Journal of Multimedia Information System VOL. 10, NO. 4, December 2023 (pp. 321-332): ISSN 2383-7632 (Online)
https://doi.org/10.33851/JMIS.2023.10.4.321

GOBT: A Synergistic Approach to Game AI Using Goal-Oriented and Utility-Based Planning in Behavior Trees

Yoosung Hong[1], Tianhao Yan[2], Jinseok Seo[3*]

Abstract

In this paper, we propose a novel game AI framework, the Goal-Oriented Behavior Tree, using Unity game engine for simulations. This framework integrates the advantages of the Goal-Oriented Action Planning architecture and Utility Theory with traditional Behavior Trees, enabling more flexible agent responses to various situations. The simulated environment contains customizable game characters capable of actions like patrol, attack, retreat etc. GOBT allows developers to design agent decision-making processes by applying logic in traditional BTs and using the dynamic planning capabilities of GOAP and utility-based action selection when necessary. The performance of GOBT framework is verified through simulations using a synthetic dataset of agent behaviors in response to changing environmental factors.

Key Words: Behavior Tree, GOAP, Utility System, Artificial Intelligence.

I. INTRODUCTION

In gaming, Artificial Intelligence (AI) endows virtual agents with the capability to emulate intelligent behavior, thereby providing players with the impression that the game world responds in a manner akin to reality. This engenders a predictable and logical gaming experience. To achieve these objectives, game developers utilize an array of AI frameworks for decision-making processes. Among these frameworks, Behavior Trees (BTs) are noteworthy for their hierarchical structure, which is efficacious not only in orchestrating the actions of game characters but also in affording the advantage of intuitive visual interpretations [8].

Hierarchical Task Networks (HTN) is another AI architecture that utilizes hierarchical structures similar to BT to design agent behaviors. Both HTN and BT enable structuring behaviors hierarchically. However, BTs have a more intuitive flow-chart like structure using various node types while HTN relies on task decomposition relationships [9-10]. Also, BTs allow dynamically switching behaviors based on state using decorators, whereas HTN generates full plans upfront, resulting in less flexibility. Overall, BTs tend to have better usability in terms of intuitiveness, authoring, debugging, and reactiveness compared to HTN.

Nonetheless, BTs present their own array of challenges. Specifically, they can become intractable in large-scale games owing to the exponential escalation of tree complexity that arises from the combinatorial growth of behavioral elements. Furthermore, their inherent static nature, established at a specific stage of development, constrains their adaptability to dynamic scenarios that were not anticipated [1-2].

The aim of this research is to introduce a new artificial intelligence architecture, termed "Goal-Oriented Behavior Tree (GOBT)." This advanced framework seamlessly combines traditional BTs with Goal-Oriented Action Planning (GOAP), a planning paradigm focused on goal-driven behaviors, as well as with utility systems. The key contribution of our work is a decision-making framework that combines the logical structuring capability of BTs with the dynamic planning of GOAP and utility-driven action selection. By introducing a new planner node with these capabilities into BTs, GOBT enables agents to make contextually optimal choices. As a result of this tripartite amalgamation, the GOBT framework enables a decision-making process that is not only in alignment with the intentions explicitly defined by developers, but also sufficiently versatile to permit AI agents to adapt dynamically across a wide range of operational scenarios. The synthesis of traditional BT's intuitive structure, GOAP's dynamic planning capabilities, and the utility system's utility-based action selection empowers agents to make diverse and contextually appropriate deci-

Manuscript received September 01, 2023 ; Revised October 03, 2023; Accepted October 25, 2023. (ID No. JMIS-23M-09-035)
Corresponding Author (*): Jinseok Seo, +82-51-890-2712, jsseo@deu.ac.kr
[1]Department of Game Engineering, Dong-eui University, Korea, yoosung5508@gmail.com
[2]Department of Digital Media Engineering, Graduate School, Dong-eui University, Korea, 1416419445@qq.com
[3]Department of Game Engineering, Dong-eui University, Korea, jsseo@deu.ac.kr

sions, even within dynamically changing environments.

