Journal of Korea Multimedia Society Vol. 11, No. 3, March 2008(pp. 353-363)

매크로 행동을 이용한 내시 Q-학습의 성능 향상 기법

성연식[†], 조경은[††], 엄기현[†††]

요 약

단일 에이전트 환경에서는 Q-학습의 학습 시간을 줄이기 위해서 학습결과를 전파시키거나 일렬의 행동을 패턴으로 만들어 학습한다. 다중 에이전트 환경에서는 동적인 환경과 다수의 에이전트 상태를 고려해야하기 때문에 학습에 필요한 시간이 단일 에이전트 환경보다 길어지게 된다. 이 논문에서는 단일 에이전트 환경에서 시간 단축을 위해서 유한개의 행동으로 정책을 만들어 학습하는 매크로 행동을 다중 에이전트 환경에 적합한 내시 Q-학습에 적용함으로써 다중 에이전트 환경에서 Q-학습 시간을 줄이고 성능을 높이는 방법을 제안한다. 실험에서는 다중 에이전트 환경에서 매크로 행동을 이용한 에이전트와 기본 행동만 이용한 에이전트의 내시 Q-학습 성능을 비교했다. 이 실험에서 네 개의 매크로 행동을 이용한 에이전트가 목표를 수행할 성공률이 기본 행동만 이용한 에이전트 보다 9.46% 높은 결과를 얻을 수 있었다. 매크로 행동은 기본 행동만을 이용해서 적합한 이동 행동을 찾아도 매크로 행동을 이용한 더 낳은 방법을 찾기 때문에 더 많은 Q-값의 변화가 발생되었고 전체 Q-값 합이 2.6배 높은 수치를 보였다. 마지막으로 매크로 행동을 이용한 에이전트는 약 절반의 행동 선택으로도 시작위치에서 목표위치까지 이동함을 보였다. 결국, 에이전트는 다중 에이전트 환경에서 매크로 행동을 사용함으로써 성능을 향상시키고 목표위치까지 이동하는 거리를 단축해서 학습 속도를 향상시킨다.

A Performance Improvement Technique for Nash Q-learning using Macro-Actions

Yunsik Sung[†], Kyungeun Cho[††], Kyhyun Um[†††]

ABSTRACT

A multi-agent system has a longer learning period and larger state-spaces than a single agent system. In this paper, we suggest a new method to reduce the learning time of Nash Q-learning in a multi-agent environment. We apply Macro-actions to Nash Q-learning to improve the learning speed. In the Nash Q-learning scheme, when agents select actions, rewards are accumulated like Macro-actions. In the experiments, we compare Nash Q-learning using Macro-actions with general Nash Q-learning. First, we observed how many times the agents achieve their goals. The results of this experiment show that agents using Nash Q-learning and 4 Macro-actions have 9.46% better performance than Nash Q-learning using only 4 primitive actions. Second, when agents use Macro-actions, Q-values are accumulated 2.6 times more. Finally, agents using Macro-actions select less actions about 44%. As a result, agents select fewer actions and Macro-actions improve the Q-value's update. It The agents' learning speeds improve.

Key words: Reinforcement Learning(강화학습), Q-Learning(Q학습), Multiagent System(다중에이전트), Macro Actions(매크로 액션)

※ 교신저자(Corresponding Author) : 조경은, 주소 : 서울시 중구 필동 3가 26(100-715), 전화 : 02)2260-3834, FAX: 02)2260-3766, E-mail : cke@dongguk.edu
접수일 : 2007년 10월 29일, 완료일 : 2008년 1월 29일
[†] 정회원, 동국대학교
(E-mail : sung@dongguk.edu)
[††] 종신회원, 동국대학교 영상미디어대학 게임멀티미디어공학과 조교수
[†††] 종신회원, 동국대학교 영상미디어대학 게임멀티미디어공학과 교수
(E-mail : khum@dongguk.edu)

1. 서 론

결정 트리구조를 이용한 의사 결정 방법은 에이전트가 발생할 수 있는 환경 상태에 따라 취할 행동을 트리구조로 정의하기 때문에 발생할 수 있는 환경 상태에 대한 정의가 필요하다[1]. 하지만 Q-학습은 발생할 수 있는 환경 상태를 정의하지 않고 모든 상태를 학습한다. 그래서 상태를 표현하기가 어려운 복잡한 환경에서는 에이전트의 행동과 에이전트의 내부 상태 정보만 정의하고도 학습이 가능한 Q-학습이 적절하다[2].

Watkins가 제안한 Q-학습은 다음과 같은 단점을 포함한다. 첫째, 모든 상태에서 각각의 행동에 대한 Q-값을 정의하기 때문에 상태 공간이 커지고 학습에 걸리는 시간이 길어지는 문제가 발생된다. 둘째, 상태 정보를 이산 상태로 표현하기 때문에 두 개의 이산값 사이에서 정의하지 못한 상태는 정확한 행동을 계산할 수 없다.

Q-학습은 단일 에이전트 환경에 적합하기 때문에 다중 에이전트 환경에 적용할 때 몇 가지 문제가 발생된다. 첫 번째는 환경이 더 이상 정적이지 않다. 단일 에이전트의 경우에는 의사 결정을 할 때 환경이 정적이지만 다중 에이전트 환경에서는 다른 에이전트의 위치값과 같이 변화하는 상태값을 포함하기 때문에 이를 고려한 학습이 이루어져야 한다. 두 번째는 환경이 정적이지 않기 때문에 학습된 결과가 임의의 값으로 수렴하지 않는다. 즉, 에이전트의 행동에 대한 평가는 행동을 취한 에이전트뿐만 아니라 같은 환경에 있는 에이전트가 취한 행동에 의해서 결정된다. 마지막으로 다중 에이전트 환경에서는 동적인 환경과 다수의 에이전트 상태를 고려해야하기 때문에 학습에 필요한 시간이 단일 에이전트 환경보다 길어지게 된다. 다중 에이전트에 관련된 학습 연구들이 활발하게 이루어지면서 다중 에이전트 환경에 적용할 때 생기는 문제점을 해결하기 위한 연구들이 늘어나기 시작했다[3].

이 논문에서는 다중 에이전트 환경의 Q-학습 문제 중에서 학습 속도를 향상시키기 위한 방법을 소개한다. Q-학습의 성능 향상을 위해 사용된 매크로 방식을 다중 에이전트의 Q-학습에 적용함으로서 다중 에이전트 환경에서의 학습 시간을 단축할 수 있는 방법을 제안한다. Q-학습은 연속적인 값을 가지는 환경 정보를 이산 상태로 바꾸어서 각각의 상태에 적합한 행동을 학습한다. 하지만 학습 시간의 문제로 상태 공간의 개수를 줄이기 위해서 하나의 상태 범위를 크게 잡기 때문에 학습 효과가 낮아지는 현상이 있다. 학습 속도의 향상은 동일한 시간에 더 많은 학습이 가능하고 상태 범위를 세부적으로 잡아도 정해진 시간에 학습이 가능하기 때문에 학습 성능이 높아지는 결과로 이어진다.

논문의 구성은 다음과 같다. 2장에서는 Q-학습을 소개하고 단일 에이전트 환경과 다중 에이전트 환경에서 Q-학습을 향상시키기 위한 연구들을 기술한다. 3장에서는 매크로 행동을 다중 에이전트 환경에 적합한 Q-학습에 적용하는 방법에 대해 기술하며 4장에서는 3장에서 언급한 방법을 구현하고 이를 이용해서 실험한 내용을 설명한다. 마지막으로 5장에서는 4장에서 기술한 실험결과를 이용해서 결론을 짓는다.

2. 관련 연구

이 장에서는 Q-학습을 간략하게 소개하고 이와 관련된 연구들을 기술한다. 단일 에이전트 환경에서 학습 속도를 높이기 위한 방법을 소개한다. 그리고 다중 에이전트에 Q-학습을 적용하기 위한 기존 연구들을 나열한다.

