Journal of Korea Game Society 2016 Dec; 16(6): 79-90
http://dx.doi.org/10.7583/JKGS.2016.16.6.79

게임 프로그래밍

MCTS 기법을 활용한 불완전 정보 카드 게임에서의 인공지능 에이전트 생성 : 하스스톤을 중심으로

오평*, 김지민*, 김선정*, 홍석민**
한림대학교 인터랙션디자인*, 한림대학교 광고홍보학과**
{vudhrh, rudengkim}@gmail.com, {sunkim, seokminhong}@hallym.ac.kr

Generation of AI Agent in Imperfect Information Card Games Using MCTS Algorithm: Focused on Hearthstone

Pyeong Oh*, Ji-Min Kim*, Sun-Jeong Kim*, Seokmin Hong**
Graduate School, Dept. of Interaction Design, Hallym University*, Dept. of Advertising and Public Relations, Hallym University**

요 약

최근 게임분야에서 수준 높은 인공지능 에이전트의 구현은 많은 주목을 받고 있다. 그 중 Monte-Carlo Tree Search(MCTS)는 완전 정보를 가진 게임에서 무작위 탐색을 통해 최적의 해를 구할 수 있는 알고리즘으로, 수식으로 표현되지 않는 경우에 근사치를 계산하는 용도로 적합하다. 하스스톤과 같은 Trading Card Game(TCG) 장르의 게임은 상대방의 카드와 플레이를 예측할 수 없기 때문에 불완전 정보를 가지고 있다. 본 논문에서는 불완전 정보 카드 게임에서 인공지능 에이전트를 생성하기 위해 MCTS 알고리즘을 응용하는 방법을 제안하고, 현재 서비스되는 하스스톤 게임에 적용하여 봄으로써 MCTS 알고리즘의 실용성을 검증한다.

ABSTRACT

Recently, many researchers have paid attention to the improved generation of AI agent in the area of game industry. Monte-Carlo Tree Search(MCTS) is one of the algorithms to search an optimal solution through random search with perfect information, and it is suitable for the purpose of calculating an approximate value to the solution of an equation which cannot be expressed explicitly. Games in Trading Card Game(TCG) genre such as the heartstone has imperfect information because the cards and play of an opponent are not predictable. In this study, MCTS is suggested in imperfect information card games so as to generate AI agents. In addition, the practicality of MCTS algorithm is verified by applying to heartstone game which is currently used.

Keywords : Monte-Carlo Tree Search (몬테-카를로 탐색 트리), Artificial Intelligence (인공지능), Finite State Machine (유한 상태 기계), Behavior Tree (행동 트리), Hearthstone (하스스톤)

Received: Sep. 2, 2016 Revised: Oct, 14, 2016
Accepted: Dec. 1. 2016
Corresponding Author: Seokmin Hong(Hallym University)
E-mail: seokminhong@hallym.ac.kr

ISSN: 1598-4540 / eISSN: 2287-8211

1. 서 론

게임 업계는 지난 십 수 년 간 게임의 흥미유발과 사용자 경험의 증대를 위해 그래픽스, 사운드, 인터페이스 등 많은 요소를 높은 수준까지 발전시켜왔다. 그럼에도 게임에서 유저의 플레이에 직접적으로 관여하며 상호작용하는 인공지능에 대한 연구는 사실상 최근까지 미비했으며, 구현 방식 역시 과거의 알고리즘과 비교해도 크게 발전되었다고 보기 어렵다.

기존의 게임에서 인공지능을 구현하기 위해 사용된 알고리즘으로는 FSM(Finite State Machine), HFSM(Hierarchical Finite State Machine), Behavior Tree 등으로 과거와 비교하여 상당부분 유사하고 단지 게임에 맞게 어느 정도 수정된 수준일 뿐이다. 높은 수준의 지능을 가지는 에이전트를 구현하기 위해 기존의 방법들 외에도 유전 알고리즘이나, 진화 알고리즘, 강화 학습 알고리즘 등 많은 기법들이 있지만, 실시간으로 계산되고 처리되어야 하는 게임에서는 그 적용이 현실적으로 불가능하다. 그러나 실시간으로 처리하기에 적합한 계산량과 연산 시간에 비해 뛰어난 성능을 보여주는 알고리즘이 대두되고 있는데, Monte-Carlo Tree Search(MCTS) 가 바로 그것이다. MCTS는 무작위 탐색을 이용해 목표한 값에 가장 근사한 값을 계산하기 위한 알고리즘으로 인공지능 에이전트를 구현하기 위한 새로운 방법으로써 많은 연구가 진행되고 있으며, 좋은 결과를 산출하고 있다.

