Journal of Korea Multimedia Society Vol. 25, No. 2, February 2022(pp. 432-439)
https://doi.org/10.9717/kmms.2022.25.2.432

효용이론 기반 숙고형 행동트리를 이용한 게임 인공지능 에이전트

권민지†, 서진석††

Game AI Agents using Deliberative Behavior Tree based on Utility Theory

Minji Kwon†, Jinseok Seo††

ABSTRACT

This paper introduces deliberative behavior tree using utility theory. The proposed approach combine the strengths of behavior trees and utility theory to implement complex behavior of AI agents in an easier and more concise way. To achieve this goal, we devised and implemented three types of additional behavior tree nodes, which evaluate utility values of its own node or its subtree while traversing and selecting its child nodes based on the evaluated values. In order to validate our approach, we implemented a sample scenario using conventional behavior tree and our proposed deliberative tree respectively. And then we compared and analyzed the simulation results.

Key words: Behavior Tree, Utility System, Artificial Intelligence

1. 서 론

현대의 많은 게임에서 사용되는 인공지능 에이전트는 플레이어와 상호작용을 하거나 플레이어와 상관없이 자율적으로 행동하기도 한다. 어드벤처 게임 등에서 등장하는 적이나 NPC(Non-Player Character)가 대표적인 인공지능 에이전트의 예라고 볼 수 있다. 인공지능 에이전트의 설계 방법에는 반응형(reactive) 시스템인 유한 상태 기계(FSM), 행동트리(BT)와 숙고형(deliberative) 시스템인 계층적 작업 네트워크(HTN), 목표 지향적 행동 계획(GOAP), 효용 기반 시스템(Utility System) 등이 있다[1]. 이 중 대표적인 반응형 시스템인 행동트리는 게임 인공지능에 자주 사용되는 설계 방법으로, 에이전트의 행동 조건과 동작을 트리 형식으로 정의하여 표현한다. 행동트리는 직관적이고 구현하기 쉬워 복잡한 AI를 깔끔하게 구현할 수 있지만, 획일화된 의사 결정을 수행하기 때문에 상황에 따라 다양한 의사 결정을 내리는 것이 어렵다는 것이 단점이다. 상황에 따라 다양한 의사 결정을 내릴 수 있는 설계 방법으로는 효용 기반 시스템이 있다. 효용 기반 시스템은 에이전트가 수행하는 행동에 효용값이라는 점수를 부여하고 각 행동에 대한 가치를 평가하여 최종적으로 실행할 행동을 선택한다. AI가 수행하는 행동의 가치를 수치화함으로써 여러 가지 잠재적 행동에 대한 선택 가능성을 주기 때문에 다양한 의사 결정을 내릴 수 있는 장점이 있다.

본 논문에서는 기존 행동트리의 한계점을 분석하

※ Corresponding Author: Jinseok Seo, Address: (46282) 176 Eomgwangno, Busan Jin-gu, Busan, 47340, Korea, TEL: +82-10-9896-0713, E-mail: jsseo@deu.ac.kr
Receipt date: Feb. 11, 2022, Approval date: Feb. 23, 2022

† Game Animation Engineering Major, Dong-eui University (E-mail: asjj1125@naver.com)
†† Game Engineering Major, Dong-eui University
※ This work was supported by the National Research Foundation of Korea(NRF) grant funded by the Korea government(MSIT). (No. NRF-2019R1F1A1041854)

여 효용이론을 적용한 확장된 행동트리를 제안하고자 한다. 구성은 다음과 같다. 2장에서는 행동트리와 효용이론의 기본적인 이론과 구성 요소를 분석하고 행동트리 확장과 관련한 연구사례를 살펴본다. 3장에서는 2장에서 분석한 행동트리의 한계점을 효용이론을 통해 보완하는 방법을 소개하며 4장에서 기존 행동트리와 제안된 행동트리를 예시를 통해 비교해 본다. 마지막 5장에서는 결론으로 제안된 행동트리의 평가와 함께 연구방향에 대해 서술한다.

2. 배경 이론

2.1 행동트리(Behavior Tree)

인공지능 에이전트를 트리 형태로 정의하는 설계 방법이다[3,4]. 행동트리의 최소 단위인 노드는 행동, 조건, 실행 방법 등의 카테고리를 가지고 있으며 인과관계 혹은 우선순위에 맞게 연결하여 트리를 구성한다. Fig. 1에서는 행동트리로 설계된 간단한 인공지능 에이전트의 논리 구조를 보여준다. 이 에이전트는 전등이 켜져 있는지 확인하고 스위치를 찾아 전등을 끄는 과정을 수행한다. 이 그림을 이해하려면 우선 행동트리의 노드에 대해 알아볼 필요가 있다.