2.1 Q-학습

Q-학습은 Watkins가 제안한 대표적인 강화 학습이다. Q-학습은 행동의 평가로 가치함수를 사용한다. 수식으로 표현하면 다음과 같다.

$$V^{\pi}(s_t) \equiv r_t + \beta r_{t+1} + \cdots = \Sigma \beta^i r_{t+i} \quad (1)$$

식(1)에서 가치함수 $V^{\pi}(s_t)$는 t시간에 발생되는 상태s_t에서 행동Π를 취할 때 평가되는 값으로 정의한다. 이때 r_t는 t시간에 얻은 보상값이며 r은 시간에 따른 보상의 감소율이며 0에서 1사이의 값을 가진다. 식(1)에서 구한 값은 Q-값을 구하기 위한 식(2)에서 보상값 r로 사용된다. 공식은 다음과 같다.

$$Q(s,a) \leftarrow Q(s,a) + \alpha[r + \beta \max Q(s',a') - Q(s,a)] \quad (2)$$

식(2)에서 Q(s,a)는 s상태에서 행동 a를 선택할 때

의 Q-값이다. max Q(s',a')는 t+1시간에 얻을 수 있는 최대 Q-값이다. α는 학습률을 나타내고 r과 r은 시간에 따른 보상의 감소율과 보상값을 나타낸다. 새로운 Q-값은 기존에 구한 Q-값과 보상값을 이용해서 계산한다.

2.2 단일 에이전트 환경에서 Q-학습의 성능 개선 연구

Q-학습은 한번의 행동으로 하나의 Q-값이 갱신되기 때문에 학습시간이 길어진다. 이 장에서는 학습시간문제를 해결하기 위해 여러 개의 Q-값이 동시에 갱신되는 방법과 행동 목록을 만들어 학습 속도를 높이는 방법을 소개한다.

가장 먼저 학습 시간을 단축하기 위해서 한 번의 학습으로 하나 이상의 Q-값이 갱신되는 분포 기여도를 제안한 연구가 있다[4]. 일반적으로 Q-학습은 하나의 행동과 하나의 상태에서 한번 Q-값을 갱신한다. 하지만 분포 기여도를 이용할 경우에는 인접한 Q-값에도 기여정도에 따라서 동시에 갱신하기 때문에 학습효과가 높아진다.

두 번째로 영향력분포도를 이용한 Q-학습이 있다[5]. 영향력분포도란 게임의 지형 정보를 격자 형식의 무늬 형태의 그리드에 표현하는 방식이다. 지형에 영향을 미치는 요소를 그리드에 수치값으로 표현하고 각각의 값을 인접한 격자 형식의 무늬에 전파시키는 방법이다. 이 학습 방법은 영향력분포도를 이용해서 Q-테이블의 이산 값으로 정의하지 못한 중간 값을 채우는 방식이기 때문에 다른 Q-학습으로 얻은 Q-테이블의 크기를 확장해서 에이전트의 성능을 향상 시키는 것이 가능하다.

마지막으로 매크로 행동을 이용한 Q-학습이 있다[6]. 매크로 행동은 에이전트가 수행할 행동을 기본 행동으로 정의한다. 기본 행동은 순서대로 유한개 정의하여 행동 정책을 구성한다. 만약, 에이전트가 매크로 행동을 선택하면 행동 정책에 정의된 모든 기본 행동을 수행해야 다음 행동을 선택한다. 예를 들어 그림 1에서 에이전트가 매크로 행동1을 선택하면 기본 행동1, 기본 행동3, 기본 행동1 그리고 기본 행동2 순서대로 실행하고 다음 행동을 선택한다. 에이전트는 행동을 결정할 때 기본적인 행동이나 매크로 행동 중에 하나를 선택한다. 매크로 행동을 이용한 Q-학습에서는 학습한 매크로 행동이 환경에 적합한지에 따라서 학습 시간의 차이가 발생된다. 매크로

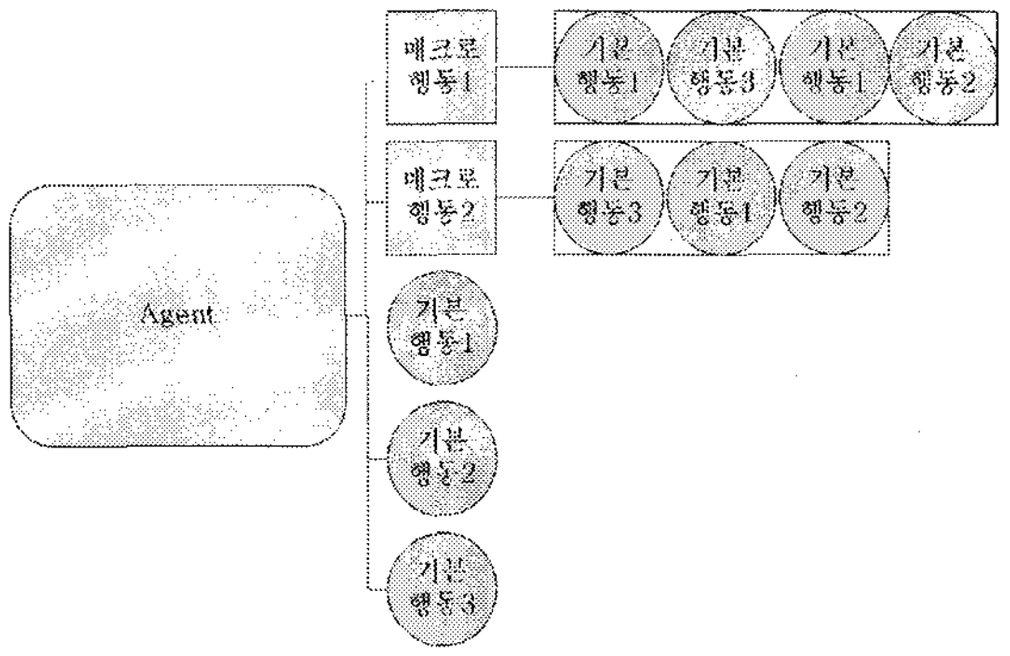

그림 1. 단일 행동과 매크로 행동

행동의 가장 큰 장점은 학습한 Q-값을 인접한 Q-값에 전파시키는 성능이 탁월하다는 것이다.

이러한 방법은 모두 단일 에이전트 환경에서 Q-학습의 학습 속도를 향상시키기 위한 방법이다. 다중 에이전트 환경은 단일 에이전트 환경보다 더 많은 상태 공간을 정의하기 때문에 이러한 학습 시간 단축에 대한 필요성이 높아진다.

2.3 다중 에이전트 환경에서 Q-학습의 성능 개선 연구

다중 에이전트 환경에서 Q-학습과 관련된 기존 연구들은 복잡한 환경에서 효율적으로 에이전트가 학습하는 방법들을 소개한다. 첫 번째로 Maker는 MAXQ 프레임워크를 다중 에이전트 환경에 적용해서 계층적으로 학습하는 방법을 소개하고 있다[7]. 계층적 학습 방법은 학습을 여러 단계로 나누어서 하위 단계 학습 결과가 상위 단계 학습에 영향을 미치는 방법으로 협동 작업이 필요한 다중 에이전트 환경에서 학습 속도를 높이는 효과를 준다.

두 번째로 Tesauro는 다른 에이전트와 협력하는 방안으로 베이지안 추론과 혼합된 전략을 이용한다[8]. 다중 에이전트 환경에서는 자신의 행동이 전체 에이전트에 이익이 되어야 하기 때문에 행동을 선택함에 있어 다른 에이전트 행동을 예상하는 방법으로 베이지안 추론을 사용하고 있다. 앞서서 소개한 매크로 행동과의 차이점은 혼합된 전략은 서로 다른 에이전트의 기본 행동을 조합해서 만든 것이고 매크로 행동은 하나의 에이전트가 취할 수 있는 기본 행동을 조합해서 만든 것이다. 혼합된 전략을 학습할 때 에이전트는 기본적인 행동을 선택하기 보다는 혼합된 전략을 선택해서 다른 에이전트와 협력하기 위한 방

법을 학습한다.

마지막으로 Hu와 Wellman이 내시 균형 이론을 이용해서 Q-학습을 다중 에이전트 환경에 적용한 내시 Q-학습이 있다[9]. Q-학습은 단일 에이전트 환경을 위한 학습 알고리즘이기 때문에 현재 환경에서 취할 수 있는 행동 중에서 최대의 보상을 얻는 행동을 선택하는 방법이다. 하지만 내시 Q-학습은 Q-학습 알고리즘에 모든 에이전트가 균일하게 보상을 받는 내시 Q-값의 계산 공식을 추가한다. 이 값을 이용해서 행동을 결정할 때 다른 에이전트의 행동을 예측하고 각각 행동의 보상값 합이 최대가 되는 행동을 에이전트가 결정하도록 한다. 내시 Q-학습은 다중 에이전트 환경에서 다른 에이전트의 행동을 예측하기 위해서 자신의 행동을 결정하기 위한 알고리즘을 그대로 사용한다.