게임의 인공지능을 구현할 때 가장 중요한 요소 중 하나는 에이전트의 현재 상태를 검사하는 품질의 추정치를 계산하는 평가 함수이다. 전통적인 게임에서의 인공지능에 대한 개발은 높은 수준의 관련 지식을 보편적으로 필요로 한다[1]. 하지만 MCTS 기법은 위와 같은 요구사항인 높은 수준의 관련 지식을 필요로 하지 않고, 평가 함수에 대한 요구 사항 또한 존재하지 않기 때문에 바둑과 같은 평가 함수를 공식화하기 어려운 게임의 인공지능으로써 활용되기에 적합하다[2]. 실제로 최근 몇 년 동안 인공지능 에이전트를 생성하는 새로운 기법으로써 MCTS에 대한 연구가 활발히 진행되어 왔으며[3], 이러한 연구의 결과로 컴퓨터 바둑, 포커, 헥스 등 많은 장르의 게임에서 그 활용 가능성을 입증하였다[4,5,6]. MCTS는 2006년 처음 바둑 게임에서 사용되고 개발되었지만 바둑이나 전통적인 보드게임에 국한되지 않고 다른 장르의 게임에서 쉽게 적용할 수 있으며 구현이 상대적으로 간단하다는 장점이 있다[1].

MCTS는 입력 변수 혹은 시스템의 확률적 특성으로 인한 결과를 정확하게 예측할 수 없는 확률 모델에서 수치적으로 일련의 난수를 반복적으로 발생시켜 시뮬레이션을 수행해 기대한 해에 근사치를 구하는 알고리즘이다[7,8]. MCTS는 최우선 깊이 탐색 기법을 사용하며 유한한 길이의 어떤 게임에서도 적용될 수 있다[1]. MCTS에 대한 관련 연구는 완전 정보 게임의 영역에서 이미 훌륭한 수준의 인공지능을 생성하는데 많은 발전과 결과를 보였지만, 비완전 정보 게임에 대한 MCTS의 연구와 발전 수준은 아직 상대적으로 빈약하다[7]. 이와 관련된 대표적인 연구로는 포커와 스캣 게임에 MCTS를 적용한 연구가 있으며[8], 본 논문에서 적용하여 인공지능 에이전트를 구현할 블리자드사의 하스스톤과 같은 Trading Card Game(TCG) 장르에 적용된 사례로는, 전통적인 TCG 장르의 게임으로 세계적으로 많은 인기를 끌고 있는 매직 더 게더링(Magic :The Gathering)에 MCTS를 적용한 연구 사례가 있다[9].

본 논문에서 MCTS 기법을 적용한 하스스톤과 같은 TCG 장르의 게임 모델은 사용자가 선택하는 각각의 카드에 따라서 게임의 구도와 양상이 달라져 그로 인한 결과를 예측할 수 없는 확률 모델에 가깝기 때문에 MCTS를 적용하기 적합한 모델이라고 할 수 있다. 또한 기존의 TCG 에서 보여주는 인공지능의 수준은 일반 사용자의 숙련도와 비교했을 때 현저히 떨어지는 수준이며, 개발사는 이를 보완하기 위해 밸런스상의 우위에 있는 카드들을 인공지능에게 사용하도록 하는 등의 우회적인

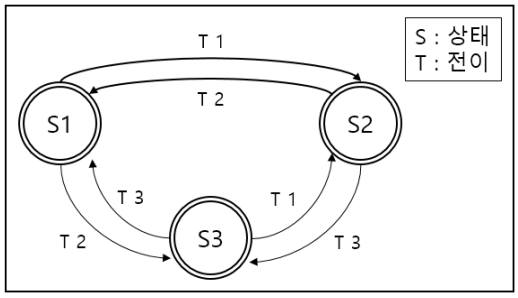
방법으로 인공지능의 난이도를 상승시켜 왔다.

하스스톤과 같은 TCG 게임의 플레이 양상을 2가지로 간략화 하면 그 중 첫 번째는 사용자가 핸드에 보유한 카드를 제시하는 행동이 있으며, 두 번째로는 필드에 제시된 카드를 제어하는 행동이 있다. 본 논문에서는 위와 같은 2가지 행동 양상 중 전자인 카드를 제시함에 있어 좀 더 효율적이고 효과적인 카드를 선택하여 동일한 밸런스 상의 카드로도 사용자와 대전 수준이 맞춰질 수 있도록 MCTS 기법을 사용해 해당 인공지능 에이전트를 구현하였다.

2. 관련 연구

2.1 Finite State Machine (FSM)

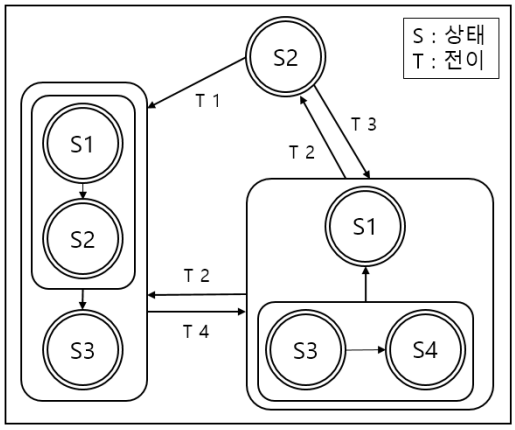
유한 상태 기계라고 불리는 FSM은 간단한 형태의 상태 기계이며, 컴퓨터 프로그램과 순서적 로직을 위해 디자인된 수학적 모델이다. 게임에서 인공지능을 구현하기 위해 가장 보편적으로 사용되어 온 알고리즘이다. 한 번에 하나의 상태만을 보유하고 있을 수 있으며, 환경으로부터 어떠한 조건이 충족이 되면 다른 상태로 전이되는 간단한 구조를 가지고 있다[10]. [Fig. 1]은 FSM의 흐름도를 나타낸다.



