행위 기반 강화 학습 에이전트 구조
An Agent Architecture for Behavior-Based Reinforcement Learning

황종근[a] 김인철[b]

[a,b]경기대학교 정보과학부
경기도 수원시 장안구 이의동 산 94-6
Tel: +82-31-243-9670, E-mail:{wargen[a], kic[b]}@kyonggi.ac.kr

요약

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 행위 기반 에이전트 구조인 CAA에 강화 학습 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 수행 가능 조건과 효용성을 검사함으로써 행위 라이브러리로부터 실행할 행위들을 추출한다. 하지만 첫 번째 단계에서 다수의 행위가 추출되면, 두 번째 단계에서는 강화 학습의 도움을 받아 이들 중에서 실행할 하나의 행위를 선택한다. 즉, 강화 학습을 통해 갱신된 각 행위들의 Q 함수 값을 서로 비교함으로써, 가장 큰 기대 보상 값을 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 UTBot 들을 구현하고, 이들을 이용하여 성능 실험을 전개해본다.

Keywords:
Agent Architecture, Reinforcement Learning, Q Value, Behavior-Based Architecture, UT Game

1. 서론

최근의 많은 에이전트 구조 연구들은 인터액티브 컴퓨터 게임 혹은 실시간 로봇 제어와 같이 예측하기 어렵고 동적 변화가 심한 환경에서의 에이전트 적응 문제를 다루고 있다. 반응형 에이전트 구조(reactive agent architecture)를 발전시킨 행위 기반 에이전트 구조(behavior-based architecture)[11], 숙고형 에이전트 구조(deliberative agent architecture)[6]와 반응형 에이전트 구조의 장점을 결합한 혼합형 에이전트 구조(hybrid agent architecture)[5] 연구들은 복잡한 환경에서의 에이전트 적응 문제를 해결해보기 위한 시도로 볼 수 있다. 일반적인 행위 기반 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 행동 정책은 변화하지 않는다. 그리고 실행 제어구조가 고정되어 있기 때문에 에이전트 설계자가 예측한 상황범위 내에서 제어 메커니즘이 동작한다. 불확실성과 변화가 심한 환경에서 행위 기반 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 에이전트의 실세계 적응성을 위해 오직 기계학습 기능에 의존하여 스스로 행위를 선택하고 실행하는 에이전트 구조들이 많은 연구자들에 의해 제안되었다. 이러한 방식의 에이전트 구조는 에이전트가 환경 내에서 상호작용하는 동안 기계학습 알고리즘을 이용하여 에이전트 내부의 행동 정책을 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 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 선행 연구를 통해 개발된 CAA[8]를 확장한 에이전트 구조이다. 기존의 CAA 구조는 행위 수행 조건이 변경되지 않는 행위 기반 에이전트 구조이다. 따라서 설계자에 의해 에이전트의 행동 정책이 한번 결정되고 나면 에이전트가 실행하는 동안 이것이 변경되지 않으며 학습을 통한 행동 정책 변경을 지원하지 않는다. 따라서 불확실한 실시간 환경에서 CAA 구조는 에이전트에게 미리 정의되지 않은 상황에 직면 했을 경우 이에 따르는 문제를 해결 할 수가 없는 단점이 있다.

본 논문에서 제안하는 L-CAA 구조는 설계자의 지식과 기계 학습 알고리즘의 하나인 강화 학습의 학습 결과를 조합하여 실시간 환경에 대처 하고자 하였다. 일반적으로 예측 가능한 상황에서는 설계자의 지식을 기반으로 행위를 선택, 수행, 중지하며 긴급한 상황 혹은 예측하기 어려운 상황이 발생 했을 경우 미리 정의 된 몇 가지 행위를 수행한 뒤 행위의 성공과 실패에 따른 보상을 계산하고 이것을 기초로 Q 학습[4]을 전개한다. Q 학습을 위해 L-CAA에서는 사용자가 정의한 상태변수를 중심으로 월드모델의 다양한 정보를 하나의 상태로 표현한다. 그리고 이와 같은 상태표현을 이용하여 각 행위의 상황에 따른 수행 가능 조건과 유지조건을 정의할 수 있도록 하였다. 본 논문에서는 L-CAA를 기초로 Gamebot[1] 환경에서 자율적으로 동작하는 한 응용 에이전트를 구현하고 L-CAA의 환경 적응 능력을 분석하기 위한 실험을 전개해본다.

2. 에이전트 구조들

2.1 행위 기반 에이전트 구조

행위 기반 에이전트 구조는 지능 로봇이나 가상 에이전트 응용분야에 널리 이용되는 대표적인 에이전트 구조중의 하나이다. 행위 기반 에이전트 구조는 서로 독립적인 프로세스인 행위들의 집합으로 구성되며, 에이전트의 실행제어가 행위별로 모듈화 되어 있다. 각 행위는 인식부와 동작부(sense, act)의 쌍으로 이루어져 있다. 따라서 외부 환경에 대한 인식과 표현이 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.

행위 기반 에이전트 구조에서는 행위의 독립성과 병렬성이 보장되며, 행위 선택을 위해 복잡한 추론 메커니즘이 필요치 않다. 환경으로부터 입력된 센서정보에 따라 수행 가능한 각 행위들이 활성화되면, 이들에 대해 단순한 중재 메커니즘이 적용되어 수행할 행위를 빠르게 선택한다. Rodney Brooks가 제안한 포함구조(subsumption architecture)[2]와 Ron Arkin이 제시한 운동에너지장(potential field)은 행위 기반 에이전트 구조의 좋은 예이다.

많은 행위 기반 에이전트 구조에서 에이전트의 행위는 실행상황에 따라 동적으로 선택되지만, 행위 선택 방식 즉, 행동 정책(policy)은 에이전트 설계 당시 설계자에 의해 한번 결정되면 실행 중에는 변경되지 않는 경우가 대부분이다.

2.2 순수 강화 학습 에이전트 구조

강화 학습 에이전트 구조에서는 미리 정의된 행동 정책에 기반을 두어 행위를 수행하지 않는다. 에이전트는 행위를 선택하여 경험해 보고 난 뒤에 얻어지는 보상을 기초로 행동 정책을 학습해 나간다. 이 구조에서는 에이전트 설계자가 에이전트의 명확한 행동 정책을 미리 정의해두지 않아도 에이전트가 학습을 통해서 스스로 행동 정책을 알아낼 수 있기 때문에 알려지지 않은 환경에 대한 에이전트의 적응성이 매우 뛰어나다. 강화 학습 에이전트 구조로는 CLARION[14] 구조, Q-Con 구조[15] 등이 있다. 그러나 일반적으로 순수 강화 학습 에이전트 구조들은 학습에 긴 시간이 요구되고, 충분히 학습되기 이전에는 현재 환경에 맞는 적응성을 가진 행동 정책을 기대하기가 어렵다.

