동작시킨 뒤, 각각 시간대비 Q 함수 값의 수렴 속도를 비교 해보았다. 1000초 동안 게임을 진행하면



<그림 9> L-CAA UTBot의 사용자 인터페이스

서 각 행위의 갱신 이전 Q 값과 갱신 이후 Q 값의 차이를 구해보는 방식으로 수렴속도를 측정하였다. 첫 번째 실험을 위해서 총 30개의 경로점(way-point)로 구성된 3차원 공간맵에서 1000초 동안 LO-CAA UTBot과 L-CAA UTBot의 팀 도미네이션 게임을 전개하며 결과를 관찰해 보았다.

두 번째 실험은 L-CAA UTBot의 팀 도미네이션 게임 수행 능력을 분석하기 위한 실험이었다. 게임의 최종 점수와 승률을 통해서 L-CAA UTBot이 다른 구조의 에이전트들에 비해 높은 게임 수행 능력을 나타내는지 관찰해보았다. 이 실험을 위해 CAA UTBot, LO-CAA UTBot, L-CAA UTBot 팀들 중에서 두 팀씩 골라 최고 점수 200점을 기준으로 각각 30회씩 총 90회의 팀 도미네이션 게임 대전을 전개 해보는 방식으로 실험을 진행하였다. 그리고 이렇게 진행된 총 90회의 실험 동안 매회 각 팀간의 승패와 최종 취득점수 차이를 조사하여 비교하였다. Q학습에 필요한 인자인 할인율 γ와 학습율 α는 각각 0.8와 0.9로 설정하였다.



<그림 10> Q 함수 값 수렴 속도 비교

5.2 실험 결과

L-CAA UTBot의 학습 수렴성 분석하기 위한 첫 번째 실험 결과, <그림 10>과 같은 Q 값이 수렴하는 추이를 얻어 낼 수 있었다. 가로축은 매 1초 단위로 행위 선택, 실행, 평가를 하는 주기를 나타내며, 세로축은 Bellman 오차를 나타내고 있다. Bellman 오차는 갱신되지 않은 가치함수 Q값과 행위로 인해 변경 된 상태에서 얻는 변경 된 가치함수 Q값의 차의 평균값이다. Bellman오차가 일정 값 이하로 수렴되면 환경에 대처하는 최적의 행동정책을 획득한 것으로 볼 수 있다. <그림 10>을 살펴보면, L-CAA UTBot의 Bellman 오차가 1.0 이하로 수렴되는 것을 알 수 있다. 그리고 LO-CAA UTBot의 수렴 속도 보다 L-CAA UTBot의 수렴 속도가 빠른 것을 알 수 있다. 그 이유는 L-CAA UTBot의 경우, 순수 강화학습 에이전트인 LO-CAA UTBot와는 달리 설계자의 제어지식을 이용하여 강화학습의 범위를 적절히 제한함으로써 짧은 시간 안에 최적의 행동정책에 수렴할 수 있었다고 판단된다.

L-CAA UTBot의 게임 수행 능력을 분석하기 위한 두 번째 실험 결과, <표 4>와 <표 5>와 같은 결과를 얻어 낼 수 있었다. <표 4>와 <표 5>를 살펴보면, 거의 모든 게임에서 혼합 행동정책을 가진 L-CAA UTBot가 승리를 거두었다는 것을 확인할 수 있다. 한편, 순수 강화학습에 의존하는 LO-CAA UTBot는 모든 게임에서 가장 낮은 점수를 획득하여 게임에서 패한 것을 알 수 있다. L-CAA UTBot와 CAA UTBot의 대전에서는 총 10회 중 절반이 넘는 약 6회에 걸쳐 L-CAA UTBot가 CAA UTBot를 이긴 것을 확인 할 수 있다. L-CAA UTBot가 다른 두 종류의 에이전트에 비해 높은 성능을 보여준 것은 다음과 같은 이유 때문으로 파악된다. 첫 번째는 L-CAA UTBot가 CAA UTBot와는 달리 학습에 의한 환경 적응성을 가지고 있기 때문이고, 두 번째는 LO-CAA UTBot와는 달리 설계자의 제어 지식을 이용할 수 있기 때문이다.