[Fig. 1] FSM flowchart

2.2 Hierarchical Finite State Machine (HFSM)

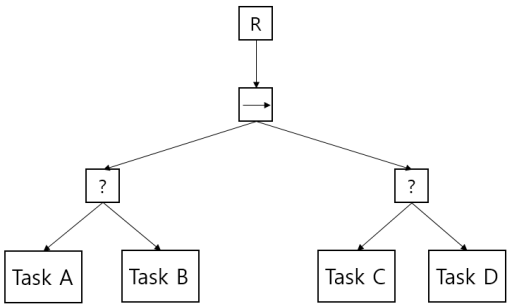
FSM은 설계와 구현이 간단하다는 장점에도 불구하고 상태들이 많아지고 다양해질수록 그에 대한 설계와 구현이 기하급수적으로 어려워지고 코드가 꼬이거나, 유지 보수가 어려워진다는 단점이 있다. 앞에 언급된 FSM의 단점을 보안하기 위해 고안된 HFSM은 FSM과의 기본적인 로직 구조는 상당히 유사하지만, 상태와 상태들 간의 관계를 하나의 계층으로 묶어 재사용성을 높인 구조이다. 상태들을 그룹으로 묶어 전이 상태를 공유한다[11]. [Fig. 2]는 HFSM의 흐름도를 나타낸다.



[Fig. 2] HFSM flowchart

2.3 Behavior Tree

Behavior Tree는 컴퓨터 과학, 로보틱스, 시스템 컨트롤, 비디오 게임 등에서 사용되는 수학적 모델이다. Behavior Tree는 모듈 방식의 유한한 태스크들 사이에서의 전환 방식을 의미하며, 간단한 태스크의 조합으로 인한 복잡한 태스크를 생성하는데 매우 큰 강점을 가지고 있다. Behavior Tree는 HFSM과 유사한 면을 가지고 있지만 가장 큰 차이점으로 HFSM은 상태를 위주로 구조화 되어 있고 Behavior Tree에서는 태스크를 중심으로 구조화 되어 있다는 점이다. [Fig. 3]은 Behavior Tree를 도식화한 그림이다.



[Fig. 3] Behavior Tree Model

2.4 Multi-Armed Bandit Problem

실제 상황 속에서 선택이라는 것은 최대화된 수치적 보상의 기대에 따라 결정된다. 그러나 그러한 선택이나 행동들은 많은 보상을 이끌어 내지 못하는 경우가 대부분이고 선택이나 행동에 대한 보상을 예측하는 것 역시 어렵다. 1952년 Robbins에 의해 제안된 Multi-armed bandit problem은 예를 들어 도박꾼이 카지노에 존재하는 수많은 슬롯머신들 중에서 어떤 슬롯머신에서 게임을 할 것인지, 각 머신에서 얼마나 많은 게임을 할 것인지 결정하는 확률 문제이며, 이로 인해 얼마나 많은 보상을 얻을 수 있는지, 그리고 그 보상치를 최대화 할 수 있는지에 관한 문제이다. 도박꾼이 플레이 할 각 슬롯머신들은 각자의 무작위 보상 값을 가지고 있으며, 보상을 얻을 수 있는지 없는지에 관해서는 알 수 없다[12]. Multi-armed bandit problem은 이와 같은 문제의 모델을 통칭 한다.

2.5 Upper Confidence Bound Tree (UCT)

Upper Confidence Bound (UCB) Method는 1985년 Lai와 Robbins에 의해 제안되고 1995년 Agrawal에 의해 소개되고 분석된 알고리즘이다. 이는 n번의 횟수 동안에 m개의 arm들 중에서 기대되는 보상 값을 위해 신뢰구간의 상한 값을 최대화하는 것을 말한다. 이 방법들 중에서 가장 많이 활용되는 방법으로는 UCB-1 알고리즘이 있다. UCB-1 알고리즘은 “follow the perturbed leader” 알고리즘의 한 범주에 속하며, 최적의 대수 비율을 유지할 수 있음이 증명되었다[13]. 그리고 이를 트리 구조의 탐색공간에 맞게 확장시킨 것이 UCT 알고리즘이다.

2.6 Hearthstone: Heroes of Warcraft

본 논문에서 MCTS를 적용한 실험 대상 게임인 하스스톤(Hearthstone)은 블리자드사에서 개발하고 2014년에 출시한 블리자드 최초의 부분 유료화 전략 시뮬레이션 카드 게임이다. 하스스톤은 기존 정통 TCG 게임이 가지고 있던 어려운 게임 방식이 아닌 게임 플레이를 최대한 간략화 시켜 유저들의 접근성을 높였다. 그 결과 주로 매니아 층의 장르로만 여겨지던 여타의 TCG 게임과는 다르게 다양한 계층의 유저들에게 많은 인기를 얻었다[Fig. 4,5].

