RTS 게임에서 에이전트와 상호 의사를 조절하는 조정 에이전트의 설계

박진영, 성연식*, 조경은[1]**, 엄기현**
동국대학교 멀티미디어학과, 동국대학교 게임공학과*,
동국대학교 게임멀티미디어공학과**
nowgo@paran.com, {sung, cke, khum}@dongguk.edu

A Design of a Coordination Agent Controlling Decision with Each Other Agents in RTS

Jinyoung Park, Yunsick Sung*, Kyungeun Cho**, Kyhyun Um**
Dept. of Multimedia, Dongguk University,
Dept. of Game Engineering, Dongguk University*
Dept. of Game and Multimedia Engineering, Dongguk University**

요 약

실시간 전략 시뮬레이션 게임에서 각각의 팀은 다수의 에이전트로 구성하고 상대 팀을 이기기 위한 전략을 수행한다. 전략은 팀에 속한 에이전트들의 협력을 필요로 하며 이를 위해서는 다중 에이전트 시스템이 필요하다. 다중 에이전트 시스템의 의사 결정방법 중에서 중앙 집중적인 의사 결정 방법은 조정 에이전트를 사용해서 팀을 위한 작업을 선택한다. 비 중앙 집중적인 의사 결정 방법은 에이전트가 각각 주체가 되어서 다른 에이전트와 의사소통을 하기 때문에 비용이 많이 든다. 본 논문에서는 조정 에이전트를 사용할 때 다수의 에이전트를 그룹으로 관리하는 방법, 경매 시스템을 사용해서 에이전트에게 작업을 할당하는 방법 그리고 할당한 작업 중에서 수행에 실패한 작업은 다른 에이전트에게 다시 할당방법을 제안한다. 실험에서는 스타크래프트 게임에서 제안한 시스템을 적용할 때 공격력과 방어력이 향상되는 것을 볼 수 있었다. 제안한 방법을 사용한 에이전트의 팀 승리 비율은 8대 2로 높아졌다.

ABSTRACT

In real-time strategy simulation (RTS) game each team is composed of agents and executes strategies to win other team. Strategy needs agents' cooperation in each team. This needs multi-agent system (MAS). Centralized decision making, one of decision making in MAS, selects actions not to agents but to team by a coordinated agent. Decentralized decision making costs high because each agent communicates with each other. In this paper we propose a system which controls agents by grouping and allocates roles through negotiation by a coordinated agent. Then, when one of allocated actions is not executed or failed, a coordinated agent allocates its role to another agent. We make experiments in starcraft, famous RTS game. When a proposed method is applied, the performance of attack and defense is increased. The improved agents' team wins eight times per ten games.

Keyword : multi-agent system, RTS, a coordinated agent

접수일자 : 2009년 06월 02일
일차수정 : 2009년 08월 07일
심사완료 : 2009년 09월 14일

1) 교신저자(Corresponding Author) : 조경은, 주소 : 서울시 중구 필동 3가 26 동국대학교(100-715)
전화 : 02)2260-3834, E-mail : cke@dongguk.edu

1. 서 론

실시간 전략 시뮬레이션(RTS) 게임에서 각각의 팀은 상대 팀을 이기기 위해 현재 상황에 맞는 최적의 전략을 선택한다. 팀은 다수의 에이전트로 구성하고 에이전트는 선택한 전략을 달성하기 위해서 같은 팀에 속한 다른 에이전트와 협력한다[1]. RTS 게임에서는 에이전트가 많은 정보를 빠르게 인지하고 전략을 결정하는 것이 중요하다. 그리고 전략을 달성하기 위해서 각각의 에이전트가 수행할 행동을 선정하는 방법이 필요하다.

Tambe는 다중 에이전트 시스템에서 같은 목표를 가지고 있는 에이전트의 행동을 협력이라 정의했다[2]. Nobuhiro는 에이전트들이 특정한 목표를 달성하기 위해서 도움이 필요하다는 것을 인지하고 성취할 수 있도록 서로 도와주는 것을 협력이라고 정의했다[3]. 에이전트가 혼자서 문제를 해결할 수 없는 상황일 경우에는 다른 에이전트의 도움으로 해결할 수 있다.