이 논문에서는 단일 에이전트 환경에서 학습 효과를 높이고 학습 시간 단축을 위해 사용되었던 매크로 행동을 다중 에이전트 환경에 적합한 내시 Q-학습과 함께 적용함으로써 다중 에이전트 환경에서의 Q-학습 속도 문제를 보완하는 방법을 소개한다.

3. 학습 시간 단축 Q-값

이 장에서는 매크로 행동을 이용해서 다중 에이전트 환경에 적합한 내시 Q-학습의 학습 시간을 단축하는 방법을 설명한다. 학습 시간의 단축은 동일한 시간에 더 많은 내용을 학습할 수 있기 때문에 성능 향상으로 이어진다.

다중 에이전트 환경에서 학습할 때 에이전트가 행동을 결정하기 위해서는 다른 에이전트의 정보가 필요하다. 그래서 이러한 정보를 얻기 위한 별도의 학습이 필요하다. 이러한 학습은 단일 에이전트에 비해서 학습 시간이 길어지는 원인이 된다. 예를 들어 n개의 에이전트가 있으면 (n-1)개의 다른 에이전트에 대한 행동 예측 학습이 필요하고 전체 n*(n-1)개의 학습이 에이전트에 대한 학습이 필요하기 때문에 다양한 상황을 고려하고 반영하기 위해서는 우선 학습 시간을 단축해야 한다.

3.1 내시 Q-값

내시 Q-학습은 자신의 의사 결정을 하기 위해서 다른 에이전트의 행동을 고려해야 하는 데 이를 공식으로 표현하면 다음과 같다

$$Q^i_{t+1}(s,a^1,...,a^n) = (1-\alpha_t)Q^i_t(s,a^1,...,a^n) + \alpha_t[r^i_t+\beta NashQ^i_t(s')] \quad (3)$$

여기서 $NashQ^i_t(s')$는 다음과 같다

$$NashQ^i_t(s') = \pi^1(s')\cdot\cdot\cdot\pi^n(s')\cdot Q^i_t(s') \quad (4)$$

식(3)에서 s는 현재 상태를 나타내고 a^n은 n번째 에이전트의 행동을 나타낸다. t는 현재 시간을 표현하고 $Q^i_t(s,a^1,...,a^n)$은 t시간에 1번째 에이전트부터 n번째 에이전트가 a^1부터 a^n까지 각각의 행동을 취할 때 s상태에서 i번째 에이전트의 Q-값을 나타낸다. 이 Q-값을 얻기 위해서는 우선 학습률 α_t가 결정되어야 하고 보상의 감소율 β가 결정되어야 한다. 그리고 r^i_t는 t시간에 i번째 에이전트의 보상값을 나타낸다. 식(3)에서 $NashQ^i_t(s')$는 s'상태에서 에이전트가 각각의 행동 π^n을 취할 때 내시 이론에 의해 결정된 보상값이고 식(4)에서 정의한다.

다중 에이전트에서 다른 에이전트의 정보는 서로 공유하지 않기 때문에 에이전트가 스스로 정보를 만들어야 한다. 그래서 식(4)를 계산하기 위해서는 다른 에이전트의 Q-값이 필요한데 이 값은 학습을 통해서 얻어지게 된다.

$$Q^j_{t+1}(s,a^1,...,a^n) = (1-\alpha_t)Q^j_t(s,a^1,...,a^n) + \alpha_t[r^j_t+\beta NashQ^j_t(s')] \quad (5)$$

식(5)는 j번째 에이전트가 다른 에이전트의 Q-값을 얻기 위한 공식으로 식(3)과 동일하다. 내시 Q-학습의 장점 중에 하나는 다른 에이전트의 행동을 예측하기 위해서 별도의 추론 알고리즘을 사용하지 않고 동일한 알고리즘을 사용함으로써 실험의 복잡성을 낮추었다.

3.2 매크로 행동을 이용한 Q-값

에이전트가 임의의 상태에서 매크로 행동 m을 선택할 때 Q-값은 다음 식(6)과 같이 갱신된다.

$$Q_{t+1}(s,m) = Q_t(s,m) + \alpha[r+\beta^n \max Q(s_{t+n},m)-Q(s_t,m)] \quad (6)$$

이때 r은 보상값을 의미하며, 다음 식(7)로 정의된다.

$$r = r_{t+1} + \beta r_{t+2} + \cdot\cdot\cdot + \beta^{n-1} r_{t+n} \quad (7)$$

식(6)에서 s는 현재 상태를 나타내고 m은 에이전트가 선택한 매크로 행동이다. 그래서 $Q_t(s,m)$은 t시간에 상태가 s이고 매크로 행동 m을 선택할 때의 Q-값을 나타낸다. 이 Q-값을 얻기 위해서는 우선 학습율 α가 결정되어야 하고 감소율 β가 결정되어야 한다. 여기서 감소율 β는 0에서 1사이의 값을 가진다. $\max Q(s_{t+n},a)$는 t+n시간의 상황 s에서 매크로 행동 m를 선택할 때 가장 큰 Q-값을 나타낸다. 이 식은 Q-학습의 Q-값을 구하기 위한 식(2)에서 유도된 식이다. 식(2)에서 에이전트 행동 a 대신에 매크로 행동 m으로 표기한다. 마지막으로 Q-값은 매크로 행동이 끝난 t+n때 Q-값이 갱신되기 때문에 식(2)에서 t+1의 시간이 t+n으로 수정된다. 여기서 보상 r은 누적된 보상값이며 식(7)과 같이 표현된다. 즉 누적된 보상값은 매크로 행동을 취할 때 얻는 보상값의 합이다. 각각의 보상값은 감소율 β가 적용된다.

3.3 다중 에이전트 환경에서의 매크로 행동 적용

이 논문에서 제안하는 방식은 다중 에이전트 환경에서 사용되는 내시 Q-학습 알고리즘에 매크로 행동을 이용한 Q-학습 알고리즘을 추가하는 방식으로 구성한다. 매크로 행동을 이용한 Q-학습은 식(7)처럼 여러 개의 기본 행동으로 구성된 매크로 행동의 누적 보상값으로 계산한다. 내시 Q-학습에 매크로 행동을 적용하기 위해서 식(3)의 보상값 r^i_t를 식(7)처럼 누적된 보상값으로 계산해서 공식으로 표현하면 다음과 같다.

$$Q^i_{t+1}(s,a^1,...,a^n) = (1-\alpha_t)Q^i_t(s,a^1,...,a^n) + \alpha_t[r^i+\beta NashQ^i_t(s')] \quad (8)$$

여기서 $NashQ^i_t(s')$는 식(4)와 동일하게 다음과 같다.

$$NashQ^i_t(s') = \pi^1(s')\cdot\cdot\cdot\pi^n(s')\cdot Q^i_t(s') \quad (9)$$

식(8)에서 s는 현재 상태를 나타내고 a^n은 n번째 에이전트의 행동을 나타낸다. 그래서 $Q^i_t(s,a^1,...,a^n)$은 t시간에 1번째 에이전트부터 n번째 에이전트가 a^1부터 a^n까지 각각의 행동을 취할 때 s상태에서 i번째 에이전트의 Q-값을 나타낸다. 식(8)의 $NashQ^i_t(s')$는 식(4)와 동일한 식(9)에서 다시 정의한다.

보상값 r^i는 식(7)과 동일하게 다음과 같이 정의된다.

$$r^j = r_{t+1} + \beta r_{t+2} + \cdot\cdot\cdot + \beta^{n-1} r_{t+n} \quad (10)$$

식(10)은 매크로 행동을 수행할 때 모든 행동의 보상값 합으로 식(7)과 동일하다. β는 감소율을 t는 시간을 표현한다. 다른 에이전트의 정보를 만들기 위해서 내시 Q-학습과 동일하게 학습할 때 사용된 동일한 알고리즘을 이용한다.

$$Q^j_{t+1}(s,a^1,...,a^n) = (1-\alpha_t)Q^j_t(s,a^1,...,a^n) + \alpha_t[r^j_t+\beta NashQ^j_t(s')] \quad (11)$$

식(11)은 j번째 에이전트가 다른 에이전트의 행동을 예측하기 위해 Q-값을 얻기 위한 공식으로 내시 Q-학습처럼 식(8)과 동일한 공식을 사용한다.