[Fig. 4] Screen of Hearthstone play

하스스톤은 두 플레이어가 서로를 상대하게 되며, 양측 플레이어는 턴(turn) 제를 기반으로 번갈아 가며 플레이하게 된다. 한쪽 플레이어가 행동을 취할 차례에는 다른 쪽 플레이어는 상대방의 플레이를 수동적으로 관람할 수만 있으며, 상대방 플레이어가 행동을 마친 후 그때서야 행동할 자격을 얻게 된다.

각 플레이어는 게임 내에서 영웅으로 표현되며, 영웅의 종류로는 9종류 (전사, 도적, 흑마법사, 사제, 드루이드, 사냥꾼, 성기사, 마법사, 주술사) 가 있다. 사용자는 이 9가지 영웅들 중에서 미리 해당 영웅에 대한 카드뭉치(덱)을 구성해 놓고 게임 시작 전에 덱을 선택하여 플레이한다.

양측 플레이어에게는 각 턴마다 1분 15초의 행동 가능 시간이 주어지게 된다. 게임을 함에 있어서 최우선의 목표는 상대방 영웅의 체력을 0이하로 만드는 것이며, 각 턴 1분 15초의 시간동안 플레이어는 보유하고 있는 카드와 필드에 제시한 카드들을 가지고 상대방 영웅의 체력을 0이하로 만들기 위해 전략적으로 플레이하게 된다.

[Fig. 5] Screen of Hearthstone play

3. Monte-Carlo Tree Search (MCTS)

입력 변수 혹은 시스템의 확률적 특성으로 인하여 결과를 정확하게 예측할 수 없는 확률 모델의 경우 분석적인 해를 찾는 것이 불가능 한 경우가 많다. 이러한 경우 수치적으로 일련의 난수를 반복적으로 발생시켜 시뮬레이션을 수행함으로써 원하는 해와 근사한 수치를 얻을 수 있는데, 이러한 기법을 MCTS라고 한다[14].

MCTS를 적용하기 위해서는 충족되어야 할 세 가지 조건이 있으며, 그 조건은 다음과 같다[10,15].

[조건 1] 시뮬레이션을 통한 득실 값의 한계가 있어야 한다. (최대/최소값이 있어야 한다.)

[조건 2] 게임의 규칙이 정해져 있어야 하며, 완전 정보 게임이어야 한다.

[조건 3] 게임의 길이가 제한되어 비교적 시뮬레이션이 빨리 수행되어야 한다.

그러나 하스스톤은 타 TCG와 마찬가지로 상대방의 손에 가지고 있는 카드의 정보나, 상대방이 매 턴 시작 시에 무작위로 섞어져 있는 덱에서 추가되는 새로운 카드의 정보를 알 수 없기 때문에, 완전 정보 게임이 아니다. 플레이어가 상대방의 정보를 알 수 있는 것은 이미 상대방이 필드에 제시한 카드뿐이기 때문이다. 따라서 본 논문에서는 매 턴 시작 시에 추가되는 상대방의 카드 정보와 손에 가지고 있는 카드의 정보를 배제함으로써 MCTS의 2번째 조건을 충족시킨다.

MCTS 알고리즘은 정해진 기준에 따라 4단계의 절차를 반복적으로 수행하게 되며, 이를 수행하기 전에, 탐색 트리를 초기화 하는 과정을 거친 뒤에 수행하게 되며, 각 4단계 과정은 다음과 같다[16][Fig. 6].

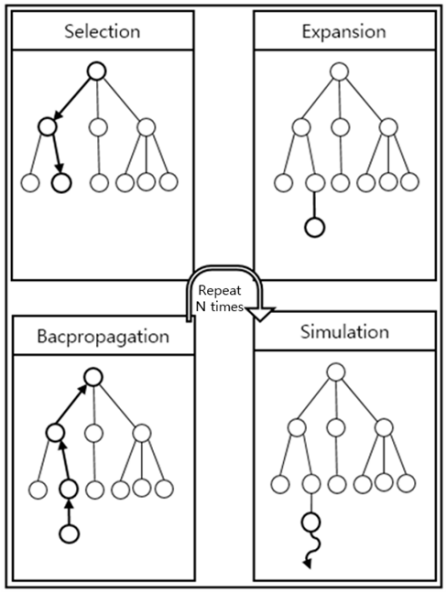
[단계 1] Selection(선택) : 현재 노드에서 무작위 난수를 통해 자식 노드를 선택한다. 이 과정은 단말 노드를 만날 때 까지 반복된다.

[단계 2] Expansion(확장) : 선택 단계에서 선택된 노드에 하나 이상의 자식 노드를 추가하며, 일반적으로는 한 개의 노드가 추가되는 것이 보편적이다.