전략을 수행할 때 다수의 에이전트가 동시에 개별 작업을 선택하기 위해서는 에이전트 간에 수행할 작업을 조율하는 방법이 필요하다. 에이전트가 수행할 작업을 결정하는 방법은 중앙 집중적인 의사 결정 방법과 비 중앙 집중적인 의사 결정 방법이 있다[4]. 모든 에이전트들의 의견을 수렴하여 최적의 작업을 선정하는 중앙 집중적인 의사 결정 방법은 각각의 에이전트보다 전체 에이전트를 위한 작업을 결정한다. 비 중앙 집중적인 의사 결정 방법은 각각의 에이전트가 모든 에이전트와 의사소통을 하고 작업을 결정하기 때문에 비용이 많이 발생한다. 다중 에이전트 환경에서 발생하는 정보는 많기 때문에 모든 정보를 다른 에이전트와 공유하는 것은 어렵다. 에이전트의 작업을 결정하는 방법은 중앙 집중적인 결정 방법보다 각각의 에이전트를 배려하고 비 중앙적인 결정 방법보다 의사소통의 비용을 낮출 수 있는 시스템이 필요하다.

제안하는 시스템(CO-시스템)은 조정 에이전트를 사용할 때 에이전트를 그룹으로 관리해서 의사소통의 비용은 낮추고 경매 시스템을 사용해서 에이전트에게 작업을 배정하는 방법을 제안한다. 이 시스템은 RTS 게임과 같이 다수의 에이전트를 그룹으로 관리할 때 적합하다.

본 논문의 구성은 다음과 같다. 2장에서는 조정 에이전트에 관련된 연구를 살펴본다. 3장에서는 협력 시스템을 제안한다. 4장에서는 실험방법을 소개하고 실험 결과를 분석한다. 마지막으로 5장에서는 이 연구를 통해 얻은 결론을 도출하고 향후 연구를 서술한다.

2. 관련연구

이 장에서는 중앙 집중적인 의사 결정 방법 중에서 에이전트 간에 수행할 행동을 조율하기 위한 방법들을 살펴본다.

한 개의 에이전트로 해결하지 못하는 복잡한 문제는 여러 에이전트간의 협력을 통해서 해결한다[5]. 조정 에이전트는 다중 에이전트 시스템에서 두 개 이상의 응용 에이전트를 조절한다[6]. 조정 에이전트는 다른 에이전트가 보이는 반응과 정보 등을 수집하고 분석하여 전체 시스템의 효율이 최대화되도록 에이전트의 기능과 역할을 조정한다.

조정 에이전트를 사용하는 연구 중에서 위급한 상황이 발생할 때 동적으로 팀을 구성하는 시스템이 있다[3]. 에이전트는 개별적으로 행동하다가 문제를 발생하면 스스로 해결할 것인지 다른 에이전트에게 도움을 요청할 것인지 판단한다. 에이전트가 스스로 해결하지 못하는 상황이 발생하면 에이전트가 조정자가 되어 근접에 위치한 에이전트에게 순차적으로 도움을 요청한다. 요청을 받은 에이전트가 동의하면 같은 팀을 이루어 문제를 해결한다. 문제를 해결하면 팀이 해산되고 초기상태로 돌아간다. 동적으로 팀을 구성하는 방법은 조정 에이전트의 수행이 불가능할 때 팀의 목표를 수행하지 못하는 문제가 발생한다. 그리고 에이전트가 독립적으로 조정 에이전트 여부를 결정하기 때문에 다수

의 조정 에이전트가 발생할 수도 있다.

조정 에이전트가 수행할 행동을 조율하는 방법에는 다음과 같은 방법이 있다. 탐욕 알고리즘은 빠르게 작업을 할당하지만 최적의 의사 결정을 보장하지 않는다[7]. A* 알고리즘을 사용해서 에이전트에게 작업을 할당하는 방법은 최적의 의사 결정을 보장하지만 할당 시간이 길어지는 문제가 있다[8]. 각각의 에이전트가 투표해서 의사를 결정하는 방식은 빠르면서 불확실한 환경에 적합하다[9]. 마지막으로 경매 방식은 여러 에이전트의 의견 중에서 하나의 에이전트가 결정한 방법을 채택하는 방식이다[10].