<표 4> 팀별 대전 총점 비교

대전 / 횟수	L-CAA : CAA	L-CAA : LO-CAA	CAA : LO-CAA
1회	200 : 173	200 : 65	200 : 92
2회	200 : 163	200 : 52	200 : 77
3회	200 : 190	200 : 68	200 : 83
4회	182 : 200	200 : 67	200 : 102
5회	200 : 153	200 : 93	200 : 53
6회	200 : 144	200 : 72	200 : 67
7회	191 : 200	200 : 95	200 : 92
8회	200 : 150	200 : 101	200 : 133
9회	181 : 200	200 : 43	200 : 71
10회	168 : 200	200 : 56	200 : 73

<표 5> 팀별 대전 승률 비교

대전 / 횟수	L-CAA : CAA	L-CAA : LO-CAA	CAA : LO-CAA
10회	6 : 4	10 : 0	10 : 0
20회	14 : 6	20 : 0	20 : 0
30회	21 : 9	30 : 0	30 : 0

6. 결론

본 논문에서는 기존의 CAA 구조를 확장하여 강화 학습을 지원하는 행위 기반 에이전트 구조를 제안하였다. L-CAA 구조는 설계자가 제공하는

에이전트의 행동정책과 강화학습을 통해 에이전트 스스로 습득한 행동정책을 함께 이용 할 수 있는 혼합 구조로서, 높은 성능과 적응성을 갖춘 에이전트 구조이다. 모든 행위의 선택을 학습을 통해서 결정하는 순수 학습 에이전트 구조의 경우 학습초기에 낮은 성능을 보이는 문제를 가지고 있으나, L-CAA 구조에서는 설계자가 제공하는 행위 제어 지식을 이용 할 수 있으므로 학습이 충분치 못한 상태에서도 높은 성능을 보장 할 수 있다. 본 논문에서는 UT 게임 환경에서 실시한 비교 실험을 통해 L-CAA 구조와 이에 기초한 응용 에이전트인 L-CAA UTBot의 높은 행위 성능과 학습 수렴성을 확인할 수 있었다. 한편, L-CAA 구조는 CAA 구조에 비해 행위 감시 능력 개선 및 학습 기능이 추가되었으나, CAA 구조에 비해 많은 계산 양을 필요로 한다는 점을 알 수 있었다. 따라서 향후 알고리즘의 개선을 통해 계산 양을 줄이는 연구가 필요하다고 판단한다.

참고문헌

Arkin R. C., Behavior-Based Robotics, MIT Press, 1998.

Brooks, R. A., "A Robust Layered Control System for A Mobile Robot", *IEEE Journal of Robotics and Automation*, Vol.2, No.1(1986), 14~23.

Carbonell, J., O. Etzioni, Y. Gil, R. Joseph, Knoblock, C., Minton, S., and Veloso M., "Prodigy : An Integrated Architecture for Planning and Learning", *ACM SIGART Bulletin*, Vol.2, No.4(1991), 51~55.

Decomite, F., S. Delepoulle, "PIQLE : A Platform for Implementation of Q-Learning Experiments", Proceeding of the NIPS 2005 workshop on Reinforcement Learning Benchmarks and Bake-offs II, (2005), 21~25.

Gerstmann, J., "Unreal Tournament : Action Game of the Year", GameSpot, (1999), http://www.gamespot.com/features/1999/p3_01a.html.

Kaminka, G. A., et al., "GameBots : A Flexible Test Bed for Multi-Agent Team Research", *Communications of ACM*, Vol.45, No.1(2002), 43~45.

Kim, I. C., "CAA : A Context-Sensitive Agent Architecture for Dynamic Virtual Environments", Lecture Notes in Artificial Intelligence (LNAI), Vol.3661(2005), 46~151.