[단계 3] Simulation(시뮬레이션) : 확장 단계에서 추가된 단말 노드로부터 시뮬레이션을 수행한다.

[단계 4] Backpropagation(역전파) : 시뮬레이션 단계에서 얻어진 보상값을 탐색 트리의 루트 노드로 되돌아가며 노드 정보를 갱

신한다.



[Fig. 6] Algorithm of MCTS [17]

MCTS에서 활용되는 탐색 트리의 각 노드들은 각자 방문 가능한 횟수와 무작위 탐색을 통해 방문된 횟수를 속성으로 가지고 있는데, 이는 [Fig. 7]과 같다.

[Fig. 7] Number of possible visit and number of already visited [17]

4. 구 현

4.1 카드 가치 평가

MCTS 알고리즘을 수행하는 과정에서 시뮬레이션의 평가 지표가 되는 것은 카드의 가치가 된다. 그러나 이런 카드의 가치를 정형화하는 것은 상황에 따라 혹은 영웅들의 특성을 반영한 직업 카드들간의 상성에 따라 카드의 유용성이 매 번 달라지기 때문에 불가능하다. 따라서 상황 요소에 의한 변수는 고려하지 않고 가장 큰 규모의 국내 하스스톤 커뮤니티에서 사용자들이 흔히 사용하는 카드 가치 계산표를 사용하였다[Table 1]

[Table 1] Table book of Card Value [18]

Cost of Cards	Worth of Cards
0 ~ 3 cost	cost x 2 + 2
4 ~ 6 cost	cost x 2 + 1
7 ~ cost	cost x 2

예를 들어 사용할 카드의 사용 비용(Cost)가 0에서 3이하인 3일 경우 해당 카드의 가치는 3 x 2 + 2 로 계산되어 8의 가치를 지닌 것으로 간주한다.

4.2 카드 정보 세팅

트리의 구축에 사용되는 카드 정보는 프로그램 시작 시 초기화 작업에서 반복적으로 호출되고 변하지 않는 정적인 성격의 정보이기 때문에 XML 파일에 작성해 두고 필요시에 정보를 읽어와 동적으로 객체를 생성하는 방식으로 구현하였다[Fig 8].

```
<card>
  <type>minion</type>
  <name>Elven_Archer</name>
  <cost>1</cost>
  <ad>1</ad>
  <hp>1</hp>
  <worth>2</worth>
</card>
```

[Fig. 8] The example of card information in XML

구현과 테스트에 사용되는 카드는 하스스톤에서 기본으로 제공하는 초심자용 기본 덱을 사용 했으며, 이 카드들의 정보와 카드들에 대한 가치 정보는 다음과 같다[Table 2].

[Table 2] List of card information

Name	Classify	Cost	Worth
Elven_Archer	Minion	1	4
Goldshire_Footman	Minion	1	4
Bloodfen_Raptor	Minion	2	6
Novice_Engineer	Minion	2	6
Ironforge_Rifleman	Minion	3	8
Dragonling_Mechanic	Minion	4	9
Gnomish_Inventor	Minion	4	9
Stormwind_knight	Minion	4	9
Stompike_Commando	Minion	5	11
Nightblade	Minion	5	11
Backstab	Spell	0	2
Deadly_Poison	Spell	1	4
Sinister_Strike	Spell	1	4
Sap	Spell	2	6
Assassinate	Spell	5	11

4.3 선택

선택 단계는 이전 탐색트리에서 확장되어 구축된 현재 상태의 탐색 트리에서 무작위 탐색을 통해 단말 노드를 찾아가는 과정이다. 무작위 탐색을 위해 UCT 알고리즘을 사용하여 탐색하도록 하였다[Fig. 9].

선택 단계에서 사용하는 UCT 알고리즘은 다음과 같다. (eq. 1).

$$v_i/(n+e)+\sqrt{\log(n+1)/(n+e)}+rxe$$

(eq. 1)

UCT 알고리즘은 UCB 공식을 트리에 적용가능하게 수정한 알고리즘을 말하며, UCB 공식은 각 레벨에서 가장 최선의 자식노드를 선택하는 역할을 한다.

v는 기대 보상값을 의미하며, n은 방문 횟수, r은 무작위 더블형 실수값이고 e는 엡실론 값으로 사용자가 지정하는 작은 상수 값을 의미한다.

```
TreeNode selected = null;
double bestValue = Double.MIN_VALUE;
for(int i = 0; i < children.size(); i++){
    TreeNode c = children.get(i);
    double uctValue =
            c.totValue / (c.nVisits + epsilon)
            + Math.sqrt(Math.log(nVisits+1)
            / (c.nVisits + epsilon))
            + r.nextDouble() * epsilon;
    if (uctValue > bestValue) {
        selected = c;
        bestValue = uctValue;
    }
}
return selected;
```