내시 Q-학습에 매크로 행동을 적용할 때 진행되는 과정을 순서도로 표현하면 그림 2와 같다. 에이전트는 행동을 선택하기 전에 내시값을 계산하면서 모든 에이전트에게 이익이 될 수 있는 행동을 선택한다. 그리고 단일 행동일 경우에는 선택한 행동을 취하고 매크로 행동일 경우에는 매크로 행동의 첫 번째 단일 행동을 실행한다. 순서도를 보면 단일 행동은 하나의 행동을 취하고 Q-값을 갱신하지만 매크로 행동은 매크로에 정의된 모든 행동이 끝나고 나서 Q-값을 갱신한다.

4. 실험 및 결과

이 장에서는 매크로 행동을 이용한 내시 Q-학습의 성능을 알아보기 위해 정의된 실험 환경과 구현 방법을 소개하고 실험 결과를 기술한다.

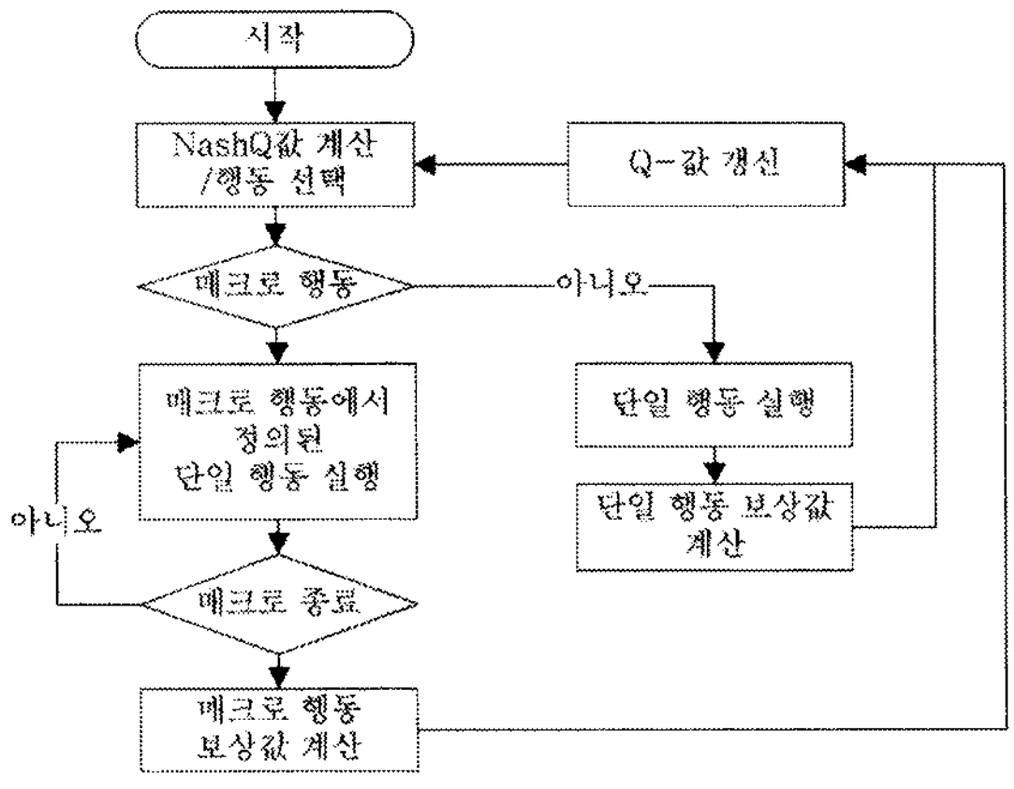

그림 2. 단일 행동과 매크로 행동 수행 과정

4.1 실험 내용

에이전트가 하나의 목표를 달성하기 위한 하나의 과정을 에피소드라고 한다. Q-학습은 에피소드를 반복적으로 학습함으로써 보상이 높은 행동과 그렇지 않는 행동으로 구분된다.

실험 환경은 격자 형식의 무늬 형태로 구성이 되고 가로, 세로 네 칸씩 총 16개의 칸으로 구성된다. 실험은 두 개의 에이전트로 진행한다. 에이전트의 초기 위치는 각각의 에피소드마다 달리 설정한다. 에이전트의 목표는 같은 팀의 에이전트와 협동해서 동시에 목표 지점까지 이동하는 것이다. 목표 지점은 그림 3처럼 알파벳 'G'로 표시를 하고 각각의 에피소드마다 위치가 변경된다. 그림 4처럼 두 개의 에이전트가 서로 목표 지점에 인접한 곳에 위치하고 있다가 동시에 목표 위치에 이동한 경우에 200의 보상을 받는다. 만약 그림 5처럼 에이전트 한 개만 목표 위치로 이동한 경우에는 -250의 보상값을 받도록 설정한다.

전체 실험은 세 단계로 진행된다. 첫 번째 단계에서는 두 개의 에이전트가 상하좌우로 한 칸씩 이동 가능한 네 개의 기본 행동만 정의하고 실험한다. 두 번째 단계에서는 에이전트에 기본 행동과 상하방향으로 두 칸씩 이동 가능한 두 개의 매크로 행동을 추가해서 실험한다. 마지막으로 에이전트에 기본 행동과 네 방향 두 칸씩 이동하는 매크로 행동을 정의하고 실험했다. 각 단계마다 40만 번 학습을 진행하고 성공률, 행동 선택 횟수 그리고 Q-값의 변화를 4만 번마다 기록하고 평가한다.