2.3 CAA 에이전트 구조

CAA[8]는 에이전트의 모든 부분을 Java 언어로 구현할 수 있는 행위 기반 에이전트 구조이며, 변화하는 상황에 맞추어 효과적으로 행위들의 실행을 제어할 수 있는 기능을 제공한다. 그림 1은 CAA구조를 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 행위모드나 히스토리(history), 목표와 같은 에이전트의 내부 상태를 나타낸다.

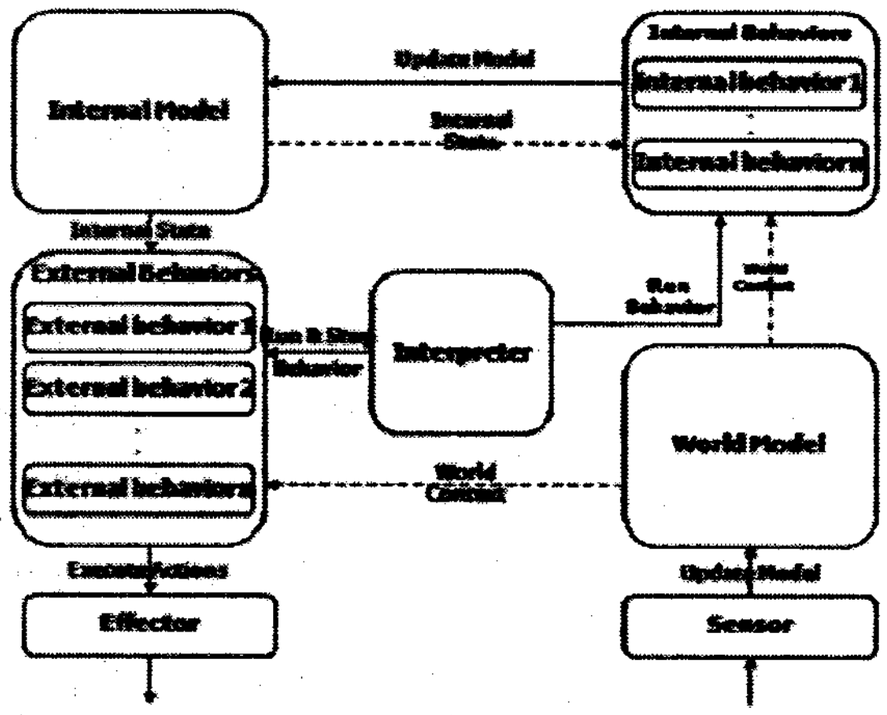

그림 1 - CAA 구조

CAA구조에서 내부 행위는 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 수행 가능성을 검사하는 수행 가능 메쏘드(applicable method), 행위의 효용성을 계산하는 효용성 메쏘드(utility method), 실행할 단위동작들을 포함한 실행 메쏘드(run method) 등으로 기술되고, 외부 행위는 여기에 주기적으로 실행중인 행위의 유지 가능성을 검사하는 유지 가능 메쏘드(maintainable method)가 추가로 기술된다. 하나의 객체로 표현되는 각 행위는 실행 상태에 따라 생성(create), 대기(wait), 실행(run), 중지(stop), 종료(finish) 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 메쏘드를 호출하여 수행 가능한 행위들을 검사하고, 검사된 행위들의 효용성 메쏘드를 이용하여 이들 중 효용성이 가장 높은 행위를 선택한다. 일단 하나의 행위가 선택되면, 그 행위의 실행 메쏘드를 호출해줌으로써 해당 행위가 수행될 수 있도록 해준다. 하나의 행위가 실행되고 있는 동안 인터프리터는 주기적으로 해당 행위의 유지 가능 메쏘드를 호출하여 환경 변화에 따른 행위 유지 가능성을 검사한다. 행위 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

3 에이전트 구조의 설계

3.1 에이전트의 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. 그림2는 L-CAA의 실행주기를 표현한 것이다.

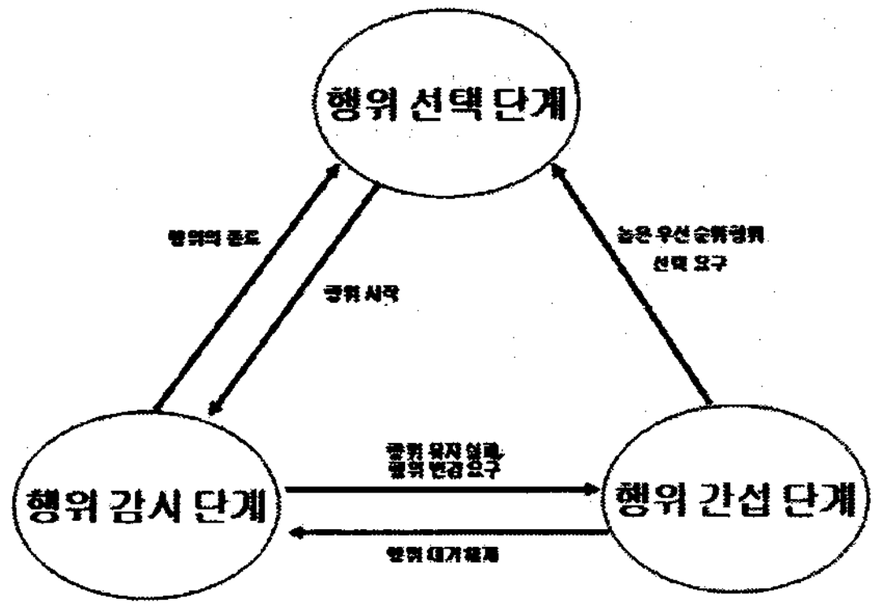

그림 2 - 행위 실행 주기

L-CAA의 행위 실행 주기는 행위 선택 단계, 행위 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 3은 L-CAA와 CAA의 행위 선택 메커니즘을 비교한 것이다.