행동트리의 노드들은 실행한 후 반드시 상태(State)를 반환한다. 상태에는 성공(Success), 실행중(Running), 실패(Failure)가 있으며 이 상태를 통해 상위와 하위 노드 간에 의사소통이 이루어진다.

액션(Action) 노드는 행위를 실행하는 노드로 트리의 말단에 위치하며 트리에 따라 인과관계에 맞는 다양한 파생 노드를 가진다. 컨디션(Condition) 노드는 조건을 실행하는 노드로 주로 에이전트의 주변

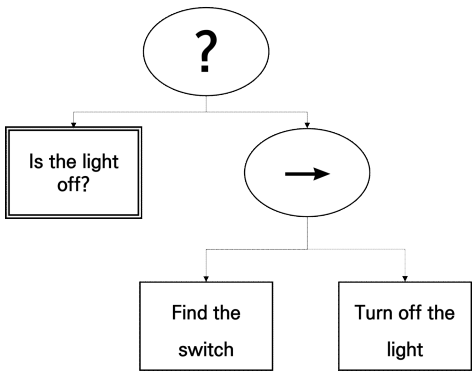

Fig. 1. An example of behavior tree.

환경 혹은 능력치 등에 대해 검사하며 액션 노드처럼 트리에 따라 다양해진다. 컴포지트(Composite,) 노드는 하위 노드의 실행 방법에 관해 결정한다. 하위 노드의 실행 순서를 결정하거나 실행할 하위 노드를 선택할 수 있다. 데코레이터(Decorator) 노드는 컨디션 노드처럼 조건을 실행한다. 다만 컨디션 노드와 달리 하위 노드가 존재하며 반환 상태에 따라 하위 노드의 실행 여부가 달라진다.

셀렉터(Selector) 노드는 컴포지트 노드의 파생 노드이다. 하위 노드 중 가장 왼쪽(첫 번째) 노드부터 실행시키며 하위 노드가 성공을 반환할 경우 다음 하위 노드의 실행을 중단하고 성공을 반환한다. 시퀀스(Sequence) 노드 역시 컴포지트 노드의 파생 노드이며 셀렉터 노드와 달리 모든 하위 노드가 성공을 반환해야 성공을 반환한다.

행동트리는 여기서 설명된 노드 외에도 다양한 노드가 존재하며 개발자에 따라 추가적인 파생 노드들을 만들 수 있어 확장성이 큰 장점이 있다. Fig. 1과 같이 가시적인 트리의 형태로 표현할 수 있어 직관성도 좋으며, 단순하고 규칙적인 노드의 연결로 구현하기도 쉬워 복잡한 AI의 구현에도 적합하다.

2.2 행동트리의 한계

에이전트가 수행해야 할 작업을 결정하기 위해서는 다른 에이전트 또는 외부 환경(세계)에 대한 정보에 의존해야 하는 경우가 많이 발생한다. 단편적인 외부 자극이나 이벤트에만 의존하지 않고 종합적인 정보를 기반으로 한 결정이 더욱 자연스러운 에이전트 행동을 완성할 수 있기 때문이다. 행동트리는 에이전트가 “할 수 있는 작업”을 트리라는 논리적인 구조로 쉽게 모델링할 수 있다는 장점이 있지만, 외부 세계의 종합적인 정보로부터 “해야 할 작업”을 결정하는 데는 뛰어나지 않다[2]. 만약 행동트리에서 외부 세계의 모든 조건을 처리하려고 한다면 매우 복잡한 트리구조를 갖게 될 뿐만 아니라 다른 에이전트나 인공지능 시스템이 변경되면 트리를 다시 모델링해야 할 수도 있다.

위에서 설명한 행동트리 노드 중 셀렉터와 시퀀스는 실행 우선순위를 결정하는 기능을 제공하지 않는다. 하위 노드의 반환 값과는 상관없이 오직 트리의 본 순서대로 순회하므로 하위 노드의 실행 우선순위가 고정적이다. 실행 우선순위가 고정적이면 상황에

상관없이 같은 순서의 행동을 선택하기 때문에 의사 결정의 폭이 좁고 획일적이다.

본 연구에서는 이 한계점을 가치를 수치화한 효용값으로 우선순위를 결정할 수 있는 개념인 효용이론을 통해 행동트리의 한계를 보완하고자 하였다.