조정 에이전트가 할당한 작업은 정상적으로 수행되고 있는 지 관찰이 필요하고 수행 중 문제가 발생되면 다시 할당하는 작업이 필요하다. 그리고 다수의 에이전트가 같은 팀에 있을 때 그룹으로 나누어 관리하고 유사한 작업을 수행하는 에이전트는 묶어서 효율적으로 구성하는 방법이 필요하다.

제안한 시스템은 조정 에이전트가 정보를 수집하는 방법, 다수의 에이전트가 있을 때 그룹으로 관리하는 방법, 경매 방식을 통해서 작업을 수행할 에이전트를 선정하는 방법 그리고 수행 중인 작업이 실패해도 조정 에이전트가 다시 작업을 할당하는 방법을 소개한다.

3. 에이전트 의사를 상호 반영하기 위한 시스템

이 장에서는 CO-시스템을 정의하고 에이전트가 요청한 행동을 다른 에이전트가 수행하기까지의 과정을 기술한다. 이 시스템은 에이전트가 인지하는 환경 범위에 제한이 있을 때 다른 에이전트와 상호 의사를 조절하기위한 방법을 제안한다.

3.1 시스템 정의

단일 에이전트 환경에서 에이전트의 의사 결정 과정은 3-튜플 <S, A, D>로 정의할 수 있다. S는 단일 에이전트 환경에서 발생 가능한 유한한 상태 s의 집합이다. A는 에이전트가 수행 가능한 유한한 행동 a의 집합이다. 그리고 D는 상태 s에서 수행할 행동 a를 결정하는 함수이다. 다중 에이전트 환경은 에이전트가 다른 에이전트의 상태도 고려하기 때문에 에이전트 집합 M을 추가해서 4-튜플 <M, S, A, D>로 정의할 수 있다.

CO-시스템에서 다수의 에이전트는 그룹으로 묶어서 관리한다. 그룹 m_g는 한 개 이상의 에이전트를 포함하는 집합이다. M_g는 그룹 m_g의 전체 집합이다. 모든 에이전트는 한 개 그룹에 속한다. 그리고 다수의 그룹은 묶어서 팀으로 관리한다. 팀 m_t는 1개 이상의 그룹 m_g를 포함하는 집합이다. M_t는 그룹 m_t의 전체 집합이다. 모든 그룹은 한 개 팀에 속한다.

CO-시스템은 에이전트를 일반 에이전트(C-에이전트)와 조정 에이전트(L-에이전트)로 구분한다. C-에이전트는 의사 결정을 할 때 환경의 상태를 인식하고 L-에이전트에게 상태 정보 sa를 전달해야하는 지 결정하는 함수 D_s, L-에이전트가 요청한 작업 a_t를 수행할지 결정하는 함수 D_t 그리고 작업 a_t를 수행할 일렬 행동 a로 변환하는 함수 D_x가 필요하다. 상태 정보 sa의 전체 집합은 Sa라고 정의하면 C-에이전트의 의사 결정 과정은 8-튜플 <M, S, A_t, A, D_s, D_t, D_x, D>로 정의할 수 있다. L-에이전트는 C-에이전트로부터 받은 상태 s_a를 가지고 C-에이전트에게 할당할 작업 at를 결정하는 함수 D_d, 그룹을 선정하는 함수 D_g 그리고 에이전트를 선정하는 함수 D_a가 필요하다. 함수 D_d는 그룹과 팀을 알아야하기 때문에 L-에이전트의 의사 결정 과정은 9-튜플 <M_t, M_g, M, S, A_t, A, D_d, D_g, D_a>로 정의할 수 있다. 한 개의 팀에서는 1개의 L-에이전트를 정의한다. CO-시스템은 설계에 따라서 L-에이전트와 C-에이전트를 합쳐서 하나의 에이전트로 구성할 수도 있고 L-에이전트를 가상의 에이전트로 정의할 수도 있다.

3.2 작업 수행 과정

제안한 CO-시스템은 C-에이전트가 현재 상태 s를 인지하고 L-에이전트에 상태 s_a를 전달한다. L-에이전트는 받은 상태를 사용해서 다른 C-에이전트에게 작업 a_t를 할당한다. 이 절에서는 작업 a_t가 수행되기까지 각 단계에서 처리하는 과정을 기술한다.