This paper presents an in-depth analysis of each framework used within the GOBT and compares their respective contributions. By retaining the simplicity of BTs and incorporating dynamic planning and utility-based action selection, GOBT offers enhanced flexibility and scalability, making it a more capable architecture compared to both BTs and HTN. Additionally, methodologies for effectively integrating these frameworks are proposed, along with the introduction of authoring tools to validate the viability of the GOBT framework. We provide a comparative analysis of the component frameworks through example scenarios and propose effective integration techniques. The viability of GOBT is validated through simulations. Our aim is to offer a comprehensive guide for advancing game AI using this hybrid approach.

The structure of this paper is as follows: Chapter 2 explains the theories and characteristics of each design method through example scenarios; Chapter 3 presents the GOBT framework which integrates the dynamic planning features of GOAP and the action selection functions from utility-based systems into BT; Chapter 4 presents authoring tools implemented for realizing the GOBT framework along with the simulators required for running content based on these tools; Chapter 5 applies example scenarios for comparison with conventional methods; Finally, Chapter 6 concludes by evaluating the proposed methods and discussing future research directions.

II. RESEARCH BACKGROUND

In this study, we aim to comprehensively compare the performance of each AI architecture used in the GOBT framework. To do this, we present appropriate criteria to elucidate the design principles behind the GOBT framework. We discuss traditional major AI architectures such as BTs, GOAP, and utility systems and analyze each based on three factors: intuitiveness, flexibility, and scalability (Table 1). These factors were selected as the core attributes that an AI architecture should offer to developers. We further substantiate our analysis by implementing these architectures in set example scenarios.

Table 1. Three metrics for comparing each method.

Metric	Description
Intuitiveness	Refers to the ease with which developers can grasp and work with an AI framework's concepts, influencing its practicality in designing agent behaviors.
Flexibility	Measures how well an AI framework dynamically adapts to diverse scenarios and complex behaviors without imposing excessive constraints.
Scalability	Assesses an AI framework's ability to efficiently add features for different game situations.

2.1. Example Scenario

The scenario models a situation where an agent identifies a threat from an enemy in a given area and responds accordingly. The agent makes decisions based on various state variables, including detected enemy units, their own ability scores, and the presence or absence of an alert. While conducting reconnaissance, the agent takes into account information about discovers enemies (such as the health and level of enemy units) enemies discovered (the health and level of enemy units) as well as their own status (health and level). If they assess that adequate defense is feasible, they will initiate an attack; otherwise, they will first sound an alarm to request support from allies in other areas before proceeding to attack. An activated alert remains in effect until the enemy threat is neutralized, and agents in other areas mobilize to assist the alerted area until the alarm is deactivated.

By implementing this scenario using BTs, GOAP, and utility-based action selection systems we aim to analyze the intuitiveness, flexibility, and scalability of each architecture.

2.2. Evaluation Methods

The comparison of each architecture is conducted through two detailed scenarios. In the first scenario, the agent is initially placed with a defensible level of enemy unit with low level and health. Subsequently, the level of the enemy unit is raised to assess the agent's response capability. In the second scenario, a change in alarm deactivation is introduced to an agent moving toward an area where an alarm has sounded, and its response is observed (Table 2).

2.3. Implementation with Behavior Tree

BTs serve as a design paradigm that guides an agent's actions through a depth-first search algorithm. This tree structure is composed of nodes that delineate the agent's behaviors, conditions, and corresponding methods [3-4]. Our example scenario was implemented using Behavior Designer [15], a tool that enjoys widespread adoption within the Unity game engine community, as depicted in Fig. 1. The process begins at the root node, where the agent initiates a reconnaissance phase and subsequently enters a specific branch to manage encounters with enemies.

Table 2. Two scenarios to evaluate flexibility.

Scenarios	Description
Scene. 1	Give the agent an initially defensible number of enemy units, but gradually increase the number.
Scene. 2	Give an agent that is traveling to an alarmed zone a change called disarmed.