2.3 효용이론(Utility Theory)

효용이론[5]이란 사람이 느끼는 효용을 측정하여 수치화시키는 것이다. 인공지능 설계 방법인 효용 기반 시스템에서는 에이전트가 수행하는 행동에 효용값이라는 점수를 부여하고 각 행동에 대한 가치를 평가하여 최종적으로 실행할 행동을 선택하는 것을 의미한다.

예를 들어, 게임에서 아이템에 대한 제작 의뢰를 넣는 방법으로 3가지의 방법이 있다고 하자. A 방법은 제작에 5만 코인이 들며 게임 시간으로 1시간 걸리고 재료비가 필요하지 않다. B 방법은 제작에 4만 코인이 들며 2시간이 걸리고 재료비로 2천 코인을 지불해야 한다. 마지막 C 방법은 제작에 1만 코인이 드는 대신 제작 시간이 21시간에 재료비 1만 코인을 지불해야 한다. 이 경우 단순히 가격을 비교하는 것으로는 해결되지 않는다. 예시를 보면 효용 가치를 가지는 요인은 제작료, 제작에 걸리는 시간, 재료비가 있다. 여기서는 제작료과 재료비는 돈이라는 같은 단위의 가치를 가지므로 함께 계산할 수 있지만, 일반적인 게임에서는 재료를 직접 구해오거나 레벨에 따른 금전 가치 변화 등의 변수가 존재하므로 상황에 맞춰 따로 둘 필요가 있다. Fig. 2는 이 예시를 정리한 것이다.

효용이론에서 효용 점수는 서로 비교하기 위해 같은 척도를 가져야 한다. 이 예시에서 제작비와 재료비는 같은 척도를 가지고 있고, 제작 시간은 그 둘과 다른 척도를 가진다. 서로 다른 척도를 가진 요인들을 비교하려면 먼저 점수들을 정규화시킬 필요가 있다. 정규화는 평균을 통해 0부터 1사이의 값으로 값을 변환시키는 방법이 일반적이다. 이 예시에서는 같은 척도를 가진 제작비와 재료비의 값을 서로 더하여 최종 제작비라 하고, **{1-(각 방법의 최종 제작비/모든 방법의 최종 제작비)}/2**로 효용값을 산출한다. 제작 시간 또한 **{1-(각 방법의 제작 시간/모든 방법의 제작 시간)}/2**로 효용값을 산출한다. Fig. 3은 이 예시를 정리한 것이다.

정규화한 효용값을 통해 최종 제작비의 효용과 제작 시간의 효용을 수치로 계산할 수 있게 되었다. 다음에는 이 두 효용값에 가중치를 적용하여 가장 큰 값을 선택한다. 이 아이템을 제작하려는 플레이어는 코인이 많고 조급한 성격을 가진 a형과, 코인은 아깝지만 시간은 많이 할애할 수 있는 b형이 있다. 두 유형의 제작비와 제작 시간에 대한 가중치를 비율로 나타내면 a형은 2 대 8이며, b형은 8 대 2이다. Fig. 4에서는 이 비율을 바탕으로 가중치를 계산한 것이다.

a형 플레이어의 최고 효용 점수는 A 방법의 0.9372이며, b형 플레이어의 최고 효용 점수는 B 방법의 0.3414이다. a형 플레이어의 경우 B, C 방법과 달리 A 방법이 압도적으로 효용이 높아 다른 선택의 여지가 없지만, b형 플레이어의 경우 B와 C 방법이 0.3414, 0.3412라는 매우 근접한 효용 점수를 가지고 있어 선택의 여지가 갈린다. 이 경우 미리 정해둔 임계치 이하의 차이는 무시하고 랜덤하게 선택함으로써 보다 자연스러운 에이전트의 행동을 기대할 수 있다.

Method A	Method B	Method C
Fee: 50,000c	Fee: 40,000c	Fee: 10,000c
Period: 1 hour	Period: 2 hours	Period: 21 hours
Material: free	Material: 2,000c	Material: 10,000c

Fig. 2. An example of the utility data for item enhancement.

Method A	Method B	Method C
Total Fee: 0.277	Total Fee: 0.312	Total Fee: 0.411
Period: 0.479	Period: 0.459	Period: 0.062

Fig. 3. Normalized utility values from utility data of Fig. 2.

Method A	Method B	Method C
a: 0.9372	a: 0.4296	a: 0.1318
b: 0.3174	b: 0.3414	b: 0.3412

Fig. 4. Weighted utility values from values of Fig. 3.