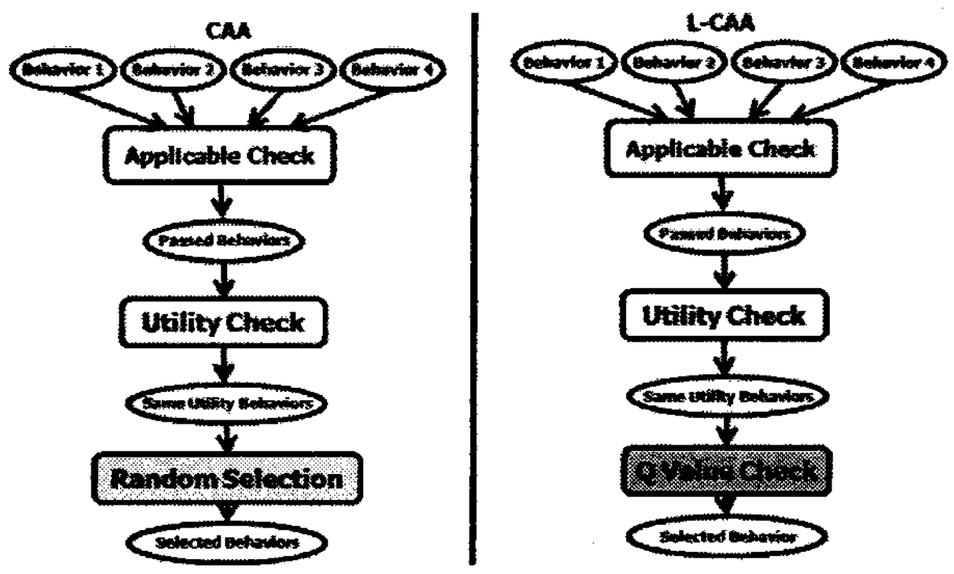

그림 3 - 행위 선택 메커니즘

그림 3에 따르면 L-CAA는 행위 선택 단계에서 행위의 수행 가능 조건을 검사하여 행위들을 가려낸다. 그리고 복수의 수행가능 행위들 중에서 에이전트 설계자의 의도가 반영된 행위 효용성을 참고한다. 이 단계까지는 CAA의 행위 선택 메커니즘과 동일하다. 그러나 설계자의 행위 효용성에서도 우열을 가릴 수가 없게 될 경우 CAA는 임의의 행위를 하나 선택하여 수행하는 반면 L-CAA는 강화 학습으로 학습된 각 행위의

가치함수인 Q값을 참고하여 행위를 선택한다. 행위 선택 단계를 통해서 행위가 선택, 수행되면 행위 선택 단계는 행위 감시 단계로 전환된다.

행위 감시 단계는 외부 환경의 변화를 지속적으로 관찰하면서 현재 수행중인 행위의 수행을 계속하는 것이 적합한가를 관찰하는 단계이다. 이 단계에서는 수행 중인 행위의 유지조건과 효용성이 더 높은 다른 행위가 발생하였는지 검사한다. 행위 감시 단계에서 현재 수행중인 행위에 간섭이 필요하다고 판단되면 행위 감시 단계는 행위 간섭 단계로 전환된다.

행위 간섭 단계는 수행 중인 행위객체에 수행 대기 요청을 하거나 수행 종료를 요청하는 역할을 수행한다. CAA 에이전트 구조에서는 하나의 행위가 수행되고 있는 동안에 효용성이 높은 다른 행위가 발생하였을 때 기존 수행중인 행위를 대기 상태로 변경 할 수 없었으며, 행위를 강제로 중지시키는 방법을 사용했어야 했다. L-CAA에이전트 구조에서는 대기 상태를 지원하여 행위 수행방식을 향상시켰다. 행위 객체가 대기상태로 전환되면 대기 행위 보다 더 우선순위가 높은 행위를 선택하기 위해 다시 행위 선택 단계로 전환된다. 선택된 효용성이 높은 행위가 종료되면 즉시 자원과 우선권을 반납하고 행위 간섭 단계로 전환 되어 대기 중인 행위 객체를 수행 상태로 전환 한다. L-CAA의 인터프리터는 그림 2의 행위 실행주기를 기초로 설계 되었다.

3.2 강화 학습

본 논문에서는 CAA 구조와 강화 학습알고리즘을 이용하여 에이전트 구조를 개선해보고자 하였다. 교사학습과 같은 학습 알고리즘의 경우 이미지 인식과 산술연산과 같은 명확한 정답이 있는 경우에 매우 뛰어난 성능을 보인다. 하지만 상황에 맞는 행위를 선택하는 문제의 경우 정확한 훈련예제를 발견하기 어렵고, 설계자가 최적의 정답을 줄 수 없다. 따라서 행동에 대한 평가와 보상을 통해서 스스로 환경에 대한 지식정보를 변경해 가는 강화 학습 알고리즘을 채택하였다. 일반적으로 강화 학습 알고리즘에는 여러 종류의 알고리즘이 있지만, L-CAA 구조에서는 환경에 대한 별도의 모델을 요구하지 않으면서도 수렴성이 뛰어난 Q학습 알고리즘을 채용하였다. 그림 4는 □-greedy를 Q학습 알고리즘[4]을 나타내고 있다.

L-CAA구조에서는 Q학습의 단위 행동을 에이전트의 각 외부 행위에 대응시켰다. 에이전트 행위 제어구조의 일부 구성요소들은 Q학습 알고리즘을 기초로 구현 되었으며, 행위 수행이 종료 된 뒤에 인터프리터의 제어 메커니즘을 통해 매번 Q값 변경 함수를 호출하도록 설계 하였다.

```
Initialize Q(s, a) arbitrarily
Set ε (0 < ε < 1)
Repeat for each episode:
  Initialize s
  Repeat for each step of episode :
    Select the action a = argmax_a Q(s , a) with the probability 1 − ε
    or random select action with the probability ε
    Take the selected action a
    Observe the reward r and the 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 ← s'
  until s is terminal
```

그림 4 - □-greedy Q 학습

그리고 행위의 선택은 □-greedy 전략을 기본으로 설계된 인터프리터에 의해 결정된다. L-CAA의 인터프리터는 □-greedy 전략을 통해서 CAA 구조의 행위 효용성 계산 방식에서 벗어나 학습의 결과로 동일한 행위 효용성을 가진 복수의 행위들 중 학습의 결과인 Q값이 가장 높은 행위를 선택하도록 설계하여 상황에 따라 적합한 행위를 수행 할 수 있도록 하였다.