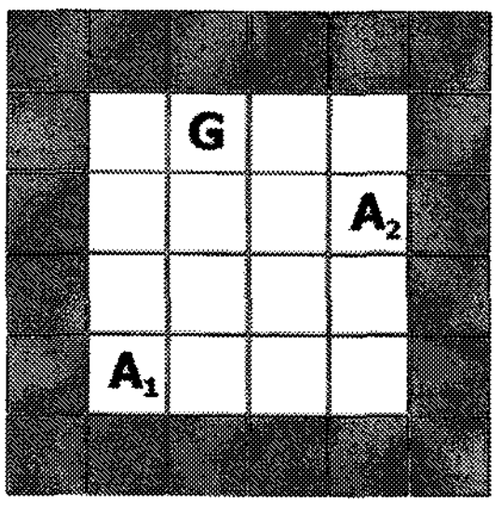

그림 3. 실험 환경

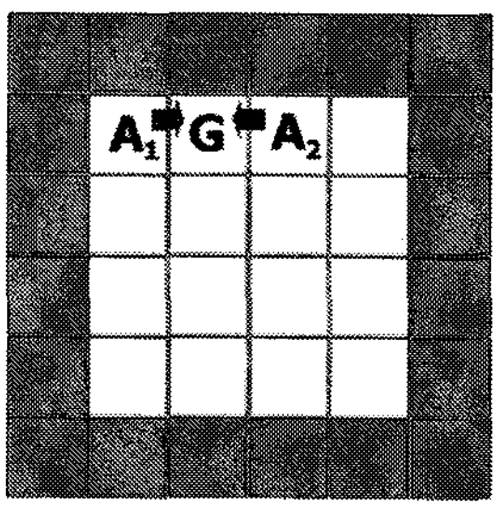

그림 4. 목표 위치에 동시에 진입할 때 성공

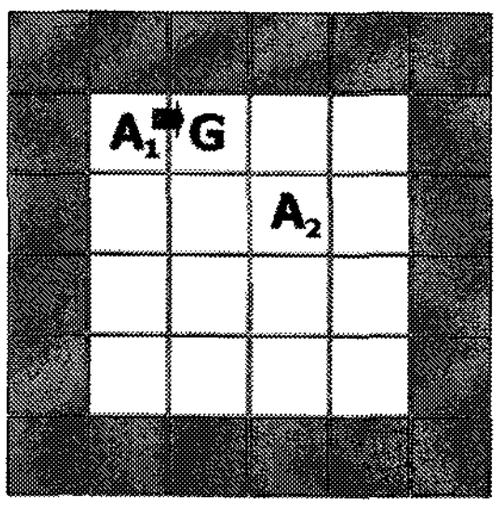

그림 5. 하나의 에이전트만 목표에 진입하면 실패

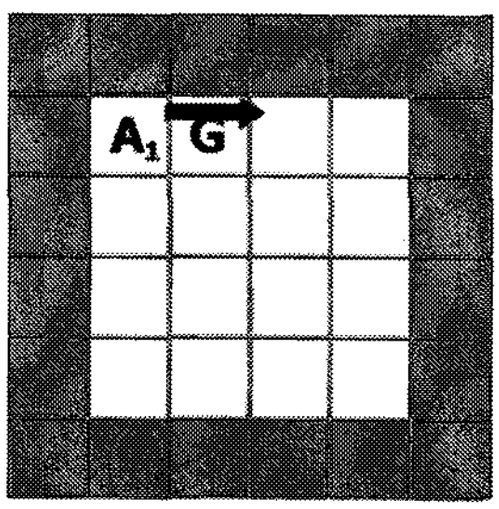

그림 6. 매크로 행동

에이전트는 행동을 선택할 때 각 단계에서 선택이 가능한 네 가지의 기본 행동과 네 가지의 매크로 행동 중에서 하나를 선택한다. 에이전트가 이동하다가 정지하는 조건이 두 가지가 있다. 첫 번째는 이동시 벽에 부딪히는 경우에 이동하지 않고 부딪힐 때까지만 이동한다. 두 번째는 이동 중에 목표 위치가 있는 경우에 멈추게 된다. 예를 들면 그림 6과 같이 에이전트 A_1이 매크로 행동을 선택해서 오른쪽으로 두 칸 이동하려고 할 때 이동 경로에 목표 G가 있는 경우에 더 이상 이동하지 않고 목표 위치에서 정지한다.

4.2 실험 구현

이 장에서는 논문에서 제안한 방법의 성능을 실험하기 위해 구현한 시스템을 간략하게 소개한다. 그림 7은 실험에 사용된 프로그램의 구조를 클래스 다이어그램으로 표기한다.

실험에서 사용한 시스템에서는 두 개 이상의 에이전트를 관리하기 위해서 그룹 클래스를 정의한다. 이

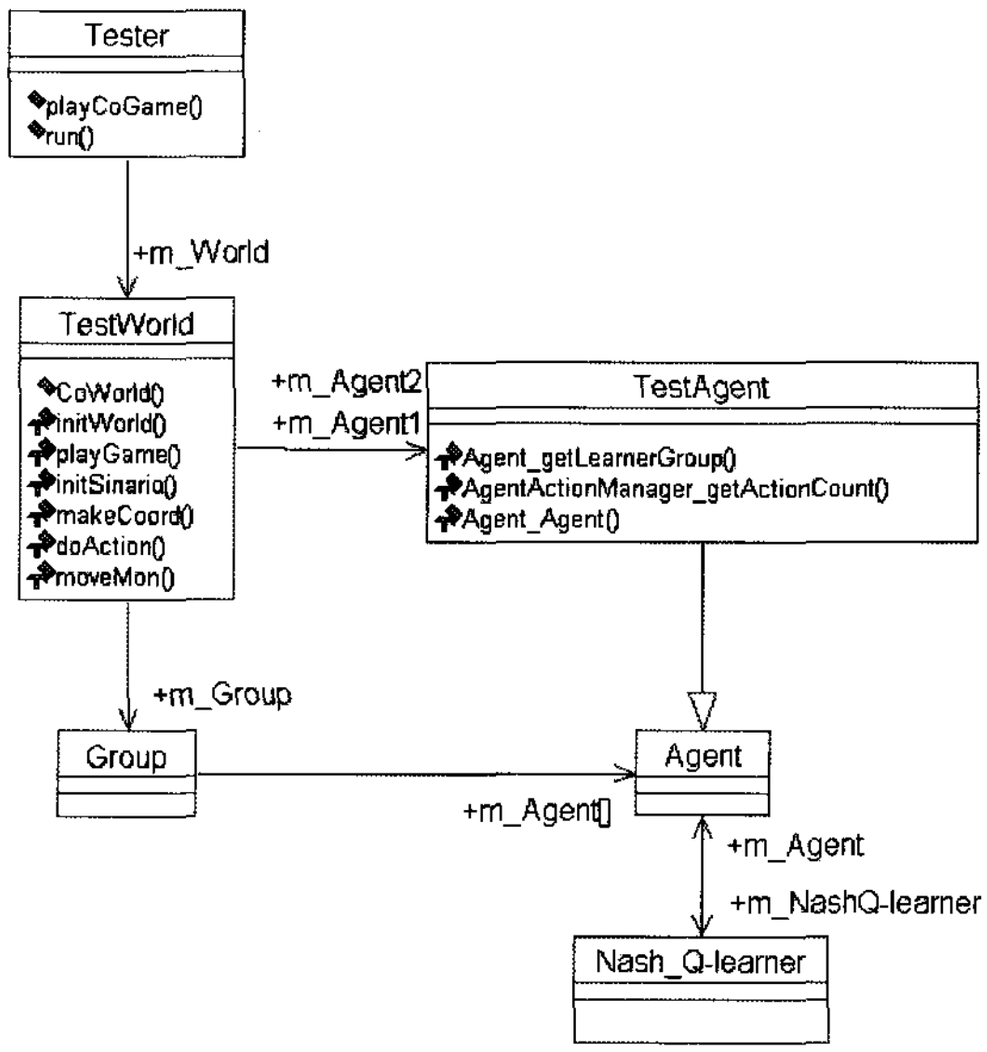

그림 7. 시스템 클래스 다이어그램

클래스를 통해서 실험에서 사용할 에이전트를 등록하고 통합적으로 에이전트를 관리하는 기능을 제공한다. 에이전트가 행동을 결정할 때 학습 중인 경우에는 Q-학습 방법에 따라 가장 큰 Q-값을 가지는 행동을 선택하거나 임의의 행동을 선택한다. 그리고 행동 결과를 Q-테이블에 반영한다. 반면에 학습을 마친 경우에는 가장 큰 Q-값을 가지는 행동을 선택하는 작업만 필요하다. 학습에만 필요한 기능은 TestAgent 클래스로 분리해서 구현하고 기본적으로 에이전트가 이동하고 행동을 결정하는 기능은 Agent 클래스에 구현한다. 에이전트의 학습 기능은 확장성을 고려하여 에이전트 클래스 안에서 구현하지 않고 별도의 클래스로 분리해서 구현한다. 학습 클래스에서는 다음과 같이 세 가지 기능을 제공한다. 학습에 필요한 인터페이스를 제공하는 기능, 일반 Q-학습을 위한 기능 그리고 내시 Q-학습을 위한 기능이 제공된다. 내시 Q-학습은 일반 내시 Q-학습과 매크로 행동을 이용한 내시 Q-학습을 모두 구현한다.

그림 7처럼 Tester 클래스는 실행 어플리케이션으로 인터페이스를 통한 각종 입력 처리를 담당한다. TestWorld 클래스는 실험 환경 정보 관리와 실험 진행처리를 담당하고 있다. Test World 클래스는 실험에 필요한 두 개의 TestAgent 클래스와 그룹 클래스를 포함한다.

실험 환경은 자바로 구현되었고 실험 프로그램 인터페이스는 그림 8과 같다. 실험 프로그램 인터페이스를 이용해서 강화 학습 알고리즘에 적용할 감소율, 학습률을 정의한다. 그리고 에이전트가 임의의 행동을 선택할 확률을 0.0에서 1.0사이의 값을 입력한다.