첫 번째, C-에이전트는 상태 s를 인지하고 함수 D_s를 사용해서 현재 상태 s에서 L-에이전트에게 전달할 상태 s_a가 있는 지 판단한다. L-에이전트에 전달이 필요하면 인지한 상태 s에서 추출한 상태 s_a를 L-에이전트에 전송한다. 만약에 C-에이전트가 전송할 상태 sa가 없으면 상태 s에서 수행할 행동 a를 함수 D를 사용해서 결정한다.

두 번째, L-에이전트는 함수 D_d와 C-에이전트로부터 받은 상태 s_a을 사용해서 C-에이전트가 수행할 작업 a_t를 계산한다.

세 번째, 작업 a_t를 수행할 C-에이전트는 함수 D_a와 함수 D_g를 사용해서 찾는다. 수행할 C-에이전트를 찾기 위해서 같은 그룹 g에 속하면서 수행이 가능한 다른 C-에이전트를 함수 D_a를 사용한다. 만약에 같은 그룹에 있는 C-에이전트가 수행이 불가능하다면 적합한 그룹 g를 함수 D_g로 찾는다. 다른 그룹 g에 있는 C-에이전트는 함수 D_a로 찾고 찾은 C-에이전트는 작업을 요청한 C-에이전트의 그룹으로 편성한다. C-에이전트가 수행이 가능한지는 L-에이전트가 C-에이전트를 함수 D_a를 사용해서 결정한다.

네 번째, C-에이전트가 작업 a_t을 수행할 수 있는 지 판단한다. C-에이전트는 함수 D_c를 사용해서 L-에이전트로부터 받은 작업 a_t의 수행 여부를 결정한다. C-에이전트가 작업 a_t를 수행하지 못하면 L-에이전트는 수행할 C-에이전트를 다시 찾는다. 이 과정을 통해서 L-에이전트와 C-에이전트는 작업 a_t의 수행 여부를 두고 협상한다.

마지막으로 C-에이전트가 작업 a_t를 수행하기로 결정하면 받은 작업 a_t를 함수 D_x를 사용해서 일련의 행동 a를 생성한다. 그리고 생성한 행동들은 구동부에 보내서 수행한다. C-에이전트는 할당받은 작업 at를 수행하기로 결정했지만 완수하지 못하는 상황이 발생하면 L-에이전트에게 다른 C-에이전트가 수행하도록 요청한다.

CO-시스템은 각각의 C-에이전트가 인지한 상태 s를 다른 C-에이전트와 공유하지는 않지만 인지한 상태 s_a를 L-에이전트에게 전달함으로써 협력적인 작업을 가능하게 한다. 협력 시스템 구현을 위해서는 에이전트가 스스로 해결하지 못하는 상황을 인식하고 도움을 요청하는 것이 중요하다. 그러나 요청을 받은 에이전트들의 현재 상태와 에이전트의 의사를 고려하지 않는다면 전체 에이전트에게 오히려 해가 될 수 있다.

3.3 의사 결정 방법

L-에이전트는 작업 a_t를 수행할 C-에이전트를 선정하기 위해서 그룹을 함수 D_g를 사용해서 선택한다. 함수 D_g는 식(1)과 같이 그룹간의 거리 값을 계산하여 가장 낮은 그룹을 선정한다. G^m_x와 G^m_y는 m번째 그룹의 X-Y 위치이고 G^n_x와 G^n_y는 n번째 그룹의 X-Y 위치이다.

$$D_g = \sqrt{(G^{m_x} - G^{n_x})^2 + (G^{m_y} - G^{n_y})^2} \quad \cdot\cdot \text{식(1)}$$

거리만 고려하면 도움을 요청한 그룹보다 위험한 상황에 놓인 그룹에게 지원을 요청하는 경우가 발생한다. 이를 방지하기 위해서 각 에이전트의 위험 정도 S_{ai}를 사용해서 각 그룹의 위험 정도 R_t을 식(2)와 같이 구하고 임계값 a_r보다 높은 그룹은 제외한다.

$$R_t = \frac{\sum_{i=1}^{k} S_{a_i}}{K} \quad \cdots\cdots \text{식(2)}$$

식(2)에서 그룹의 위험 정도 R_t는 그룹에 속한 각각 에이전트의 위험 정도을 그룹의 에이전트 개

수 K로 나누어 계산한다. 에이전트의 위험 정도 S_{ai}는 에이전트의 각 상태마다 양수로 정의해서 사용한다.