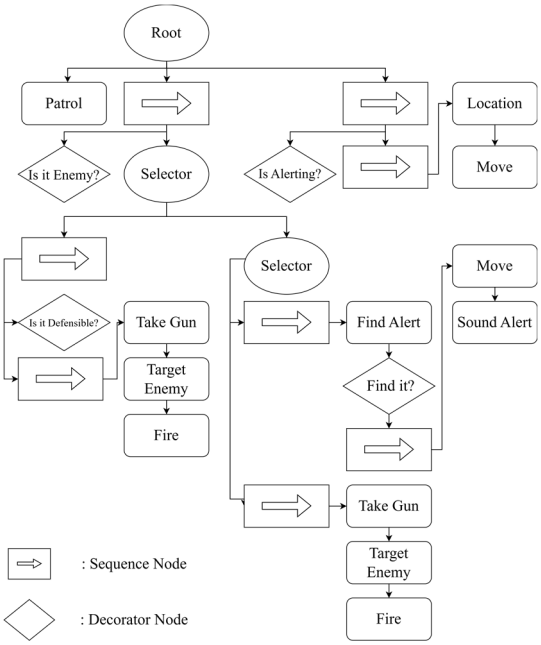

Fig. 1. Example scenario designed using a behavior tree.

Upon encountering an enemy, the agent first evaluates the feasibility of a defensive action based on available information about both itself and the enemy. If defense is deemed viable, the agent initiates an attack sequence. Conversely, if defense is not possible, the agent diverges from the current branch and transitions to another that triggers an alert sequence.

Within this alert activation branch, the agent initially executes a sequence to locate alarms. Upon locating an alarm, it activates it; following this action, an attack sequence is executed irrespective of whether an alarm was found. Additionally, in scenarios where alerts are triggered without enemy detection, the agent executes a support sequence, identifying the alert's origin and proceeding towards that location.

We examine the decision-making outcomes of agents operating within BTs through specified scenarios. To facilitate this analysis, we configure the system to generate log messages every time a node within the BT is traversed. For action nodes, the log specifies the action currently being executed (denoted as Act), while for decorator nodes, the log captures the results of condition checks (indicated as Dec). Additionally, we log changes in two state variables: isDefensible, which assesses the current defensibility of the position, and isAlerting, which indicates whether an alarm is active. These state variables serve as a representation of the agent's immediate contextual circumstances. The output logs for these scenarios are presented in Table 3 for Scene 1 and Table 4 for Scene 2, respectively.

During runtime, the agent identifies an enemy and initiates an attack based on an assessment that the situation is

Table 3. Scene 1 output log with a behavior tree.

	Log
1	Act: Patrol
2	Dec: Is it enemy?: True
4	isDefensible: True
5	Dec: Is it defensible?: True
6	Act: Take Gun
7	Act: Target Enemy
8	Act: Fire
9	...
10	Act: Take gun
11	isDefensible: False
12	Act: Target enemy
13	Act: Fire

Table 4. Scene 2 output log with a behavior tree.

	Log
1	Act: Patrol
2	Dec: Is it enemy? : False
3	isAlerting: True
4	Dec: isAlerting? : True
5	Act: Location
6	isAlerting: False
7	Act: Move
8	...

defensible. However, if the enemy's attributes subsequently improve to an indefensible level, the agent continues to execute the attack sequence without adaptation.

To incorporate a retreat mechanism into the existing BT, a new decorator node is needed to facilitate the decision-making process for retreating. Utilizing existing state variables, this node will assess whether the enemy's capabilities exceed the agent's defensive threshold. If so, the agent ceases alerting and counterattacking activities and transitions to a retreat sequence. It is important to note that the introduction of this retreat function necessitates a reorganization of the existing tree to ensure that prior logics, such as sounding an alarm and counterattacking, are not erroneously executed during the retreat.

2.4. Implementation with GOAP

GOAP is a methodology designed for systematic behavior planning and execution, encompassing of world states, goals, and actions. A world state serves as a state variable that encapsulates both the current condition of the agent and the surrounding environment. Goals specify the desired outcomes the agent aims to achieve, while actions represent the executable tasks that lead to goal attainment.