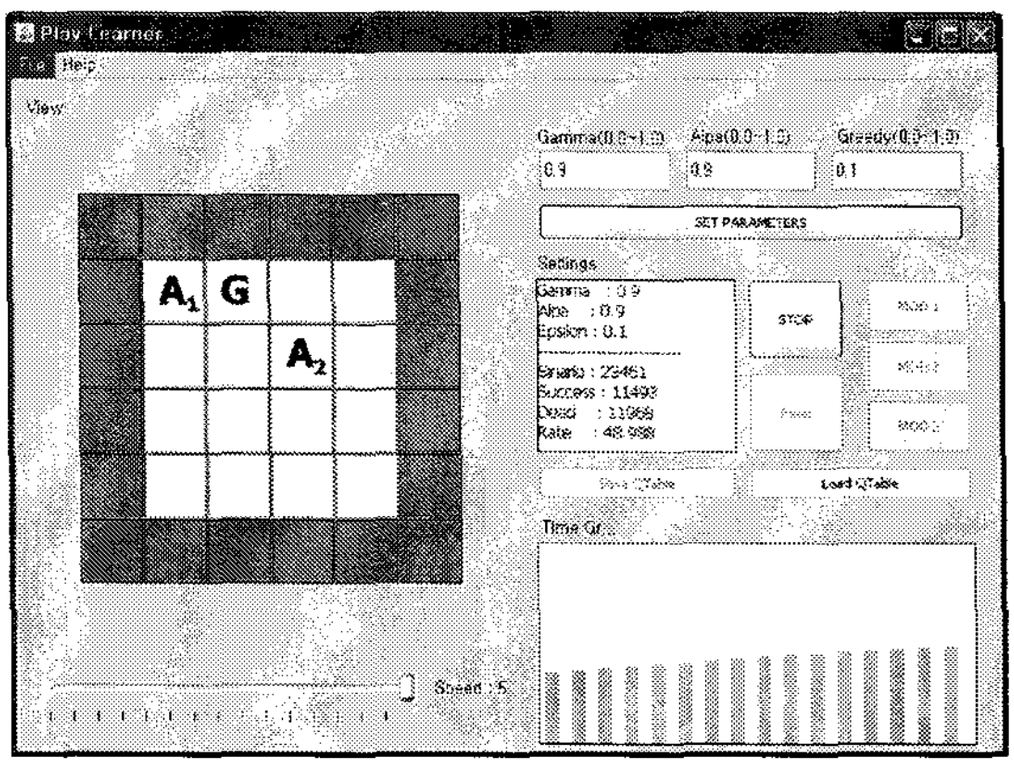

그림 8. 실험 프로그램 인터페이스

각각의 설정값은 파라미터 설정 버튼을 눌러 적용시킨다. 마지막으로 실행 버튼을 누르면 실험이 시작된다. 실험 중에는 학습 수행 속도를 조절하면서 실험 진행 결과를 볼 수 있다. 속도값을 최대값인 5로 설정하면 화면이 갱신되지 않고 빠르게 학습이 진행된다.

구현 소스는 그림 9에서 의사코드로 기술한다. 에이전트가 행동을 취하고 Q-값을 갱신하기 위해서 메인 함수는 일정한 간격으로 NashQ_LearningUsing MacroActions 함수를 호출한다. 이 함수는 에이전트가 행동을 선택하고 수행한 후 자신의 Q-값을 갱신하는 기능과 다른 에이전트의 행동을 예측하고 이에 대한 결과를 Q-테이블에 갱신하는 기능으로 구성된다. Q-값을 갱신하기 위해서는 내시 보상값과 에이전트의 기본 행동 혹은 매크로 행동의 보상값을 선행적으로 계산해야 한다. 에이전트의 행동을 얻는 getAction함수는 현재 상태에서 가장 큰 Q-값을 가지는 행동을 찾는다. Q-값을 갱신하는 updateQ Function함수는 내시 보상값과 행동 보상값을 받아 식(8)처럼 갱신한다.

4.3 실험 결과

이 장에서는 실험에서 각 단계별로 얻은 성공률, 행동 선택 횟수 그리고 Q-값의 변화를 도표로 나타내고 각각의 그래프가 의미하는 바를 설명한다.

4.3.1 성공률 비교

실험을 통해서 가장 먼저 두 개의 에이전트가 목표에 진입한 성공률을 그림 10에 표현한다. 그림 10을 보면 매크로 행동을 네 개 더 이용해서 학습한 쪽이 행동 네 개만 이용한 학습 보다 약 4.82% 높은 성공률을 보였다. 매크로 행동을 이용할 때 내시 Q-학습 효과가 향상된다. 전체 성능의 결과를 볼 때도 매크로 행동을 이용할 때가 더 좋은 성능을 보여 주고 있다. 실험을 거듭할수록 성공 횟수가 점점 벌어진다. 매크로 행동을 네 개 더 이용해서 학습한 쪽이 기본 행동만 이용할 때보다 40만 번 실험할 때 약 9.46% 성공률이 높다.

매크로 행동을 이용한 두 개의 내시 Q-학습의 결과를 보면 더 많은 매크로 행동을 이용할 때 더 좋은 성능을 보인다. 4만 번 실험부터 40만 번 실험까지 매크로 행동의 개수와 성능 향상 정도의 상관계수 값이 0.94~0.96의 값을 보이므로 아주 밀접한 관련이

```
FUNCTION NashQ_LearningUsingMacroActions
  FOR each agent in multi-agent system
    CALL selectAction with agentIndex
    RETURNING action
    CALL doAction with agentIndex, action
    CALL getNashRewards with agentIndex,
    RETURNING nashRewards
    CALL getActionRewards with agentIndex,
    RETURNING actionRewards
    CALL updateQFunction with nashRewards,
    actionRewards

    FOR other agent in multi-agent system
      CALL getAction with otheragentIndex
      RETURNING otherAction
      CALL getNashRewards
        with otherAgentIndex
      RETURNING nashRewards
      CALL getActionRewards
        with otherAgentIndex
      RETURNING actionRewards
      CALL updateQFunction
        with otherAgentIndex, nashRewards,
        actionRewards
    END FOR

END FOR

END FUNCTION

FUNCTION selectAction with agentIndex
RETURNING action

  Call random RETURNING temp;

  IF temp < Epsilon THEN
    CALL getRandomAction RETURNING action
  ELSE
    CALL getAction with agentIndex
    RETURNING action
  END IF

END FUNCTION

FUNCTION getAction with agentIndex
RETURNING action

  SET tempQValue to 0

  FOR each agentAction in QTable
    CALL getQValue with agentAction
    RETURNING QValue
    IF tempQValue > QValue THEN
      SET action to agentAction
      SET tempQValue to QValue
    END IF
  END FOR

END FUNCTION

FUINCTION doAction with agentIndex, action

  IF action = MACRO_NORTH THEN
    Call moveMacroNorth
ELSE IF action = MACRO_SOUTH THEN
  Call moveMacroSouth
ELSE IF action = MACRO_WEST THEN
  Call moveMacroWest
ELSE IF action = MACRO_EAST THEN
  Call moveMacroEast
ELSE IF action = NORTH THEN
  Call moveNorth
  ELSE IF action = SOUTH THEN
    Call moveSouth
  ELSE IF action = WEST THEN
    Call moveWest
  ELSE IF action = EAST THEN
    Call moveEast
  END IF

END FUNCION

FUINCTION getNashRewards with agentIndex
RETURNING nashRewards

FOR each agent in multi-agent system
    FOR other agent in multi-agent system
      CALL calculateNashValue
        with agentIndex, otheragentIndex
        RETURNING nashRewards
    END FOR
END FOR

END FUNCION

FUINCTION getActionRewards with agentIndex
RETURNING actionRewards

  CALL getAgentPosition with agentIndex
  RETURNING agentPosition
  CALL getOtherAgentPosition with agentIndex
  RETURNING otherAgentPosition
  CALL getGoalPosition with agentIndex
  RETURNING goalPosition

  IF agentPosition = otherAgentPosition AND
    agentPosition = goalPosition THEN
    actionRewards = 250
  ELSE IF agentPosition = goalPosition THEN
    actionRewards = -250
  ELSE IF otherAgentPosition = goalPosition THEN
    actionRewards = -250
  ELSE
    actionRewards =0
  END IF

END FUNCION

FUNCTION updateQFunction
  with nashRewards, actionRewards

  SET QValue = ( 1 - alpha ) * QValue
    + alpha * (actionRewards + beta * nashRewards);

  Call setQValue with QValue

END FUNCTION
```