3.3 에이전트 구성 요소

L-CAA구조에서는 Q학습 기반의 적응적 행위 선택을 지원하기 위해 Q학습에 필요한 구성 요소들을 가진 강화 학습 모듈들을 새로 추가하였다. Q학습을 위한 상태 관리기 (state manager), 보상계산기(reward calculator), 가치 평가자(value estimator) 등이 새로 추가되고 인터프리터의 행동 정책 결정에 Q학습의 결과를 참고하도록 변경되었다. 그림 5는 L-CAA 구조의 주요 구성 요소들을 보여주고 있다.

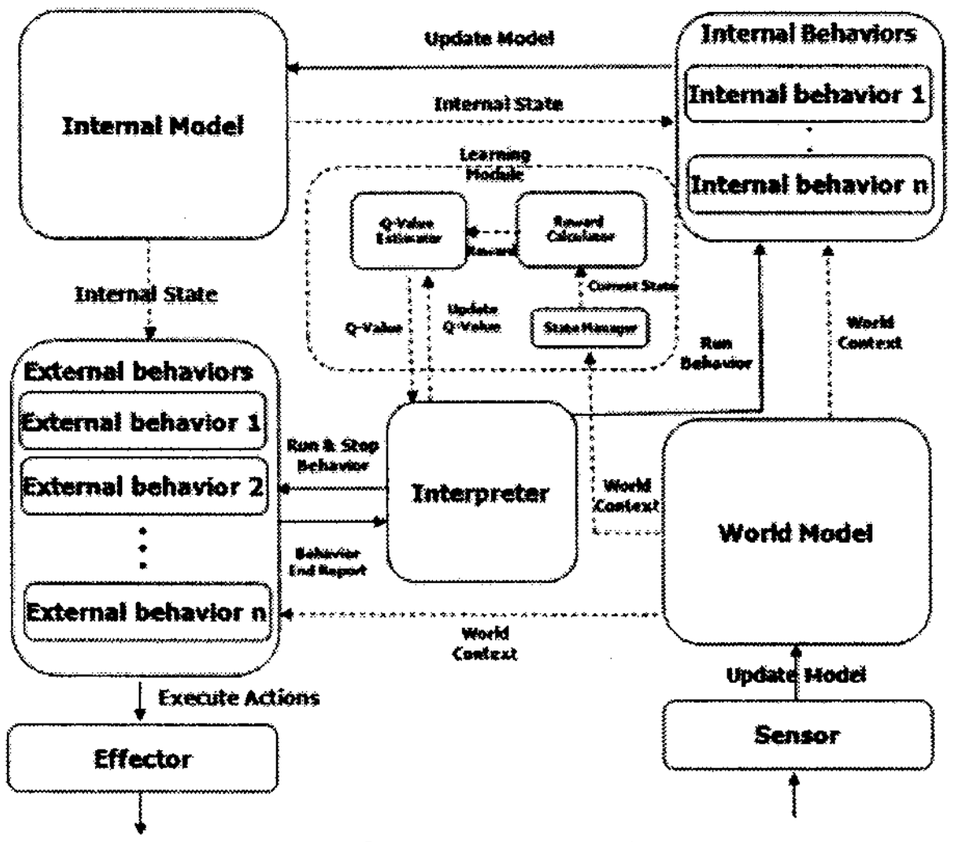

그림 5 - L-CAA 구조

L-CAA 구조에서는 에이전트 설계자가 응용 환경에 맞게 미리 상태를 결정할 수 있는 상태 변수들을 정의해준다고 가정한다. 센서 입력정보에 따라 월드 모델이 갱신되면, 상태 관리기(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 상태 관리기에 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. 가치 평가자(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. 인터프리터는 실행 가능한 행위들의 Q 함수 값과 에이전트 설계자가 제공한 효용성 값을 참고하여 1절에 설명한 행위 선택 단계를 수행한다.

L-CAA 구조의 외부행위는 CAA 구조에 비해 크게 다르지 않으나 행위의 수행 가능 메쏘드(applicable method), 효용성 메쏘드(utility method), 유지 가능 메쏘드(maintainable method) 등이 모두 상태 관리기에 의해 결정되는 명시적인 상태 표현을 참조하도록 변경되었다. 순수 학습에 기초한 행위 결정 메커니즘을 가진 에이전트 구조의 낮은 성능문제 때문에 L-CAA에서는 학습에 의해서 얻어진 행위 평가치를 기존 CAA가 가지고 있던 행동결정 메커니즘과 혼합하여 사용한다.

인터프리터(interpreter)는 기존 CAA의 행위정책에 의한 행위 결정 메커니즘을 통해 주어진 상황에 따른 행위들을 수행한다. CAA의 행위정책으로 수행할 행위를 결정할 수 없는 단계에서는 에이전트 설계자에 의해 주어진 효용성 값을 참고한다. 그리고 Q학습으로 얻어진 학습결과와 혼합하여 행위를 선택하는 기준으로 사용한다. 그리고 행위의 수행 도중 선택된 행위 보다 효용성이 높은 행위들이 있을 경우에도 가치 평가자의 도움을 받아 행위들의 가치함수인 Q값을 계산하고 가장 Q값이 높은 행위를 수행한다. 따라서 에이전트 설계자가 특정상황에서 효용성이 동일한 복수의 행위를 설계했을 경우에는 학습의 결과를 보조적으로 이용하여 가장 상황에 적합한 행위를 선택할 수 있다.

3.4 월드 모델과 상태 관리기

CAA 구조에서는 환경을 구성하는 다양한 객체들의 조합으로 월드 모델의 정보를 표현하였다. L-CAA 구조에서는 CAA에서 설계된 Java 객체로 이루어진 월드모델을 그대로 사용하였다. Java 객체로 이루어진 월드모델은 외부환경의 정보를 일정주기마다 확인한다. 그리고 비교적 자유롭게 에이전트 설계자가 원하는 정보를 Java 객체형태로 표현가능하게 설계되어 있다. 일반적으로 복잡한 실세계 문제에 강화 학습을 효과적으로 적용하기 위해서는 환경변화를 잘 표현할 수 있는 유한개의 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 실세계의 복잡한 환경상태를 간결하게 표현할 수 있다. 센서입력을 통해 월드 모델이 갱신되면 상태 관리기는 정해진 상태변수들의 값을 새로이 계산함으로써 환경의 현재 상태를 결정한다. 또한 상태 관리기는 언제나 특정 행위가 실행되기 이전 상태와 실행 이후 상태를 함께 저장하였다가 제공함으로써 행위 실행 이후 가치 평가자의 Q 함수 값 갱신을 돕는다.