[Fig. 9] Selection of card node

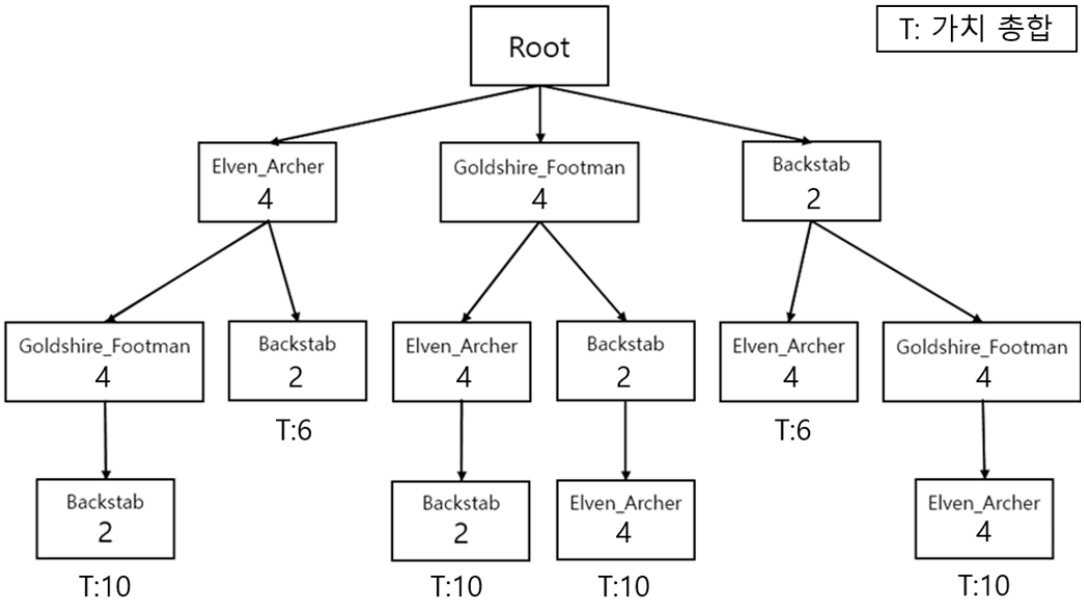
4.4 확장

확장 단계는 선택 단계에서 무작위 탐색을 통해 도착한 단말 노드에서 확장 가능한 노드가 있을 경우 노드를 추가하는 과정이다. 본 구현에서는 UCT 알고리즘을 이용한 무작위 탐색을 통해 단말 노드를 만나도록 한 후, 현재 사용 가능한 카드를 노드에 저장시켜 자식 노드로 확장 한다[Fig. 10].

```
children = new TreeNode[nActions];
for (int i=0; i<nActions; i++) {
    children[i] = new TreeNode();
}
```

[Fig. 10] Expansion of card node

코드 첫 째 줄에서처럼 현재 노드에서 자식노드가 필요한 만큼 배열을 선언하고, 그 자식노드의 배열 수만큼 새로운 노드를 생성함으로써 트리를 확장한다.



[Fig. 11] Result of Simulation

4.5 시뮬레이션 역전파

선택과 확장 단계 이후인 시뮬레이션 단계에서는 선택된 노드에 대한 시뮬레이션을 수행하고 역전파 단계에서 수행된 결과값을 저장하며 방문 했던 노드를 타고 올라가면서 노드에 대한 상태 값을 갱신한다. 알고리즘 수행 이후에는 트리 형식으로 카드들에 대한 알고리즘 수행 결과가 표시되도록 구현하였다[Fig. 11].그림에서 보이는 트리 구조에서의 각각의 네모 칸들은 카드 노드를 의미하며 각 노드안의 첫 번째 줄은 카드의 이름을 의미하고, 두 번째 줄의 숫자는 해당 카드의 가치를 의미한다. 출력된 결과에서 각 각의 단말 노드까지로 이동하면서 만나게 되는 모든 가치의 합이 높은 경로가 최선의 수라고 판단한다. 중복 값이 있을 경우, 트리의 가장 왼쪽의 있는 경로를 최적의 수로 사용하였다.

5. 결 과

하스스톤의 규칙은 크게 분류했을 때, 보유한 자원(마나 크리스탈)을 사용해 손에 보유한 카드(핸드)를 전장(필드)에 사용하고, 그 카드로 상대방을 공격해 상대방의 영웅의 체력을 0 이하로 먼저 떨어트리는 것이다. 여기서 플레이어는 크게 2가지 분류의 행동을 할 수 있는데, 첫 번째는 핸드의 카드를 전장에 사용하는 것이며, 두 번째로는 전장에 나와 있는 카드를 제어 하는 것이다. 본 논문에서는 2가지 행동 중에 전자만을 우선 고려하여 구현하였으며, 후자의 행동은 사용자의 경험적 판단에 따라 플레이 하였다. 하스스톤에는 기본적으로 제공하는 초심자용 기본 덱과 전문가용 기본 덱이 존재하는데, MCTS를 적용한 AI는 초심자용 기본 덱만을 사용하여 총 20게임의 실험을 진행하였다. 10게임은 초심자용 기본 덱을 사용한 하스스톤 내의 AI와 대전을 진행 하였고, 나머지 10게임은 상대적으로 오버 밸런스의 카드들을 사용한 전문가용 기본 덱을 사용하여 진행 하였다.