그림 9. 실험 구현 의사 코드

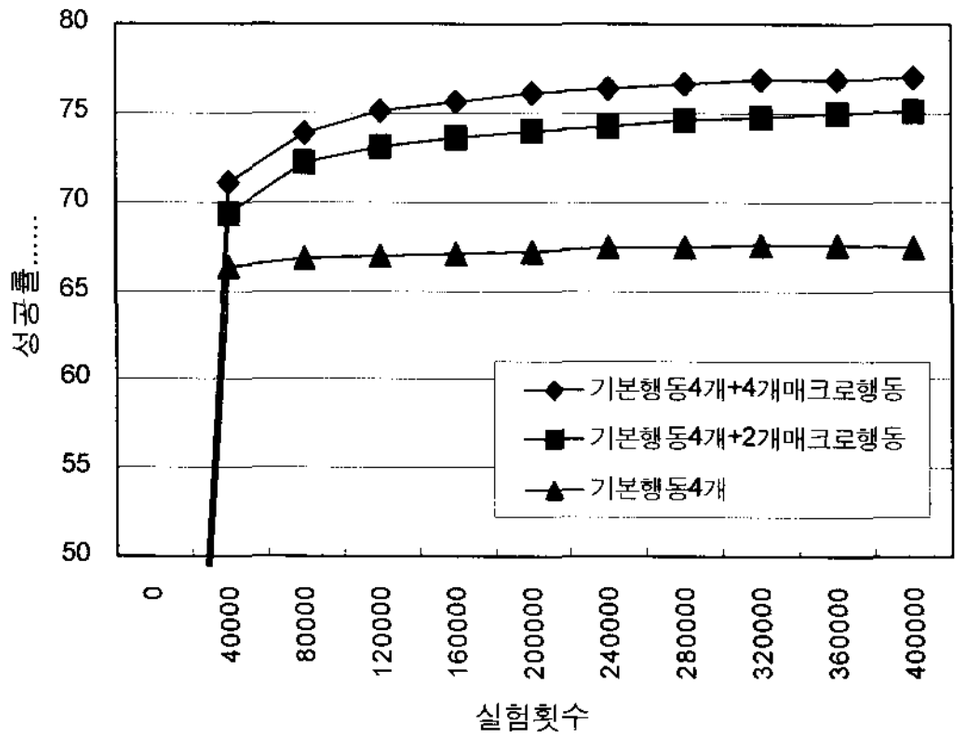

그림 10. 성공률 비교

있음을 알 수 있다.

4.3.2 행동 선택 횟수 비교

성공률을 통해서 에이전트가 목표를 달성하는 정도를 평가했다면 행동 선택 횟수는 목표를 달성하기 위한 과정이 얼마나 빠르고 효율적인지를 평가한다. 에이전트의 모든 행동은 이동 행동으로 구성되기 때문에 행동 선택 횟수는 이동 횟수와 비례한다. 그림 11은 에이전트가 목표를 달성할 때까지 선택하는 평균 행동 선택 횟수를 나타낸다.

그림 11을 보면 학습이 이루어질수록 두 가지 학습 방법 모두 행동을 선택하는 횟수가 줄어든다. 처음 5000번 학습할 때 매크로 행동을 이용한 학습이 그렇지 않은 경우에 비해서 평균 25.28번 더 많이 행동을 선택한다. 하지만 5만 번 학습이 이루어진 이후에 행동 선택 횟수를 보면 매크로 행동을 이용할 때 가 그렇지 않은 경우에 비해 더 적게 12.36번 행동을 선택한다. 매크로 행동을 이용할 때는 에이전트가 취하는 행동의 개수가 더 많아져서 동일한 횟수에 학습할 때 처음에는 학습 효과가 더 낮지만 학습의 효과가 일반적인 내시 Q-학습에 비해 좋아서 만 번 이상 학습할 때부터는 행동 선택의 횟수가 줄어드는 것을 볼 수 있다.

만약에 에이전트가 행동을 선택하는 과정이 복잡하고 시간이 많이 소요된다면 매크로 행동을 이용할 때 더 적게 행동을 선택하게 함으로써 추가적인 학습 시간의 단축효과도 기대된다.

4.3.3 Q-값의 변화 비교

마지막으로 학습되는 과정을 추측하기 위해서 학습이 이루어지는 동안 발생되는 모든 Q-값의 합을 관찰한다. 그림 12는 5000번마다 모든 Q-값을 더한 값을 측정한 그림이다.

그림 12를 보면 매크로 행동을 이용하는 경우에는 Q-값이 지속적으로 변화한다. 하지만 일반적인 내시 Q-학습의 경우에는 일정 수준의 학습이 이루어진 이후에는 더 이상 Q-값의 변화가 발생되지 않는다. 이렇게 매크로 행동을 이용할 때 Q-값이 지속적으로 변화하는 이유는 학습할 때 기본 행동만을 이용해서 적합한 이동 행동을 찾아도 매크로 행동을 이용해서 에피소드를 수행하는 방법을 찾기 때문이다.