함수 D_a는 그룹에 속한 에이전트 중에 작업을 수행할 에이전트를 선정한다. 함수 D_a는 경매방식을 사용한다. 그룹이 선정되면 L-에이전트는 C-에이전트와 입찰가를 계산하여 가장 최소의 비용을 소비하는 에이전트를 선별한다. 입찰가 N은 식(3)과 같이 구한다.

$$N = D_w(S_{a_i}) + D_{a_i} + T_{a_i} \quad \cdots\cdots \text{식(3)}$$

입찰가 N은 에이전트의 위험 정도 S_{ai}에 가중치를 적용한 값, 에이전트의 거리 D_{ai}, 그리고 작업의 중요도 T_{ai}를 더해서 계산한다. 에이전트의 위험 정도 S_{ai}는 에이전트가 인지한 상태의 위험 정도에 따라 수치가 높아진다. 작업의 중요도 T_{ai}는 각 작업마다 양수로 정의한다. 값은 현재 수행 중인 행동의 중요한 정도에 따라 수치가 높아진다. 에이전트 선정에 필요한 세 개의 값 중에서 에이전트의 위험 정도 S_{ai}이 가장 중요하기 때문에 함수 D_w를 사용해서 가중치를 적용한 값을 사용한다. 함수 D_w와 에이전트의 의사 결정에 필요한 다른 함수들은 도메인에 따라 다르게 정의한다.

4. 실 험

이 장에서는 CO-시스템의 성능을 알아보기 위해서 진행하는 실험 방법을 소개하고 실험 결과를 분석한다.

4.1 실험 방법

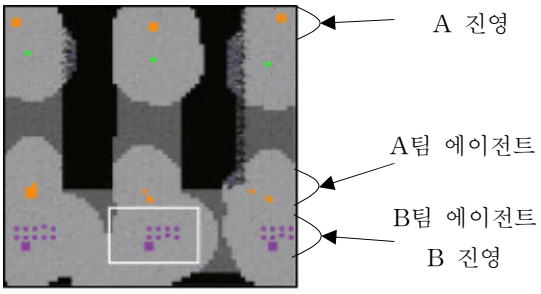
실험에서는 제안한 방법의 성능을 알아보기 위해서 RTS 게임 중에서 스타크래프트를 사용한다. 게임 환경은 다중 에이전트의 성능을 평가하기에 좋은 환경을 제공하기 때문에 다양한 사례를 관찰해 볼 수 있다.

스타크래프트의 게임에서 인공지능 스크립트는 'Primary.mpq' 파일 내에 압축되어 'aiscript.bin' 파일에 정의되어 있다. ScAIEdit 도구를 이용하여 스타크래프트의 인공지능을 수정하고 컴파일한다[11]. 스타크래프트의 인공지능 스크립트에서 11의 유닛을 제어하기 위해서는 스타포지 캠페인 편집기를 사용한다[12]. 스타크래프트의 캠페인 편집기는 제약이 많아서 제안하는 시스템을 실험하기가 어렵다.