3.5 행위와 보상계산기

L-CAA 구조에서는 각 행위 객체를 구성하는 수행 가능 메쏘드, 효용성 메쏘드, 유지 가능 메쏘드 등이 모두 불리 언 상태변수 값의 벡터로 표현된 현재 상태를 참조하여 반환 값을 결정하도록 수정되었으며, 대기 메쏘드, 복귀메쏘드, 중지 메쏘드 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 발생시켰었다.

L-CAA 구조에서는 행위의 수행과 종료가 인터프리터의 요청에 의해서 처리 되도록 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 감시 단계에서 행위의 상태를 주기적으로 관찰하며. 행위가 완료된 시점을 기준으로 상태 관리기를 통해서 얻어진 현재 상태를 이용하여 보상메커니즘을 호출한다. 그림 6은 보상계산기의 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

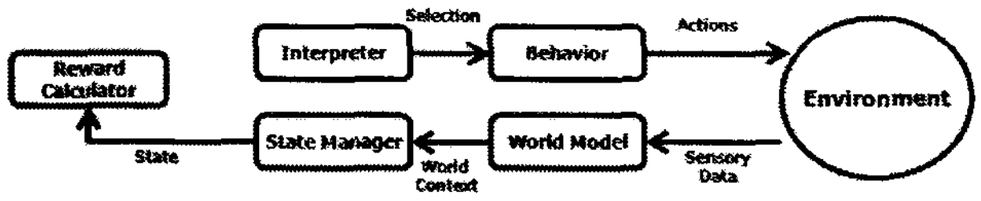

그림 6 - 보상 계산 메커니즘

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 메쏘드가 수행 된 뒤 현재 행위가 실행 대기상태로 전환 됐을 때의 상태를 나타낸다. 중지 상태는 행위의 유지조건이 만족하지 않았을 때 행위 객체가 수행 중지 메쏘드를 호출하면서 가지고 있던 자원과 선점 권을 반환하는 작업을 수행 할 때의 상태이다.

3.6 가치 평가자와 인터프리터

가치 평가자는 인터프리터에게 행위를 선택하기 위한 행위의 가치함수 값을 제공한다. 수행 가능 검사 과정을 통과한 행위의 우선순위는 Q 값과 효용성 값에 의해 결정되는데 각 행위의 Q값은 해당 행위의 수행 결과에 따라서 얻어지는 보상 값을 토대로 예측된다. 가치 평가자는 식1에 따라 상태 s에서 실행 가능한 행위 a의 Q 값을 갱신한다. 식1의 보상 값 r은 보상계산기로부터, 현재 상태 s와 다음 상태 s'은 상태 관리 기로부터 각각 제공되며, 학습율 γ와 감쇠율 α 는 각각 에이전트 설계자가 설정한 값을 이용한다.

$$Q(S, a) \leftarrow (1 - \alpha)\, Q(S, a) + \alpha\, [r + \gamma Max_{a'}\, Q\,(S', a') - Q(S, a)]$$

(식1)

L-CAA구조에서 인터프리터의 실행주기는 3.1절에서 설명한 행위 실행주기를 기초로 설계되었다. 인터프리터의 실행주기는 다음과 같은 그림 7의 의사코드(pseudo code)로 다시 표현할 수 있다.

인터프리터의 실행주기를 살펴 보면 행위 선택 단계에서 행위 수행 가능 메쏘드를 통해서 행위들의 수행 가능성을 검사하고 효용성 메쏘드와 학습으로 얻어진 Q값을 이용하여 선택되는 것을 알 수 있다. 설계자의 의도가 반영된 각 행위의 효용성 값이 크게 차이가 날 경우에는 효용성 값에 따라 행위가 결정되지만 그 외에 경우에는 학습의 결과에 따라 행위가 선택된다. 그리고 행위의 유지조건이 만족하지 않을 경우 즉각적으로 행위가 중단 되지만 유지조건이 만족되는 동안 효용성 값이 현재 수행중인 행위 보다 높은 행위가 발견 되었을 경우에도 현재 상태를 기준으로 우선순위가 높은 행위들 중에서 효용성 값과 Q값이 가장 높은 행위를 선택하는 것을 볼 수 있다.

```
Set previous_state = StMgr.GetCurrentState();
Set emergent_behavior = null;
Loop
  Set state = StMgr.GetCurrentState();
  Set current_behavior = FindRunningBehavior(state);
  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();
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();

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;
else
     current_behavior.Stop();
     Set current_behavior = null;
```

그림 7 - 인터프리터의 실행주기 의사코드

4. 응용 에이전트

4.1 응용 에이전트 환경

본 연구에서는 3차원 인터액티브 컴퓨터 게임인 Unreal Tournament (UT) 게임에서 자율적으로 행동하는 지능형 캐릭터 에이전트, 즉 UT-Bot을 L-CAA 구조를 이용하여 구현해 보았다. 그림 8는 UT-Bot이 전투를 벌이는 UT게임의 한 장면을 보여주고 있다.