이 절에서는 게임 아이템 생산 예시를 통해 효용이론을 설명하고 더 나아가 효용값 산출 방법과 효용 선출에 대해 서술하였다. 예시에서 보인 바와 같이, 효용이론을 통해 에이전트가 수행하는 행동의 가치를 수치화함으로써 마치 사람이 직접 생각한 것과 같은 효용적인 선택을 내릴 수 있었다. 더 나아가 실시간으로 효용값에 변화를 줄 경우, 여러 가지 잠재적 행동에 대한 선택 가능성을 줄 수 있어 에이전트에게 다양한 의사 결정을 내리게 할 수 있는 장점도 존재한다.

2.4 관련 연구

행동트리의 효율성은 Halo2[3,4]와 같은 대형 상용화 게임에서 이미 검증되었으며, 이미 Unity[6]와 Unreal[7]과 같은 대중적인 게임엔진에서도 내장되어 사용되고 있다. 게다가 행동트리를 일반화하여 게임뿐만 아니라 로봇을 위한 제어시스템에서도 활용이 되고 있다[8].

행동트리의 장점에도 불구하고 모든 응용에 적합한 에이전트를 행동트리로만 설계하는 것은 쉽지 않다. 이는 행동트리를 포함하는 반응형 인공지능 시스템의 한계라고도 할 수 있다. 반응형 시스템의 단점을 보완하고자 여러 연구자들은 다양한 확장을 행동트리에 추가하는 방식으로 연구를 진행하였다.

Topiary[9]라는 도구는 행동트리의 각 노드에 매개변수화(parametarization)를 적용하였다. 이 연구에서는 런타임에 변화하는 매개변수 값을 통해 에이전트가 보다 다양한 환경변화에 적응할 수 있도록 하였다. [10]에서는 매개변수화를 포함하여 더 다양한 변환 연산을 행동트리 노드에 적용하고자 하였다. 이 연구에서는 행동트리에 매개변수 노드, 확률 노드, 변이 노드, 대체 노드 등 네 가지 확장 노드를 추가하여 동적으로 행동트리의 각 노드가 변화하도록 하였다.

행동트리에 효용이론을 접목하고자 하는 시도도 찾아볼 수 있다[11]. [11]에서는 행동트리의 구성요소인 셀렉터에 효용계산 기능을 추가하여 반응형 시스템과 숙고형 시스템의 장점을 결합하고자 하였다. 본 연구에서는 [11]의 연구를 좀 더 일반화하여 세 가지 유형의 효용기반 노드를 정의하여 복잡한 에이전트 행동을 더 쉽고 간결하게 표현히 가능하도록 하였다.

행동트리 노드에 좀 더 복잡한 연산을 적용한 연구도 찾아볼 수 있다. [12]에서는 유전자 알고리즘을 적용하여 행동트리가 변화하는 상황에 대응하여 진화하도록 하였으며, [13]에서는 강화학습 알고리즘을 이용하여 행동트리가 동적으로 진화하도록 하였다. 행동트리는 아니지만 반응형 인공지능 시스템의 또 다른 예인 상태머신(STM)에 유전자 알고리즘과 인공 신경망 알고리즘을 적용한 연구 사례도[14]도 찾아볼 수 있다.

3. 확장된 행동트리 제안

본 연구에서는 행동트리의 한계점을 보완하기 위하여 기존 행동트리에 3개의 노드를 추가하여 효용이론을 적용한 확장된 행동트리를 제안한다. 추가된 노드는 효용액션, 효용셀렉터, 효용시퀀스이다.

효용액션은 기존 액션 노드를 확장하기 위하여 효용값 계산 기능을 추가한 노드이다. 뒤에서 서술할 효용컴포지트를 사용하기 위한 전용 액션 노드라고도 할 수 있다. 기존의 액션 노드가 실행한 결과, 즉 “성공”이나 “실패” 혹은 “실행중”을 반환한다면 효용액션은 해당 액션에서 계산된 효용값을 함께 반환한다. 효용액션이 반환한 효용값은 상위 노드인 효용컴포지트가 하위 효용 노드들을 선택하기 위해 사용된다. 효용액션 노드는 반드시 뒤에서 설명할 효용컴포지트의 하위 노드로만 작동한다.

효용액션의 사용방법은 Fig. 5와 같다. 효용액션 노드를 작성한 후 효용컴포지트의 하위 노드로 추가하면 된다. 효용액션의 상위 노드인 효용컴포지트의 순회 차례가 되면, 이 효용컴포지트는 우선 자신의 하위 노드들에게 효용값을 요청한다. 이 상위 효용컴포지트 노드는 하위 노드로부터 전달받은 모든 효용

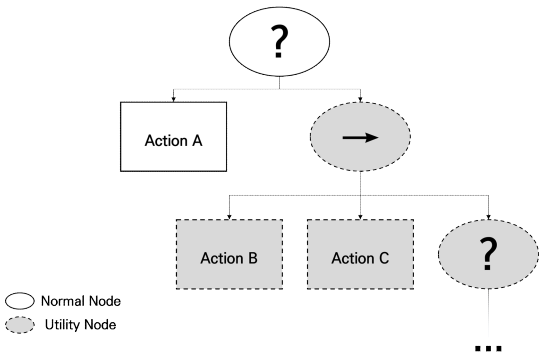

Fig. 5. Utility action nodes under a utility composite node.