[그림 1] 스타크래프트 실험화면

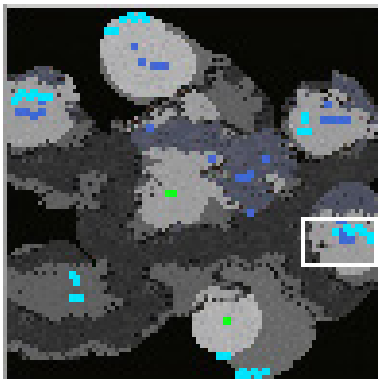
첫 번째 실험에서는 CO-시스템과 각각의 에이전트가 독립적으로 의사를 결정하는 시스템(IDS-System)의 성능을 비교 분석한다. [그림 2]와 같이 게임은 두 개의 팀으로 구성하고 각 팀에는 세 개의 그룹을 정의한다. 위쪽에 진형을 잡은 A팀은 각 그룹에 두 개에서 12개 사이의 C-에이전트를 생성한다. 아래쪽에 진형을 잡은 B팀은 각 그룹마다 10개의 C-에이전트를 생성하고 배치한다. L-에이전트는 A팀과 B팀 모두 가상으로만 존재한다. 게임이 시작되면 A팀의 C-에이전트는 아래쪽으로 이동해서 B팀의 C-에이전트를 공격한다. 게임을 시작하고 매 20초마다 A팀의 각 그룹에는 두 개에서 12개의 C-에이전트를 A 진영에 새로 생성하고 생성한 C-에이전트는 다시 B팀을 공격한다. 실험하는 대상은 B팀이다. 동일한 환경에서 동일한 에

이전트로 구성한 B팀에 CO-시스템을 적용한 경우와 그렇지 않은 경우를 비교한다. 실험은 10번을 반복적으로 진행하고 5분이 경과할 때 남은 B팀의 C-에이전트 개수로 성능을 비교한다. 이 실험을 통해서 C-에이전트가 방어할 때 CO-시스템이 미치는 영향을 알아본다.



[그림 2] 방어 실험 지형 및 에이전트 배치

두 번째, CO-시스템을 적용한 팀과 적용하지 않은 팀의 전투를 발생시켜서 성능을 비교한다. [그림 3]와 같이 두 개의 팀을 정의하고 각 팀은 6개의 그룹을 정의한다. 각각의 그룹은 모두 5개의 C-에이전트를 배정한다. C-에이전트는 두 개 그룹을 묶어서 10개의 C-에이전트를 세 개의 장소에 배치한다.



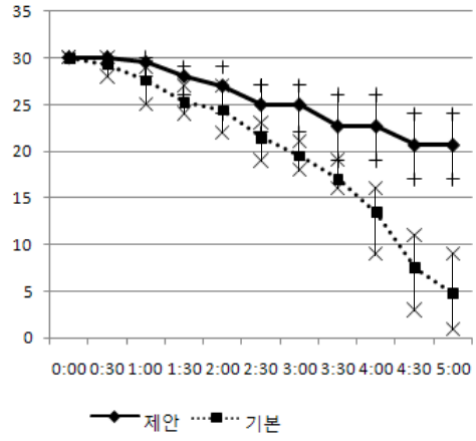
[그림 3] 대전 실험 지형 및 에이전트 배치

게임이 시작하면 세 개의 장소에서 한 그룹씩 지형 가운데로 이동하고 상대 팀을 만나면 전투한다. 가운데에서 살아남은 C-에이전트는 상대 팀으로 이동해서 공격한다. 각 팀의 남은 세 그룹은 세 개의 장소를 지키며 다른 팀의 C-에이전트의 공격으로부터 방어한다. 두 팀의 성능은 동일한 환경에서 10번 진행하고 4분 동안 진행해서 남은 C-에이전트의 개수로 평가한다. 이 실험을 통해서 C-에이전트가 공격할 때 CO-시스템이 미치는 영향을 알아본다.

4.2 실험 결과 및 분석

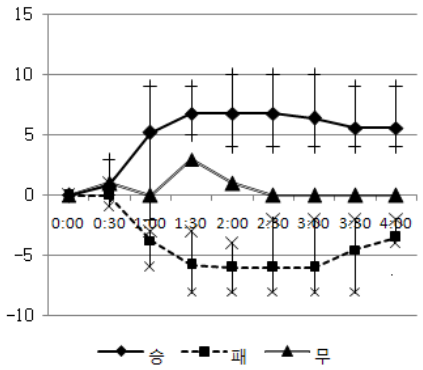
이 절에서는 CO-시스템을 적용한 실험과 IDS-시스템의 실험을 비교한다.

첫 번째는 CO-시스템을 적용할 때 C-에이전트의 방어력에 미치는 영향을 알아본다. 각각의 방법으로 10번씩 실험한다. 그리고 남은 개수를 비교하면 10번 모두 제안한 CO-시스템에서 더 많은 C-에이전트가 남는다. 다중 에이전트 환경에서는 두 팀의 성능 차이가 조금만 나더라도 전투 결과에 많은 영향을 미치기 때문에 이와 같은 결과가 발생한다. 이 실험에서는 두 팀의 성능 차이를 명확히 알기 어렵기 때문에 시간 별 남은 개수를 비교한다.