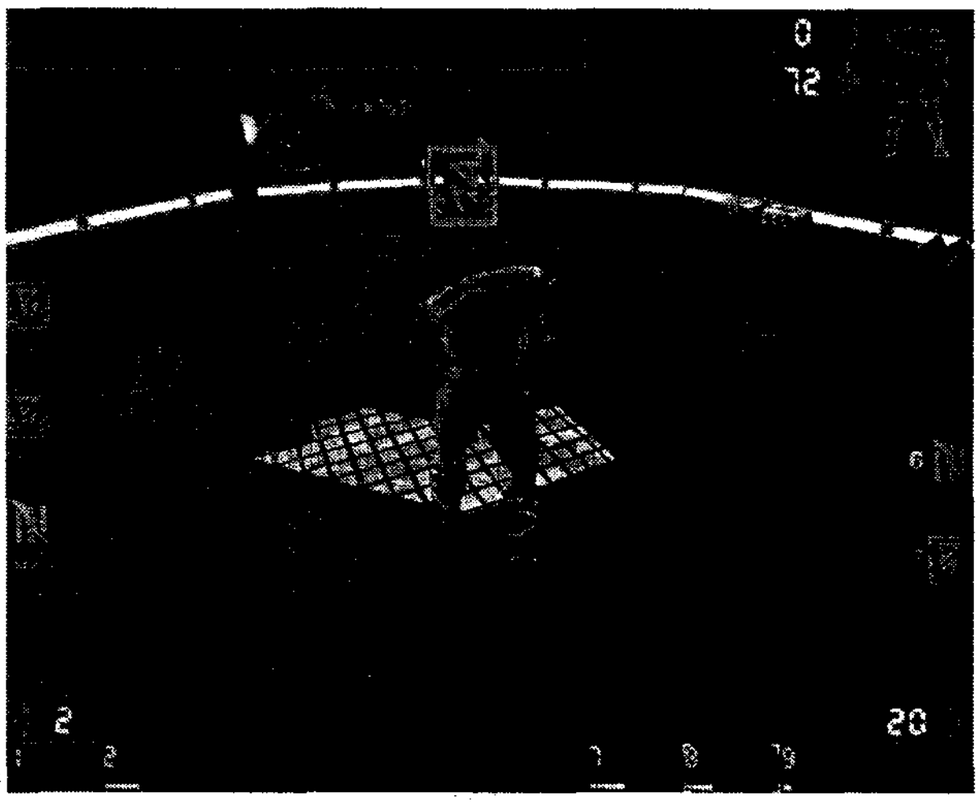

그림 8 - Unreal Tournament 게임

그리고 L-CAA의 UT게임 UT-Bot을 이미 개발된 Gamebots 시스템을 이용하여 제어하였다. Gamebots 시스템은 네트워크를 통해 원격의 UT-Bot을 제어하는 지능형 에이전트를 구현하고 실험할 수 있도록 개발된 연구용 테스트베드이다. Gamebots 시스템은 지능형 에이전트들인 보트 클라이언트(bot client)들과 Gamebots 서버로 구성된다. Gamebots 서버는 원격의 보트 클라이언트들에게 실시간 센서정보를 전달하고, 각 보트 클라이언트가 내린 행동명령을 받아 실행하는 역할을 한다. Gamebots 시스템은 환경의 변화, 센서 정보와 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-CAA구조를 이용해서 UT게임에서 지원하는 팀 도미네이션 게임을 자율적으로 수행하는 지능형 에이전트 L-CAA UT-Bot을 구현하였다. 팀 도미네이션 게임은 세 개의 거점을 점령, 유지 하는 팀이 승리하게 되는 게임이다.

먼저 L-CAA UT-Bot을 구현하기 위해 표 1과 같은 외부 행위를 설계, 구현 하였다 표 1의 행위들은 팀 도미네이션 게임에 사용되는 일반적인 행위들로 구성되어 있다. 탐색, 거점공격, 아이템 탐색, 일반적인 공격과 같은 행위들은 팀 도미네이션 게임을 수행하기에 충분하다.

표 1 - 외부행위 설계

행위기호	행위이름	설명
B1	Explore	맵을 탐색
B2	Dominate	거점 공격
B3	PowerUp	아이템 탐색
B4	Offensive Attack1	접근하면서 무기 1로 사격
B5	Offensive Attack2	접근하면서 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

표 2 - 불리언 상태 변수

상태 변수	설명
Visible_Enemy	적이 보이면 True, 아니면 False
Close_Enemy	상대방과 거리가 30 이하면 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	알고 있는 Node의 수가 절반이상이면 True, 아니면 False
Enough_Dom_Point	점령중인 거점이 2개 이상이면 True, 아니면 False

효율적인 팀 도미네이션 게임을 위해서 행위들의 수행 가능 조건과 유지 조건에 사용되는 상태변수들을 설계 하였다. 표 2는 불리 언 형태로 표현된 상태변수들을 나타낸다. 설계된 상태변수들은 표 1의 외부 행위들의 수행 가능 조건과 유지 조건을 구성하는데 사용되었다. L-CAA 상태 관리기는 실시간으로 Gamebot 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 3과 같다. 본 연구에서 설계한 응용 에이전트는 전투 능력향상에 학습을 이용하기 위해서 모든 보상은 전투와 관련되게 설계하였다. 그리고 Q-학습에 필요한 상태변수는 표 2의 모든 상태를 사용할 경우 $2^7 = 128$개의 상태가 표현 가능하게 된다. 하지만 모든 상태변수가 전투 행동에 필요한 요소가 아니기 때문에 상태변수 중 장애물, 적과의 거리, 탄환의 수,

체력 4가지 상태변수들만을 이용하여 $2^4 = 16$개의 상태만을 Q학습에 이용하였다.

표 3 - 평가 기준 별 보상 설계

평가기준	보상기준1	보상기준2	보상기준3
체력의 비교	손상 없음 3점	50이하의 손상 1점	50점 이상 혹은 사망 0점
상대편 제압평가	상대편 제압 1점	없음	없음
탄환의 비교	탄환 X 0.1 씩 감점 (기본 2점)	없음	없음

그림 9는 구현된 L-CAA UT-Bot의 사용자 인터페이스를 보여주는 화면이다. 사용자는 먼저 학습율과 감쇠율을 설정 할 수 있다. L-CAA UT-Bot은 환경에 따라 다양한 파라미터 설정이 필수적이다. 그리고 L-CAA UT-Bot의 네비게이션 행위를 실시간으로 관찰함으로써 직접 3차원 환경을 보지 않고도 UT-Bot의 이동 경로를 모니터링 할 수 있다. 뿐만 아니라 또 다른 UT-Bot을 추가, 삭제하는 기능을 추가하여 싱글 에이전트, 멀티 에이전트 환경을 동시에 설정할 수 있도록 설계하였다.