값을 비교한 후 하위 노드를 정렬하여 먼저 실행할 하위 노드를 선택한다.

효용셀렉터는 하위 노드인 효용액션 노드로부터 효용값을 전달받아 하위 노드를 재정렬하여 순차적으로 실행하는 셀렉터 노드이다. 효용셀렉터와 효용액션의 조합을 이용하면 매 시뮬레이션 프레임마다 하위 노드들의 순회 순서가 변경되어 가장 큰 효용값을 갖는 하위 노드를 선택하여 실행할 수 있다.

효용시퀀스도 효용셀렉터와 마찬가지로 자신을 순회하기 전에 하위 노드를 재정렬한다. 효용시퀀스 역시 순서가 고정된 기존의 시퀀스와 달리 매 시뮬레이션 프레임마다 하위 노드의 실행 순서 달라진다.

4. 제안된 행동트리의 응용

4.1 시나리오

식당 주방을 배경으로 한 간단한 경영 게임을 제작한다고 가정한다. 이 게임에서는 아르바이트생을 고용할 수 있고, 아르바이트생은 작업하거나 쉬는 행동을 반복한다. 4장에서는 이 아르바이트생을 시뮬레이션하는 에이전트를 기존의 행동트리와 효용이론을 적용한 행동트리로 구현하며 비교해볼 것이다.

아르바이트생은 설거지 행동으로 그릇 수거 및 그릇 씻기 액션을, 휴식 행동으로 눈붙이기 및 화장실 가기 액션을 가지고 있다. 그릇이 싱크대에 얼마나 쌓여있는가에 따라 그릇을 씻을지 수거할지 결정하며 요의에 따라 눈붙이기를 할지 화장실을 갈지 결정한다. 이 액션들에 관여하는 상태변수에는 스태미나, 요의, 그릇량이 있다.

4.2 기존 행동트리를 이용한 예시

4.2.1 기존 행동트리를 이용한 시나리오 구현

기존 행동트리에서도 효용이론의 도입 없이 직접 데코레이터를 이용하여 조건을 지정하면 의도한 행동을 구현할 수는 있다. 다만 미리 정해진 특정 조건을 기준으로만 액션 노드를 선택할 수 있으며 알고리즘이 복잡할수록 트리 내 노드의 수가 많아지고 그에 따라 관리하기 어려운 단점이 있다. 기존 행동트리로 구현한 결과는 Fig. 6과 같다.

아르바이트생이 작업을 하기 위해 작업 시퀀스(루트 노드의 첫 번째 하위 노드)로 진입하면 컨디션에서 세척할 그의 양과 수거 여부 등을 확인하여 싱크

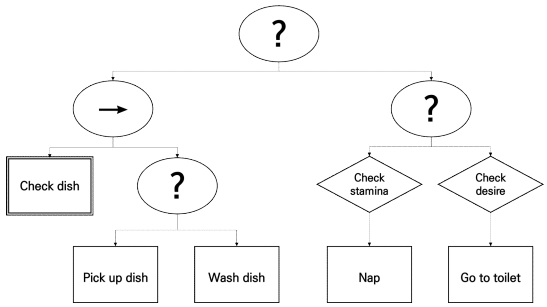

Fig. 6. Scenario using conventional behavior tree.

대 포화 정도를 계산한다. 이 싱크대의 포화 상태에 따라 작업 셀렉터(작업 시퀀스의 두 번째 하위 노드)에서 어떤 행동을 할지 결정한다. 만약 그릇이 5개 미만이면 작업 시퀀스 노드는 실패를 반환하여 휴식 셀렉터(루트 노드의 두 번째 하위 노드)로 넘어간다. 휴식 셀렉터에서는 데코레이터를 통해 상태변수를 조사하여 행동이 적합한지 판단한다. 두 행동 중 하나를 실행하면 휴식 셀렉터에 성공을 반환하고 트리의 순회를 종료한다.