[그림 4] 방어력 실험 : 시간별 남은 C-에이전트 개수

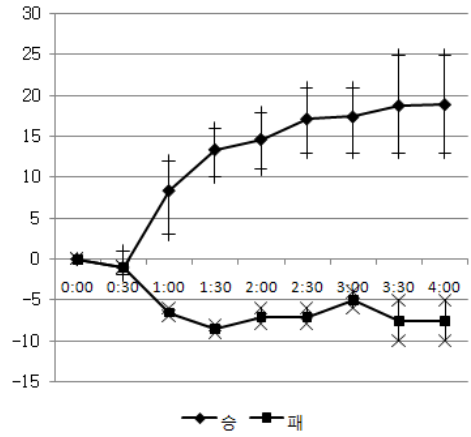
[그림 4]를 보면 시간이 갈수록 IDS-시스템에서 더 많은 C-에이전트가 감소한다. CO-시스템을 적용하면 C-에이전트는 공격해오는 다른 팀의 C-에이전트의 개수가 많을 때 L-에이전트를 통해서 다른 그룹의 C-에이전트에게 도움을 요청하기 때문에 전투를 유리하게 이끌어 나갈 수 있다. 시간에 지날 감에 따라 남은 C-에이전트의 개수가 더 벌어지는 것은 남은 C-에이전트가 많을수록 더욱 유리해지기 때문이다. 방어 실험을 통해서 제안한 CO-시스템이 에이전트가 독립적으로 의사 결정한 IDS-시스템에 비해서 평균 4배정도 많은 C-에이전트가 남는 것을 확인 할 수 있다.



[그림 5] 공격력 실험 1 : 시간별 C-에이전트 차이

두 번째 실험은 CO-시스템이 C-에이전트의 공격력에 미치는 영향을 알아본다. CO-시스템과 IDS-시스템을 비교하기 전에 IDS-시스템끼리 전투를 해서 발생하는 결과를 알아본다. IDS-시스템끼리의 대결에선 승률과 게임종료 후 남은 C-에이전트가 비슷하다. 승률은 5승 4패 1무로 양쪽 비슷한 결과를 보인다. 시간별 C-에이전트 차이에서도 양쪽 모두 이길 때 [그림 5]와 같이 C-에이전트 차이가 비슷한 것을 볼 수 있다.

CO-시스템과 IDS-시스템의 전투에서는 CO-시스템을 적용할 때 8승 2패로 더 낳은 결과를 보인다. [그림 6]에서는 CO-시스템이 승리했을 때 10개 이상의 C-에이전트가 남지만 IDS-시스템을 사용한 경우에는 10개 이하의 C-에이전트가 남았다. 남은 C-에이전트의 차이에서 CO-시스템은 쉽게 승리할 수 있지만 IDS-시스템은 힘들게 승리한다는 것을 알 수 있다. 공격에 미치는 영향을 알아보기 위한 실험에서도 CO-시스템이 같은 환경에서 IDS-시스템보다 좋은 승률을 보인다는 것을 입증한다.



[그림 6] 공력력 실험 2 : 시간별 C-에이전트 차이

실험을 통해서 CO-시스템을 적용하면 공격할 때와 방어를 할 때 모두 성능이 좋아지는 것을 볼 수 있다.

5. 결 론

제안한 CO-시스템에서는 C-에이전트에게 작업을 할당할 때 작업을 그룹단위로 작업이 가능한 에이전트를 찾았다. 그리고 L-에이전트는 C-에이전트에 강제적으로 작업을 할당하지 않고 경매 시스템을 사용하기 때문에 에이전트가 중요한 작업을 수행하고 있어도 중단하는 문제를 해결할 수 있었다. 할당한 행동에 문제가 발생되면 L-에이전트는 다른 C-에이전트에게 작업을 할당할 수 있었다.