MCTS를 적용한 AI와 하스스톤의 기본 AI와의 대전에서 양쪽 모두 초심자용 덱을 사용한 10번의 실험에서는 모두 MCTS를 적용한 AI가 승리하였다[Table 3]. 같은 조건에서는 하스스톤의 기본 AI 보다 본 논문이 제안하는 MCTS를 적용한 AI의 성능이 뛰어나다고 볼 수 있다.

하스스톤의 기본 AI는 전문가 덱을 사용하고 MCTS를 적용한 AI는 기본 덱을 사용한 10번의 실험에서는 80% 승률을 보였다[Table 4]. 카드 가치상에서 불리한 환경에서도 50% 이상의 승률을 보인다는 것은 본 논문이 제안하는 MCTS를 적용한 AI가 기존 AI보다 비교적 우수하다고 볼 수 있다.

[Table 3] Simulation Result of MCTS Algorithm vs Heathstone AI (both used basic deck)

MCTS Algorithm (Basic Deck) vs Hearthstone AI (Basic Deck)				
Index	Result	HP Gap	Game Turn	Who first Started
1st game	Win	15 - 0	8	MCTS
2nd game	Win	28 - 0	9	MCTS
3rd game	Win	19 - 0	7	Hearthstone AI
4th game	Win	8 - 0	10	Hearthstone AI
5th game	Win	12 - 0	11	MCTS
6th game	Win	27 - 0	8	MCTS
7th game	Win	16 - 0	8	Hearthstone AI
8th game	Win	30 - 0	8	MCTS
9th game	Win	12 - 0	9	Hearthstone AI
10th game	Win	12 - 0	9	Hearthstone AI

[Table 4] Simulation Result of MCTS Algorithm vs Heathstone AI (used basic and expert deck, respectively)

MCTS Algorithm (Basic Deck) vs Hearthstone AI (Expert Deck)				
Index	Result	HP Gap	Game Turn	Who first Started
1st game	Lose	0 - 9	9	Hearthstone AI
2nd game	Win	29 - 0	8	Hearthstone AI
3rd game	Lose	0 - 6	9	MCTS
4th game	Win	9 - 0	9	MCTS
5th game	Win	12 - 0	9	Hearthstone AI
6th game	Win	19 - 0	9	Hearthstone AI
7th game	Win	10 - 0	8	MCTS
8th game	Win	9 - 0	12	Hearthstone AI
9th game	Win	8 - 0	9	Hearthstone AI
10th game	Win	13 - 0	9	MCTS

표에서 첫 번째 열은 사용한 덱으로의 경기 순번을 의미하고, 두 번째는 시뮬레이션 결과, 세 번째는 게임이 종료된 시점에서 양 플레이어의 HP 수치, 네 번째는 게임이 끝나기 까지 걸린 턴의 수를 의미하며, 마지막 열은 MCTS를 사용한 AI의 선공 여부를 의미한다.

6. 결 론

MCTS는 무작위 탐색을 전제로 하는 정확한 값을 얻기 어렵거나 불가능한 경우에 근사값을 구하는 목적으로 사용되는 알고리즘으로, 넓은 범위의 영역에서의 응용이 입증되어 있는 기법이다. 본 논문에서는 MCTS의 무작위적 탐색으로 인해 선택된 값들을 평가 함수의 값으로 비교하여 최적의 수로 예측되는 수를 계산하였다.

본 논문에서는 기존 TCG 게임의 인공지능의 구현에 사용되던 휴리스틱 기법만으로 구현되던 방식이 아닌 MCTS 알고리즘을 필요에 맞춰 수정하여 구현하였다. 하지만 현재 서비스 되고 있는 게임에 대한 실제 코드를 구할 수 없었기 때문에, 구현 시에는 게임 내에서 제공하는 일부 기본 덱의 카드 정보만을 사용하여 카드 데이터를 구성하였으며, 이 데이터를 XML 파일에 정형화 시키는 작업을 임의로 구성하여 코드를 작성하였다. 실제 코드를 얻어 올 수 없는 문제점 때문에 게임 내에서 실제로 적용해 사용할 수 없다는 한계점이 있으며, 또 모든 플레이에 대한 MCTS의 적용이 아닌 매 턴 카드를 선택하는 과정만을 고려했기 때문에, 필드에서 카드를 제어하는 부분에 있어서 실험결과에 영향을 주었을 가능성이 있다. 따라서 아직까지는 완전히 신뢰할만한 결과라고 보기 어렵다. 하지만 카드를 선택하는 과정 역시 게임의 승패에 가장 큰 역할을 하는 부분임에는 틀림없으며, 오버 밸런스의 전문가 덱을 사용하는 AI를 상대함에 있어서도 더 우세한 승률을 보였기 때문에 비완전 정보 게임인 하스스톤에서의 MCTS의 적용이 좋은 결과를 보임을 증명하였다.