4.2.2 기존 행동트리로 작성된 시나리오의 실행 결과

행동트리의 시뮬레이션은 Unity Engine과 Behavior Tree Visualizer(BTV)[15]를 사용하여 진행하였다. 시뮬레이션 결과는 콘솔 창에 에이전트가 수행하고 있는 액션 노드 이름과 에이전트의 상태를 출력하여 확인하였다. Table 1은 출력 메시지 중 일부를 가져온 것이다.

트리가 한 번 순회를 마칠 때마다 요의가 1씩 증가하고 세척해야 할 그릇량은 0~3씩 증가한다. 순회 과정에서 작업 시퀀가 성공을 반환하면 스태미나가 1씩 감소한다. 한 순회 당 그릇 5개를 세척할 수 있다. 만약 세척해야 할 그릇이 없거나 5개 미만이면 스태미나에 상관없이 휴식을 취하도록 한다. 요의가 최대(10)가 되면 그릇량과 상관없이 화장실을 가게 된다. 눈붙이기 행동은 스태미나를 3씩, 화장실은 1씩 증가시킨다.

이 행동트리에서는 루트 노드의 두 컴포지트 노드의 실행순서는 항상 고정된다. 즉, 항상 첫 번째 컴포지트 노드인 작업 시퀀스부터 순회한다. 만약 두 컴포지트 노드의 실행 순서를 런타임에 결정하고자 한다면 두 배로 복잡한 행동트리를 설계해야 한다. 이러한 문제는 휴식 셀렉터에도 발생한다. 이 행동트리에서

Table 1. Log of the execution of conventional behavior tree.

	Log
1	Agent naps. (stamina 10, desire 1, dish 2, sink 0)
2	Agent naps. (stamina 10, desire 2, dish 4, sink 0)
3	Agent naps. (stamina 10, desire 3, dish 7, sink 0)
4	Agent pickups dish. (stamina 9, desire 4, dish 4, sink 5)
5	Agent washes dish. (stamina 8, desire 5, dish 5, sink 0)
6	Agent pickups dish. (stamina 9, desire 6, dish 3, sink 5)
7	Agent washes dish. (stamina 8, desire 7, dish 4, sink 0)
8	Agent naps. (stamina 10, desire 8, dish 6, sink 0)
9	Agent pickups dish. (stamina 8, desire 9, dish 4, sink 5)
10	Agent washes dish. (stamina 7, desire 10, dish 6, sink 0)
11	Agent goes to toilet. (stamina 8, desire 0, dish 9, sink 0)
12	...

는 항상 요의를 먼저 실행하지만, 요의 값이 아무리 커도 휴식부터 취하게 하려면 마찬가지로 휴식 셀렉터 이하의 노드보다 두 배 큰 서브트리가 필요하다.

4.3 제안된 행동트리를 이용한 예시

4.3.1 제안된 행동트리를 이용한 시나리오 구현

제안된 행동트리에서는 효용이론을 이용하여 기존의 행동트리가 가진 제약된 행동 선택의 폭을 넓혀 상황에 따라 맥락을 가진 행동을 취하는 에이전트를 구현하기 쉬워진다. 기존의 행동트리는 알고리즘이 복잡해질수록 노드 수가 많아지며 많아지는 만큼 조건을 관리하기에도 어려움이 있다. 제안된 행동트리는 대부분의 조건이 효용값의 비교로 결정되므로 기존 행동트리가 섬세하게 구현하기 어려운 복잡한 조건들을 보다 적은 노드를 이용해 구현할 수 있다.

제안된 행동트리로 구현한 시나리오는 다음과 같다. Fig. 7를 보면 기존 행동트리의 시나리오와 달리

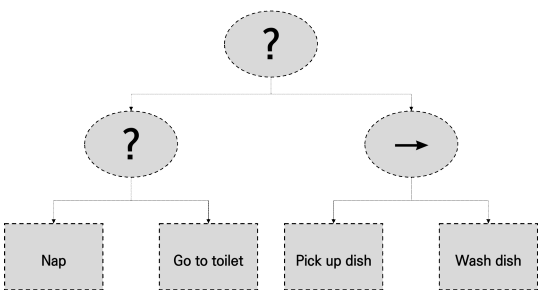

Fig. 7. Scenario using proposed behavior tree.

노드 수가 확연히 줄어든 것을 확인할 수 있다. 컴포지트들은 효용이론을 적용한 효용컴포지트로 교체되었으며 불필요한 하위 노드들을 삭제하였다.