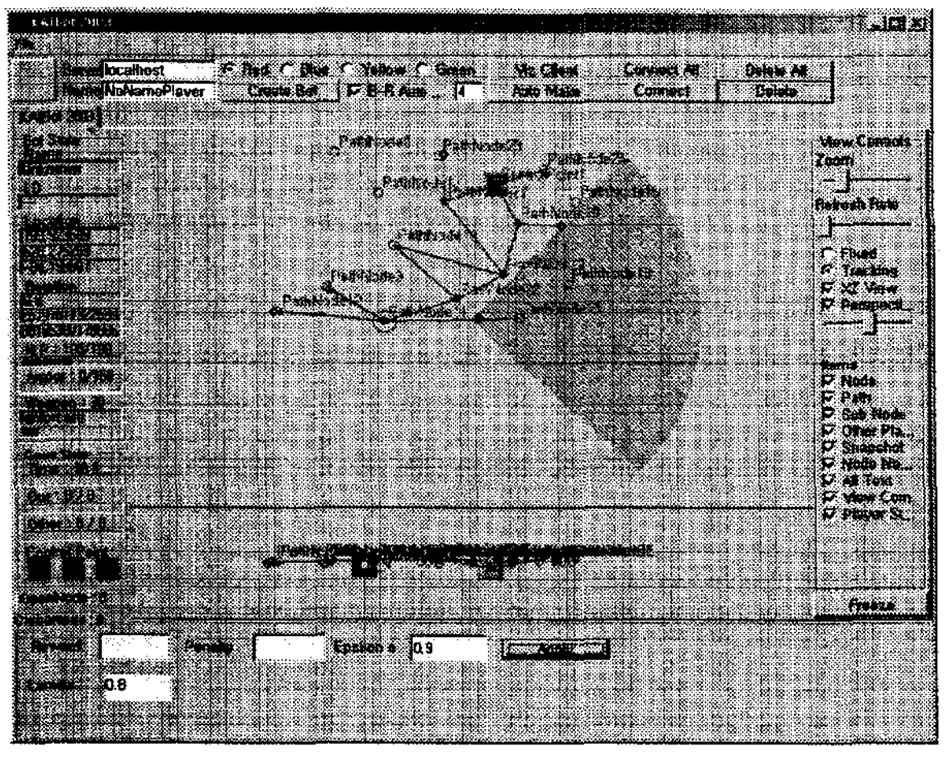

그림 9 - L-CAA UT-Bot 사용자 인터페이스

5. 실험

5.1 실험 목적 및 방법

L-CAA UT-Bot과 비교 실험을 위해서 기존 CAA UT-Bot과 순수 학습 에이전트 LO-CAA(Learning Only CAA) UT-Bot 2팀을 추가 구현하였다. 그리고 각각 팀 도미네이션 게임 수행을 통해서 L-CAA UT-Bot 성능비교 실험을 전개해보았다. 그리고 각각 목적별로 2가지 실험을 전개해 보았다.

첫 번째 실험은 L-CAA UT-Bot의 학습 수렴성을 분석하는 것이다. L-CAA 구조를 이용한 L-CAA UT-Bot이 UT게임 환경에서 정책을 학습하고 수렴된 정책을 얻어 내는 것을 관찰하기 위해서 LO-CAA UT-Bot과 L-CAA UT-Bot 2가지 에이전트를 일정 시간 동안 게임 환경에서 동작시킨 뒤 수렴성을 측정해 보았다. 그리고 시간대비 학습수렴 속도를 측정하여 L-CAA 구조의 학습속도를 순수 학습 에이전트 구조와 비교 해 보고자 하였다. 측정 방법은 600초 동안 모든 행위의 이전 Q값과 갱신된 Q값의 차의 평균을 조사해보는 방법으로 결정하였다. 첫 번째 실험을 위해서 총 30개의 노드로 구성된 공간에서 1000초 동안 LO-CAA UT-Bot과 L-CAA UT-Bot의 팀 도미네이션 게임을 전개 하여 결과를 관찰해 보았다.

두 번째 실험은 L-CAA UT-Bot의 팀 도미네이션 행위 성능 비교 실험이다. 게임의 최종 점수와 승률을 통해서 L-CAA UT-Bot이 응용환경에서 다른 에이전트 구조에 비해 높은 행위 성능을 나타내는지 관찰해보았다. 실험 방법은 CAA UT-Bot과 LO-CAA UT-Bot 그리고 L-CAA UT-Bot간에 최고 점수 200점을 기준으로 각각 30회씩 90회의 대전을 전개해보는 방식으로 진행되었다. 실험 측정은 각 에이전트간의 매회 점수 차이와 각 에이전트들 간의 90회 동안의 승패를 수를 조사하여 승률을 분석해 보는 방법으로 결정하였다. 두 가지 실험에서 Q학습에 필요한 파라미터는 감쇠율 α를 0.8로 학습률 γ를 0.9로 각각 설정하였다.

5.2 실험 결과

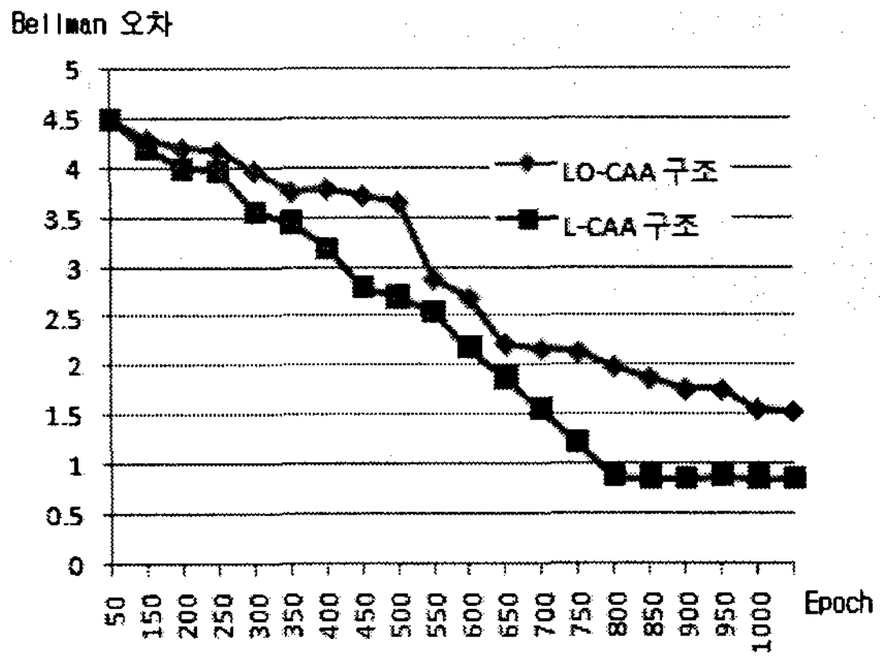

그림 10 - Q값 수렴 속도 비교

첫 번째 실험인 학습 수렴성 분석 실험을 전개하여 그림 10과 같은 Q값 수렴의 추이를 얻어 낼 수 있었다. <그림 10>을 보면 L-CAA UT-Bot의 Bellman 오차가 1.0 이하의 값으로 수렴되는 것을 알 수 있다. 그리고 LO-CAA UT-Bot의 수렴 속도보다 L-CAA UT-Bot의 수렴 속도가 빠른 것을 알 수

있다. <그림 10>의 결과를 통해서 에이전트 설계자의 효용성 메쏘드와 수행가능 메쏘드를 통해서 행위의 수행가능성을 검사하면서 보조적으로 학습의 결과를 이용한 것이 학습 수렴에 많은 영향을 준 것을 알 수 있었다. 다시 말해서 학습의 범위가 설계자의 지식을 통해서 제한되었고 이로 인해 짧은 시간 안에 수렴 된 정책을 얻어 낼 수 있었다는 결론을 얻을 수 있다.