향후 연구에서는 필드 제어에 관련된 MCTS의 적용과 함께 게임 플레이에 영향을 미치는 카드 간의 상성을 고려해 좀 더 실용성을 갖추고 신뢰

성을 갖춘 알고리즘을 보완할 필요성이 있을 것이다.

REFERENCES

[1] G. Chaslot, S. Bakkes, I. Szita, & P. Spronck, “Monte-Carlo Tree Search: A New Framework for Game AI”, In AIIDE, 2008.

[2] M. Enzenberger, M. M¨uller, B. Arneson, and R. B. Segal, “Fuego – An Open-Source Framework for Board Games and Go Engine Based on Monte Carlo Tree Search”, IEEE Trans. Comp. Intell. AI Games, Vol. 2, No. 4, pp. 259 - 270, 2010.

[3] H. J. van den Herik, “The Drosophila Revisited”, Int. Comp. Games Assoc. J, Vol. 33, No. 2, pp. 65 - 66., 2010.

[4] C. S. Lee, M. H. Wang, G. M. J.-B. Chaslot, J. B. Hoock, A. Rimmel, O. Teytaud, S.-R. Tsai, S.-C. Hsu, and T.-P. Hong, “The Computational Intelligence of MoGo Revealed in Taiwan’s Computer Go Tournaments”, IEEE Trans. Comp. Intell. AI Games, vol. 1, no. 1, pp. 73 - 89, 2009.

[5] A. Rimmel, O. Teytaud, C. S. Lee, S. J. Yen, M. H. Wang, and S.-R. Tsai, “Current Frontiers in Computer Go”, IEEE Trans. Comp. Intell. AI Games, Vol. 2, No. 4, pp. 229 - 238, 2010.

[6] C. S. Lee, M. M¨uller, and O. Teytaud, “Guest Editorial: Special Issue on Monte Carlo Techniques and Computer Go”, IEEE Trans. Comp. Intell. AI Games, Vol. 2, No. 4, pp. 225 - 228, 2010.

[7] G. Chaslot, “Monte-carlo tree search.”, Maastricht: Universiteit Maastricht, 2010.

[8] M. Buro, J. R. Long, T. Furtak, and N. R. Sturtevant, “Improving State Evaluation, Inference, and Search in Trick-Based Card Games”, in Proc. 21st Int. Joint Conf. Artif. Intell., Pasadena, California, pp. 1407 - 1413, 2009.

[9] P. I. Cowling, C. D. Ward, E. J. Powley, “Ensemble Determinization in Monte Carlo Tree Search for the Imperfect Information Card Game Magic: The Gathering”, IEEE Trans. Comp. Intell. AI Games, Vol. 4, No. 4, pp. 241-257, 2012.

[10] https://en.wikibooks.org/wiki/A-level_Computing/AQA/Paper_1/Theory_of_computation/Finite_state_machines

[11] http://aigamedev.com/open/article/hfsm-gist/

[12] J. Vermorel, & M. Mohri, “Multi-armed bandit algorithms and empirical evaluation”, In Machine learning: ECML 2005, Springer Berlin Heidelberg, pp. 437-448. 2005.

[13] A. Garivier, & E. Moulines, “ ”, arXiv preprint arXiv:0805.3415. 2008.

[14] Kyung-Min. Seo, and Hae-Sang. Song, “Importance Sampling Embedded Experimental Frame Design for Efficient Monte Carlo Simulation”, The Journal of the Korea Contents Association, Vol. 13, No. 4, pp.53-63, 2013.

[15] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P.Rohlfshagen, ... & S. Colton, “A survey of monte carlo tree search methods”, Computational Intelligence and AI in Games, IEEE Transactions on, Vol. 4, No. 1, pp. 1-43, 2012.

[16] Young-Wook. Choi, Byung-Doo. Lee, “The best move sequence in playing Tic-Tac-Toe game”, Korean Society For Computer Game, Vol. 27, No. 3, pp. 11-16. 2014.

[17] https://en.wikipedia.org/wiki/Monte_Carlo_tree_search

[18] http://www.inven.co.kr/board/powerbbs.php?come_idx=3559&name=subject&keyword=%EA%B0%80%EC%B9%98&l=1807

오 평 (Oh, Pyeong)

약 력 : 2014.2 한림대학교 유비쿼터스 게임공학과 졸업
2016.8 한림대학교 인터랙션디자인 석사과정(現)

관심분야 : 게임프로그래밍

김 선 정 (Kim, Sun-Jeong)

약 력 : 2003 고려대학교 컴퓨터학과 이학박사
2005- 한림대학교 융합소프트웨어학과 교수

관심분야 : 컴퓨터 그래픽스, 게임 프로그래밍

김 지 민 (Kim, Ji-Min)

약 력 : 2014.2 한림대학교 유비쿼터스 게임공학과 졸업
2016.8 한림대학교 인터랙션디자인 석사과정(現)

관심분야 : 게임프로그래밍

홍 석 민 (Hong, Seok min)

약 력 : 2003 The University of Texas at Austin 광고학박사
2003-현재 한림대학교 광고홍보학과 교수

관심분야 : 마케팅, 뉴미디어, 사회연결망분석