아르바이트생이 처음 출근하여 작업을 시작할 때 설거짓거리가 없다면 작업 효용시퀀스의 효용값은 0이어야 하며, 휴식 효용셀렉터는 0보다 커야 한다. 게임이 시작된 후 여러 상태변수가 변화하면 본격적으로 효용값이 변화하기 시작한다. 휴식 효용액션은 **(1-스태미나) X (요의 혹은 1-요의) X (액션 가중치)**의 계산식을 통해 효용값을 산출하며, 작업 효용액션은 **(스태미나) X (그릇량 혹은 1-그릇량) X (싱크대 포화 상태 혹은 1-싱크대 포화 상태)**로 산출한다. 여기서 상태 변수들은 각 상태의 최댓값으로 나눈 0과 1사이의 수로 정규화한다.

4.3.2 제안된 행동트리의 실행 결과

4.2.2의 실행 결과와 같은 상태변수 변화를 가진 채 실행하였다. 그릇량의 최대 수는 50그릇이며 싱크대의 용량은 20, 액션 가중치는 0.5로 통일하였다. Table 2의 괄호에는 상태변수의 정보와 효용컴포지트에 의해 계산된 효용액션의 효용값이다.

Table 2를 보면 효용값 중 가장 큰 값의 노드를 실행하는 것을 확인할 수 있다. 4.2의 기존 트리와 달리 행동에 관성이 있어 행동 변화가 비교적 늦지만, 이는 오히려 너무 빠른 시간 내에 급변하는 에이전트의 행동을 방지할 수 있으며 자연스러운 행동 변화를 유도할 수 있다. 만약 더 빠른 행동 변화가 필요하다면 효용 계산식에 들어가는 가중치의 크기를 더 크게

Table 2. Log of the execution of proposed behavior tree.

	Log
1	Agent naps. (stamina 10, desire 1, dish 2, sink 0, UV 0.0000)
2	Agent pickups dish. (stamina 9, desire 2, dish 2, sink 2, UV 0.0400)
3	Agent washes dish. (stamina 8, desire 3, dish 5, sink 0, UV 0.0864)
4	Agent pickups dish. (stamina 7, desire 4, dish 0, sink 5, UV 0.0800)
5	Agent washes dish. (stamina 6, desire 5, dish 1, sink 0, UV 0.175)
6	Agent pickups dish. (stamina 5, desire 6, dish 3, sink 1, UV 0.0120)
7	Agent washes dish. (stamina 4, desire 7, dish 6, sink 0, UV 0.2350)
8	Agent goes to toilet. (stamina 5, desire 0, dish 7, sink 0, UV 0.2100)
9	Agent naps. (stamina 8, desire 1, dish 9, sink 0, UV 0.2500)
10	Agent pickups dish. (stamina 7, desire 2, dish 2, sink 9, UV 0.1440)
11	Agent washes dish. (stamina 6, desire 3, dish 3, sink 0, UV 0.3024)
12	...

조절하면 된다.

4.4 실행 결과 비교

기존 행동트리에서 아르바이트생 에이전트는 오직 특정 조건이 충족되는 행동만 취할 수 있었지만, 제안된 행동트리에서는 에이전트 자체의 상태나 그릇량과 같은 외부 환경에 따라 효용값이 변화하여 선택방법에 따라 그릇이 아직 남아있음에도 화장실을 다녀와서 휴식을 취하거나 최대한 스태미나를 회복하고 효율적으로 작업을 시작하는 등 좀 더 자연스러운 행동을 취할 수 있었다. 이렇듯 이전에 처한 상황에 따라 다채로운 대처를 하는 아르바이트생의 예시를 통해 제안된 행동트리는 보다 지능적이면서도 직관적인 에이전트 설계가 가능하게 한다는 것을 확인할 수 있었다.

4.3에서 구현한 시나리오의 경우 4.2와 달리 효용값을 이용해 다른 행동으로 전이할 때 관성이 생기는 특징이 있다. 이 관성으로 인해 오히려 원하지 않는 흐름이 일어날 수 있는 문제점이 있지만, 이 문제는 가중치를 통해 전이 속도를 조절하여 해결할 수 있다. 이렇듯 복잡한 에이전트를 비교적 적은 노드로 간결하게 표현하고, 단편적인 조건들이 아닌 전체 흐름을 제어하여 자연스러운 행동 과정을 유도하는 점에서 제안된 행동트리의 가능성을 확인할 수 있었다.

5. 결 론

본 논문에서는 기존 행동트리의 한계점인 획일화된 의사 결정을 효용이론과 결합하여 확장된 행동트리를 제안하였다. 효용이론과의 결합을 통해 효용값 계산을 수반하는 효용 노드를 바탕으로 효용 액션, 효용셀렉터, 효용시퀀스를 구현하여 행동트리를 보완하였다.