5. 결 론

이 논문에서는 내시 Q-학습의 학습 시간을 줄이

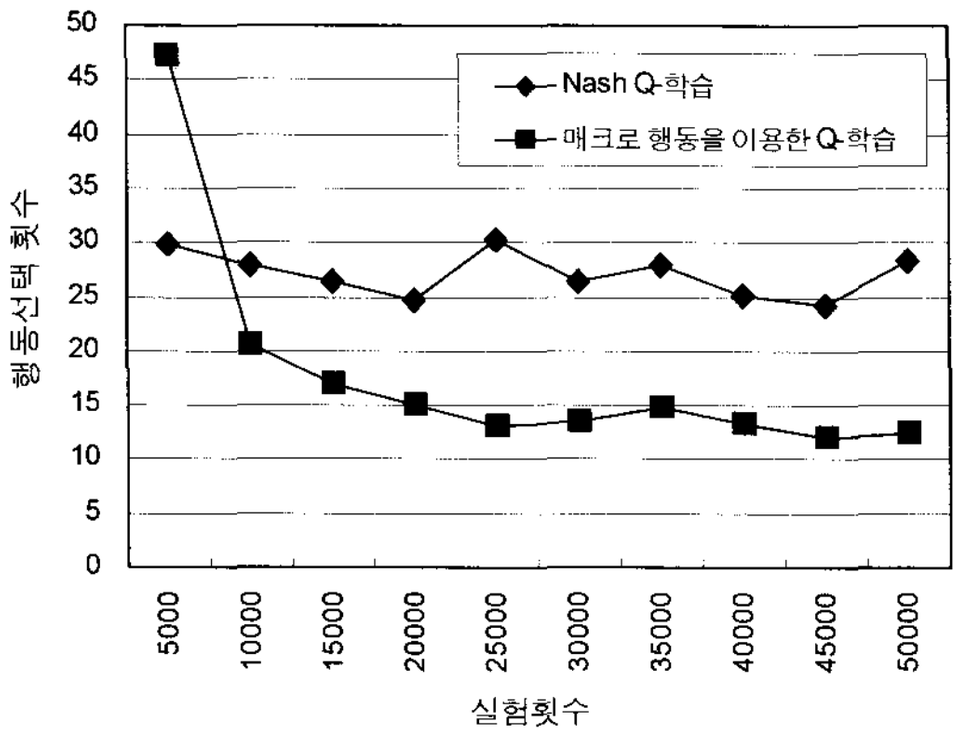

그림 11. 목표 달성하기 위해 수행한 행동 선택 횟수

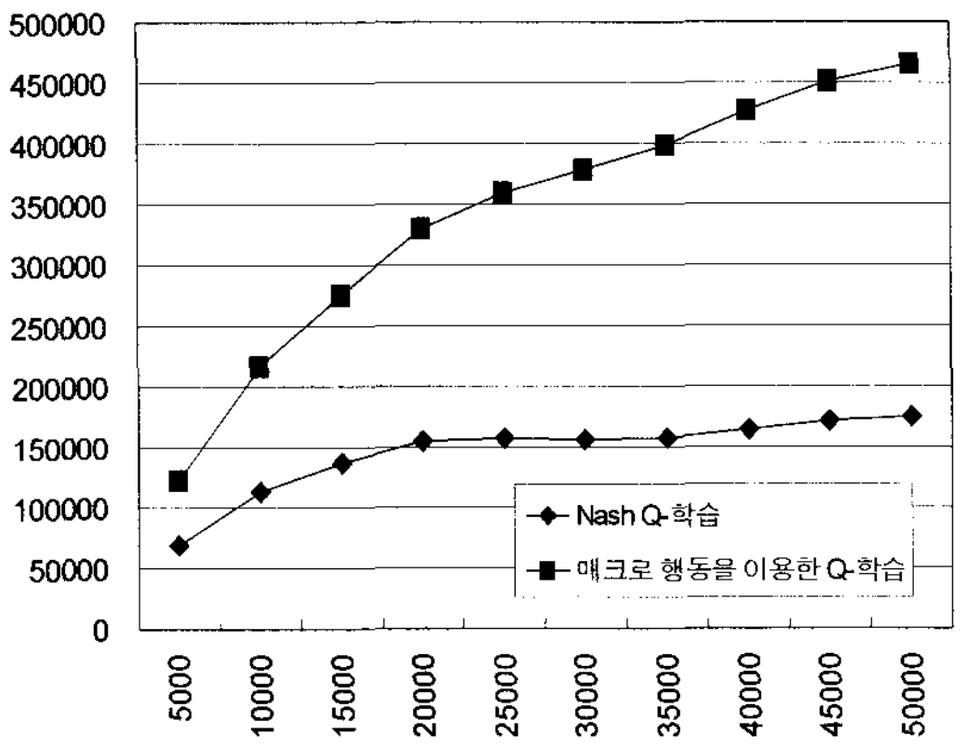

그림 12. Q-값의 변화

고 학습 성능을 높이기 위해서 단일 에이전트 환경에서 사용된 매크로 행동을 사용했다. 성능 평가를 위해서 매크로 행동을 사용하지 않은 내시 Q-학습과 동일한 횟수 동안 실험한 후에 비교하였다. 그리고 목표를 달성하기 위한 과정을 비교해보기 위해서 선택하는 행동 횟수를 비교했다. 실험 결과는 네 개의 매크로 행동을 추가해서 학습한 내시 Q-학습이 약 9.46% 높은 성공률을 보였고 행동선택에 있어서는 절반보다 적은 약 12번의 행동 선택으로 목표 위치에 도달하는 것을 보였다.

단일 에이전트 환경에서는 학습 시간의 차이는 발생되지만 매크로 행동을 사용하지 않은 에이전트의 성공률값이 매크로 행동을 사용한 에이전트의 성공률값에 수렴한다. 하지만 이 논문에서 실험한 다중 에이전트 환경에서는 36만 번 실험부터는 약 0.1%이하의 변화율을 보이며 결과값이 평행선을 나타내고 있다. 결국 다중 에이전트 환경에서 내시 Q-학습을 사용할 때 매크로 행동을 이용한다면 일반 내시 Q-학습에서 얻을 수 없는 향상된 결과를 얻을 수 있다.

매크로 행동을 이용할 때 처음에는 에이전트가 의사 결정해야 할 행동과 매크로 행동이 많아서 목표 지점에 도착할 때까지 많은 행동을 선택한다. 하지만 학습이 진행되면서 매크로 행동을 이용한 내시 Q-학습의 경우에 행동을 선택하는 횟수가 일반적인 Q-학습에 비해 적게 선택하는 것을 알게 되었다. 기본 이동 행동으로 학습한 경우에도 매크로 행동에 의해서 대치되기 때문이다. 행동을 선택하는 횟수가 줄어드는 것은 에이전트의 전체 행동이 이동 행동으로만 구성되었기 때문에 목표를 이루기 위한 이동이 짧아지는 것을 의미한다. 그리고 매크로 행동의 성능 향상 원인을 파악하기 위해서 다양하게 관찰한 결과 중에서 Q-값을 비교할 때 매크로 행동을 이용한 에이전트는 Q-값을 갱신할 때 하나의 행동이 아닌 매크로 행동을 구성한 모든 행동의 보상값을 적용하기 때문에 전체 Q-값 합이 2.6배 높은 수치를 보였다. 그리고 Q-값이 지속적으로 변화한다는 것은 그 만큼 더 좋은 행동 의사 결정 방향으로 가기 때문이다. 결국 매크로 행동을 이용할 때 일반적인 내시 Q-학습에 비해서 좋은 성과를 얻을 수도 있었다.

내시 Q-학습은 다중 에이전트 환경에서 Q-학습을 적용하기 위한 방향을 제시했다. 이 논문에서는 내시 Q-학습의 단점으로 지적되었던 학습 속도 문제 해결 방향을 제시함으로써 많은 학습 시간을 필요로 했던 게임과 같은 다중 에이전트 환경에서 내시 Q-학습을 사용하기 위한 연구의 첫 단계가 되었으면 한다.

참 고 문 헌

[1] S. R. Safavian and D. Landgrebe, "A Survey of Decision Tree Classifier Methodology," *IEEE Transactions on Systems, Man, and Cybernetics*, Vol.3, pp. 660-674, 1991.

[2] C. J. C. H. Watkins, "Learning from delayed rewards," *Ph.D thesis, Cambridge University, Cambridge, U.K.*, 1989.

[3] M. Wooldrigde, "An Introduction to MultiAgent Systems," *Wiley*, 2002.

[4] 정석일, 이연정, "분포 기여도를 이용한 퍼지 Q-learning," *퍼지 및 지능시스템학회 논문지* 제11권 제5호, pp. 388-394, October 2001.

[5] 성연식, 조경은, "영향력 분포도를 이용한 Q-학습," *멀티미디어학회 논문지* 제9권 제5호, pp. 649-657, May 2006.

[6] A. McGovern, R. S. Sutton and A. H. Fagg, "Roles of Macro-Actions in Accelerating Reinforcement Learning," *Proceedings of the 1997 Grace Hopper Celebration of Women in Computing*, pp. 13-17, 1997.

[7] R. Makar, S. Mahadevan, and M. Ghavamzadeh, "Hierarchical Multi-Agent Reinforcement Learning," *Proceedings of the Fifth International Conference on Autonomous Agents*, pp. 246-253, 2001.

[8] G. Tesauro, "Extending Q-Learning to General Adaptive Multi-Agent Systems," *Advances in Neural Information Processing System*, Vol. 16, pp. 871-878, 2003.

[9] J. Hu and M. P. Wellman, "Nash Q-learning for General-Sum Stochastic Games," *Journal of Machine Learning Research*, Vol.4, pp. 1039-1069, 2003.

성 연 식

1998년~2001년 케이원시스템
2004년~2005년 (주) 포켓스페이스
2004년 8월 부산대학교, 전자전기정보컴퓨터공학(공학사)
2006년 8월 동국대학교, 컴퓨터공학과 대학원(공학석사)
관심분야 : 다중 에이전트 시스템, 학습, 전문가 시스템, 게임 이론

조 경 은

1993년 2월 동국대학교,전자계산학과(공학사)
1995년 2월 동국대학교, 컴퓨터공학과 대학원(공학석사)
2001년 8월 동국대학교, 컴퓨터공학과 대학원(공학박사)
2003년 9월~2005년 8월 동국대학교 정보산업대학 컴퓨터멀티미디어공학과 전임강사
2005년 9월~2007년 6월 동국대학교 정보산업대학 컴퓨터멀티미디어공학과 조교수
2007년 7월~현재 동국대학교 영상미디어대학 게임멀티미디어공학과 조교수
관심분야 : 컴퓨터 게임 알고리즘, 게임 인공지능, 멀티미디어 정보처리

엄 기 현

1975년 2월 서울대학교 공과대학 응용수학과 공학사
1977년 2월 한국과학기술원 전산학과 이학석사
1994년 2월 서울대학교 대학원 컴퓨터공학과 공학박사
1978년 3월~2007년 6월 동국대학교 컴퓨터멀티미디어공학과 정교수
2007년 7월~현재 동국대학교 영상미디어대학 게임멀티미디어공학과 교수
1995년 3월~1999년 2월 동국대학교 정보관리처장 역임
2001년 3월~2003년 2월 동국대학교 정보산업대학 학장 역임
2005년 3월~현재 한국 게임학회 자문위원
1998년 12월~2001년 12월 한국 멀티미디어학회 부회장, 자문위원, 수석부회장 역임
2007년 1월~2007년 12월 한국멀티미디어학회 회장
관심분야 : 게임시스템 및 구조 설계, 멀티미디어응용시스템, 멀티미디어데이터베이스