두 번째 실험인 L-CAA UT-Bot의 행위 성능 비교 실험 실험의 결과로 표 4와 표 5의 결과를 얻어 낼 수 있었다. 표 4와 표 5를 참고해 보면 학습된 정책만을 사용하는 에이전트의 경우 가장 낮은 점수를 획득하고 전 게임에서 패한 것을 알 수 있다. 그리고 혼합 행동 정책을 가진 L-CAA UT-Bot은 CAA UT-Bot과 근소한 차이로 이기는 것을 확인 할 수 있다. L-CAA UT-Bot의 높은 성능은 다음과 같은 원인 때문으로 파악된다.

첫 번째로 CAA UT-Bot에 비해 학습에 의한 환경적응성을 가지기 때문이다. 두 번째로는 LO-CAA UT-Bot에 비해 사용자의 제어 지식을 이용할 수 있기 때문이다. 따라서 2가지의 원인 때문에 L-CAA UT-Bot이 팀 도미네이션 게임에서 높은 성능을 나타낸 것으로 결론을 낼 수 있다.

표 4- 팀 별 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	200 : 191	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- 팀 별 대전 승률비교

대전 횟수	L-CAA:CAA:	L-CAA:LO-CAA	CAA:LO-CAA
10회	7 : 3	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 행위 선택 기준과 온라인으로 학습을 통해 변경되는 행위 선택 기준 모두를 이용 할 수 있는 혼합 구조로서 높은 성능과 적응력을 동시에 갖춘 에이전트 구조이다. 모든 행위의 선택을 학습을 통해서 결정하는 순수 학습 에이전트 구조의 경우 초기 낮은 성능의 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. 또한 L-CAA 구조에서는 실시간 우선순위 행위 선택 기능을 추가하여 실시간 상황 감지 능력을 개선하였다. 본 논문에서는 UT 게임 환경에서 실시한 비교 실험을 통해 L-CAA 구조와 이에 기초한 응용 에이전트인 L-CAA UT-Bot의 높은 행위 성능과 학습 수렴성을 확인 할 수 있었다.

한편, L-CAA 구조는 CAA 구조에 비해 행위 감시 능력 개선 및 학습 기능이 추가되었으나, CAA 구조에 비해 많은 계산량이 필요한 점이 발견되었다. 따라서 계산 알고리즘의 개선을 통해서 계산량을 간소화하는 연구가 필요하다.

참고문헌

[1] Adobbati, R., Marshall, A.N., Scholer, A., Tejada, S., Kaminka, G.A., Schaffer, S., Sollitto, C. "GameBots: A 3D Virtual World Test Bed for Multiagent Research.", In Proceedings Of The Second International Workshop on Infrastructure for agents, MAS, and Scable MAS, 2001

[2] Brooks, R. A. "A Robust Layered Control System for A Mobile Robot". IEEE Journal of Robotics and Automation, pp 14-23, 1986.

[3] Brooks, R. A and Maja J Matarić, "Real Robots, Real Learning Problems", in Robot Learning, Jonathan H. Connell and Sridhar Mahadevan, eds., Kluwer Academic Press, pp 193-213, 1993..

[4] Christopher J. C. H. Watkins and Peter Dayan, Q-Learning, Machine Learning, Vol 8, No.3-4, pp 279-292, 1992..

[5] Dante I. Tapia1 , Javier Bajo1 , Juan M. Corchado1 , Sara Rodríguez1 and Juan M. Manzano1, "Hybrid Agents Based Architecture on Automated Dynamic Environments", Lecture Notes in Computer Science , Knowledge-Based Intelligent Information and Engineering Systems , Vol 4, pp 453-460, 2007.

[6] Felix. F. Ingrand, M. P. George, and A. S. Rao. "An Architecture for Real-Time Rea-soning and System Control". IEEE Expert Knowledge-Based Diagnosis in Process Engineering, Vol 7. pp34 - 44, December 1992

[7] Gal A. Kaminka, et al, "GameBots : A Flexible Test

Bed for MultiAgent Team Research", Communications of ACM, Vol.45, No.1, pp 43-45, 2002.

[8] In C. Kim, "CAA: A Context-Sensitive Agent Architecture for Dynamic Virtual Environments", Lecture Notes in Artificial Intelligence (LNAI), September. pp 146 ~ 151, 2005.

[9] Jaime Carbonell, Oren Etzioni, Yolanda Gil, Robert Joseph, CraigKnoblock, Steve Minton, and Manuela Veloso. "Prodigy: An integrated architecture for planning and learning". ACM SIGART Bulletin, 2(4), pp 51-55, 1991.

[10] Maja J Mataric , "Behavior-Based Robotics", In the MIT Encyclopedia of Cognitive Sciences, Robert A. Wilson and Frank C. Keil, eds., MIT Press, pp.74-77, April 1999.

[11] Maja J Matarić, "Behavior-Based Control: Examples from Navigation, Learning, and Group Behavior", Journal of Experimental and Theoretical Artificial Intelligence, special issue on Software Architectures for Physical Agents, Vol 9, pp323-336, 1997.

[12] Maja J Matarić and Monica Nicolescu, "Learning and Interacting in Human-Robot Domains", IEEE Transactions on Systems, Man, Cybernetics, special issue on "Socially Intelligent Agents - The Human in the Loop", K. Dautenhahn, ed., 31:5, pp419-430, 2001,

[13] R. Alami, R. Chatila, and B. Espiau. Designing an intelligent control architecture for autonomous robots. In ICAR - International Conference on Advanced Robotics, pages 435{440, Tokyo, Japan, November 1993.

[14] Ron. Sun, "An Agent Architecture For On-line Learning of Procedural And Declarative Knowledge". Proceedings of ICONIP'97, Springer-Verlag. 1997. pp.766-769,

[15] Valery Kuzmin, "Connectionist Q-Learning in Robot Control Task", Scientific Proceeding of Riga Technical University 5. Serija. Datorzinatne. Information Technology and Management Science, pp 38-48, Octorber 2002.

[16] Wooldridge, M.: Intelligent Agents. In: G. Weiss (ed) Multiagent Systems, MIT Press, Berlin Heidelberg. 1999