실험은 RTS 게임에서 제안한 CO-시스템을 사용할 때 에이전트의 공격력과 방어력에 미치는 영향을 알아보았다. 방어력을 알아보는 실험에서는 CO-시스템이 IDS-시스템과의 전투에서 항상 이겼고 승리할 때 남은 C-에이전트의 개수도 약 4배 정도 많았다. 공격력 실험에서는 제안한 CO-시스템이 8승 2패로 높은 승률을 보였고 남은 C-에이전트의 차이도 많이 벌어졌다.

향후에는 제안한 CO-시스템에서 사용하는 다양한 의사 결정 방법의 성능을 높이기 위한 연구를 진행하면 보다 향상된 시스템을 기대해 볼 수 있을 것이다.

참고문헌

[1] Brian Schwab, AI Game Engine Programming, Charles River Media, 2004.

[2] Milind Tambe, “Towards Flexible Teamwork", Journal of Artificial Intelligence Research, Vol. 7, No.4, 1997.

[3] Nobuhiro Ito, “A cooperative agent model by forming a group", IEEE ICIT, 2002.

[4] 박근수, 권기덕, 김인철, “실시간 다중 에이전트 환경에서 동적 역할 조합과 배정”, 한국정보처리학회 춘계학술발표대회, 제10권, 제1호, 2003.

[5] 송종철, “멀티에이전트 시스템의 연구 동향”, [IITA] 정보통신연구 진흥원 학술정보, 주간기술동향 970호.

[6] Katia P. Sycara, “Multiagent System", AI Magazine, pp.79-92, Summer 1998.

[7] Ranindra K. Ahuja, Thomas L. Magnanti and James B. Orlin, Network Flows: Theory, Algorithms, and Applications, Prentice Hall, UpperSaddleRiver, New Jersey, pp.309-312, 1993.

[8] 박재현, 조경은, 엄기현, “조정 에이전트를 이용한 작업 할당 최적화 기법”, 한국게임학회 논문지, 제7권 제4호, pp.93-104, 2007.

[9] 박근수, 권기덕, 김인철, “실시간 다중 에이전트 환경에서 동적 역할 조합과 배정”, 한국정보처리학회, 춘계학술발표대회 논문집, 제10권, 제1호, pp.329-332, 2003.

[10] 김광수, 김인철, “에이전트 기반의 연속다중경매 시스템의 설계 및 구현”, 정보과학회 논문지, 제7권, 제6호, pp.641-652, 2001.

[11] ScAIEdit, http://www.camsys.org

[12] StarForge, http://game.softpedia.com/get/Tools/Starcraft-StarForge.shtml

박 진 영(Jin young, Park)

2006년 2월 안양대학교 멀티미디어공학사
2009년 2월 동국대학교 영상대학원 멀티미디어학과 예술공학 석사

관심분야 : 게임인공지능, RTS게임

성 연 식(Yun sick, Sung)

2004년 2월 부산대학교 컴퓨터공학 공학사
2006년 2월 동국대학교 일반대학원 컴퓨터공학 공학석사
2008년 3월～현재 동국대학교 일반대학원 게임공학 공학 박사 과정
1998년 1월～2001년 5월 (주)케이원시스템
2004년 4월～2005년 2월 (주)포켓스페이스
2006년 8월～2009년 1월 삼성전자(주)
2009년 2월～2009년 6월 (주)레이큐브
2009년 3월～현재 신흥대학 겸임전임교수

관심분야 : 게임 인공지능, 게임 소프트웨어공학, 게임 알고리즘

조 경 은(Kyun geun, Cho)

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～현재 동국대학교 영상미디어대학 게임멀티미디어공학과 부교수

관심분야 : 컴퓨터 게임 알고리즘, 게임 인공지능

엄 기 현(Ky hyun, Um)

1975년 2월 서울대학교 공과대학 응용수학과 공학사
1977년 2월 한국과학기술원 전산학과 이학석사
1994년 2월 서울대학교 대학원 컴퓨터공학과 공학박사
1978년 3월～2007년 6월 동국대학교 컴퓨터멀티미디어공학 교수
2007년 7월～현재 동국대학교 영상미디어대학 게임멀티미디어공학과 교수
2009년 8월～현재 동국대학교 영상미디어대학 학장 겸 영상대학원 원장
1995년 3월～1999년 2월 동국대학교 정보관리처장역임
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