이 연구에서는 기존 행동트리의 단점을 효용이론과 혼합하여 상쇄하고 더 확장성 있는 행동트리를 제안함으로써 행동트리와 다른 게임 에이전트 기법의 혼합이 큰 가치가 있음을 확인하였다. 다만 실제 게임에서는 다양한 에이전트가 존재한다. 각 에이전트들의 전체 효용값을 비교하거나, 에이전트 내부에 행동트리가 복수로 존재하여 행동트리 사이에 조율이 필요할 상황이 발생할 수 있다. 이런 경우에는 에이전트들을 관리하는 관리자나 블랙보드가 필요할 것이다. 더불어 각 행동트리나 노드의 효용값을 조합하여 에이전트뿐만 아닌 게임 전체의 목표를 최적화하기 위한 방법도 필요할 것이다. 향후 연구로는 이같이 다수의 행동트리 혹은 다수의 에이전트를 위한 효용 기반 행동트리를 위한 최적화 방법을 들 수 있다.

REFERENCE

[1] M. Dawe, S. Gargolinski, L. Dicken, T. Humphreys, and D. Mark, “Behavior Selection Algorithms An: Overview,” *Game AI Pro: Collected Wisdom of Game AI Professionals*, A K Peters/CRC Press, pp. 47-60, 2013.

[2] D. Hilburn, “Simulating Behavior Trees: A

Behavior Tree/Planner Hybrid Approach," *Game AI Pro: Collected Wisdom of Game AI Professionals*, A K Peters/CRC Press, pp. 99-111, 2013.

[3] Handling Complexity in the Halo 2 AI(2005), https://www.gamedeveloper.com/programming/gdc-2005-proceeding-handling-complexity-in-the-i-halo-2-i-ai (accessed Februaray 1, 2022).

[4] Building a Better Battle: HALO 3 AI Objectives, https://aarmstrong.org/files/gdc2008/Building%20a%20Better%20Battle%20HALO%203%20AI%20Objectives.pdf (accessed February 1, 2022).

[5] D.R. Graham, "An Introduction to Utility Theory," *Game AI Pro: Collected Wisdom of Game AI Professionals*, A K Peters/CRC Press, pp. 113-126, 2013.

[6] Unity Technologies, Animation System Overview, https://docs.unity3d.com/Manual/AnimationOverview.html (accessed February 2, 2022).

[7] Epic Games, "Behavior Trees," https://docs.unrealengine.com/en-US/Engine/ArtificialIntelligence/BehaviorTrees/index.html (accessed February 2, 2022).

[8] M. Colledanchise and P. Ogren, *Behavior Trees in Robotics and AI: An Introduction*, CRC Press, 2018.

[9] A. Shoulson, F.M. Garcia, M. Jones, R. Mead, and N.I. Badler, "Parameterizing behavior trees," *International Conference on Motion in Games*, pp. 144-155, 2011.

[10] J. Seo and U. Yang, "Procedural Behavior Model using Behavior Tree in Virtual Reality Applications," *Journal of Multimedia Information system*, Vol. 6, No. 4, pp. 179-184, 2019.

[11] B. Merrill, "Building Utility Decisions into Your Existing Behavior Tree," *Game AI Pro: Collected Wisdom of Game AI Professionals*, A K Peters/CRC Press, pp. 127-136, 2013.

[12] C.U. Lim, R. Baumgarten, and S. Colton, "Evolving behavior trees for the commercial game DEFCON," *European Conference on the Application of Evolutionary Computations*, pp. 100-110, 2010.

[13] R. Dey and C. Child, "QL-BT: Enhancing Behavior Tree Design and Implemtntation with Q-Learning," *IEEE Conference on Computational Intelligence in Games*, pp. 1-8, 2013.

[14] J. Kwon and J. Jang, "A Study on Implementation of Intelligent Character for MMORPG using Genetic Algorithm and Neural Networks," *Journal of Korea Multimedia Society*, Vol. 10, No. 5, pp. 631-641, 2007.

[15] Get started with Behavior Trees, https://gamedev-resources.com/get-started-with-behavior-trees (accessed December 12, 2021).

권 민 지

2019년 3월~현재 동의대학교 게임애니메이션공학전공 학사과정

2019년 2월 부산 남산 고등학교 졸업

서 진 석

2005년 9월~현재 동의대학교 게임애니메이션공학전공 교수

2005년 2월~2005년 8월 포스텍 박사후연구원

2005년 2월 포스텍 컴퓨터공학과 박사

2000년 2월 포스텍 컴퓨터공학과 석사

1998년 2월 건국대학교 전자계산학과 학사