Kuzmin, V., "Connectionist Q-Learning in Robot Control Task", Scientific Proceeding of Riga Technical University 5. Serija. Datorzinatne, Information Technology and Management Science, Octorber, (2002).

Lin L. J., "Reinforcement Learning for Robots Using Neural Networks", Ph.D. thesis, Carnegie Mellon University, Pittsburgh, CMU-CS-93-103, (1993).

Marshall, A. R., A. N. Scholer, A. S. Tejada, G. A. Kaminka, S. Schaffer, and C. Sollitto, "GameBots : A 3D Virtual World Test Bed for Multi-Agent Research", Proceedings of the Second International Workshop on Infrastructure for Agents, MAS, and Scable MAS, (2001).

Mataric, M. J., "Behavior-Based Robotics", the MIT Encyclopedia of Cognitive Sciences, Robert A. Wilson and Frank C. Keil, eds., MIT Press, (1999), 74~77.

Murphy, R. R., Introduction to AI Robotics, MIT Press, 2000.

Sun, R., “An Agent Architecture for On-line Learning of Procedural and Declarative Knowledge”, Proceedings of ICONIP 1997, Springer-Verlag, (1997), 766~769.

Sutton, R. S., “Dyna, An Integrated Architecture for Learning, Planning, and Reacting”, *ACM SIGART Bulletin*, Vol.2, No.4(1991), 160~163.

Sutton, R. S., Barto, A. G., Introduction to Reinforcement Learning, MIT Press, Cambridge, MA, 1998.

Watkins, J. C. H. and P. Dayan, “Q-Learning, Machine Learning”, Vol.8, No.3-4(1992), 279~292.

Weiss G., Multiagent Systems, MIT Press, Berlin Heidelberg, 1999.

Abstract

L-CAA : An Architecture for Behavior-Based Reinforcement Learning

Jonggeun Hwang* · Incheol Kim*

In this paper, we propose an agent architecture called L-CAA that is quite effective in real-time dynamic environments. L-CAA is an extension of CAA, the behavior-based agent architecture which was also developed by our research group. In order to improve adaptability to the changing environment, it is extended by adding reinforcement learning capability. To obtain stable performance, however, behavior selection and execution in the L-CAA architecture do not entirely rely on learning. In L-CAA, learning is utilized merely as a complimentary means for behavior selection and execution. Behavior selection mechanism in this architecture consists of two phases. In the first phase, the behaviors are extracted from the behavior library by checking the user-defined applicable conditions and utility of each behavior. If multiple behaviors are extracted in the first phase, the single behavior is selected to execute in the help of reinforcement learning in the second phase. That is, the behavior with the highest expected reward is selected by comparing Q values of individual behaviors updated through reinforcement learning. L-CAA can monitor the maintainable conditions of the executing behavior and stop immediately the behavior when some of the conditions fail due to dynamic change of the environment. Additionally, L-CAA can suspend and then resume the current behavior whenever it encounters a higher utility behavior. In order to analyze effectiveness of the L-CAA architecture, we implement an L-CAA-enabled agent autonomously playing in an Unreal Tournament game that is a well-known dynamic virtual environment, and then conduct several experiments using it.

Key Words : Agent Architecture, Reinforcement Learning, Q Function Value, Behavior-Based Architecture, UTBot

* Dept. of Computer Science, Kyonggi University

저 자 소 개

황종근

2006년에 경기대학교 컴퓨터과학과에서 학사를, 2008년에 동대학원에서 석사학위를 취득하였다. 관심분야는 지능형 에이전트, 기계학습, 게임인공지능 등이다.

김인철

1987년에 서울대학교 계산통계학과에서 석사학위를, 1995년에 동대학원에서 박사학위를 취득하였다. 1996년부터 현재까지 경기대학교 컴퓨터과학과 교수로 재직중이며, 주요 관심분야는 지능형 에이전트, 자동계획 및 기계학습 등이다.