Upon determining an agent's goal based on the information in the world state, GOAP employs planning algorithms, such as the A* search algorithm to identify the optimal sequence of actions needed to transition from the ini-

tial to the target state. Each action within GOAP is associated with specific pre-conditions and post-events. An action can only be executed, if its pre-conditions are met. Furthermore, to satisfy these pre-conditions, another action featuring a post-event that fulfills them must be executed first.

Thus, planning in GOAP generates a coherent sequence of actions, taking into account the requisite pre-conditions and subsequent post-events to achieve the designated goals. GOAP continuously monitors world states, enabling it to adapt its objectives as circumstances evolve. Should objectives change during plan execution, GOAP abandons the current plan and formulates a new one. This flexibility ensures that GOAP maintains high-level situational responsiveness even in dynamic settings, empowering agents to formulate context-appropriate action plans [16].

GOAP was implemented by leveraging the "Unity GOAP Packages" [17]. Fig. 2 illustrates the process by which plans are formulated in GOAP, specifically for achieving the goal of eliminating enemies, denoted by the condition enemiesCount==0, within a given scenario.

In the initial step, the Fire action, which can fulfill the target state, is selected from the available action set. Subsequently, the Target Enemy action meeting the pre-condition of the Fire action, is chosen. This is followed by the selection of the Take Gun action, which satisfies the pre-condition for the Target Enemy action. The initial state allows for the choice of Take Gun, as it meets its own pre-condition, namely enemyFound==true.

In the context of GOAP, multiple plans can be generated to achieve a singular goal. To identify an optimal plan, the total costs associated with executing each action in every plan are compared. These costs may encompass time spent and resources utilized. The standard for cost can vary, leading to the selection of different plans [11]. Fig. 3 illustrates the creation of two such plans and the selection of one, using arbitrarily assigned costs for demonstrative purposes.

GOAP facilitates agent decision-making by dynamically generating action sequences from a predefined action set, based on ever-changing world states and goals. Consequently, it is not feasible to visualize the entire behavioral flow. Even when world states and goals are predictable, multiple action plans may emerge. Selecting the optimal

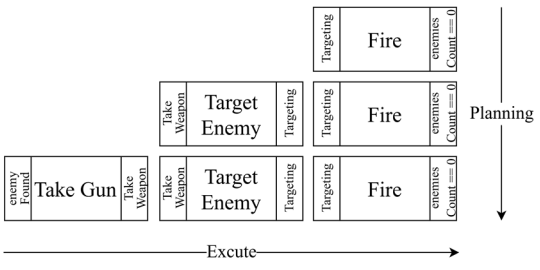

Fig. 2. Process of creating an action chain.

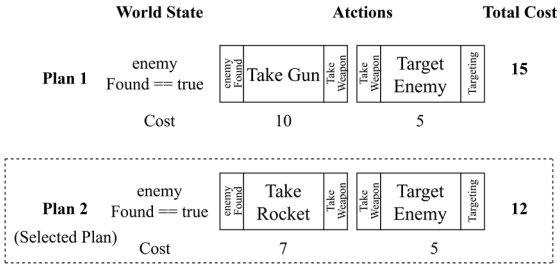

Fig. 3. The plan with the lower total cost is selected.

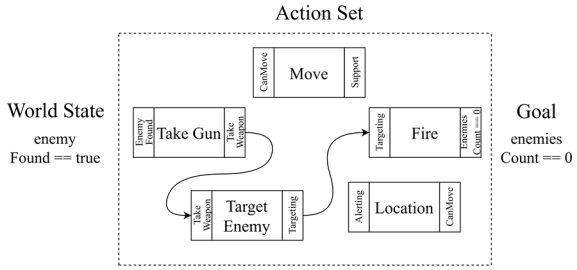

Fig. 4. A plan generated from a set of actions via world states and goals.

plan among these requires a comprehensive consideration of various factors complicating the prediction and understanding of the agent's behavior.

Fig. 4 illustrates potential plans generated from the action set under the condition that the world state is enemyFound==true and the goal is enemiesCount==0.

The log output messages generated from scenarios implemented with GOAP are presented in Table 5 and Table 6. Similar to the BT framework, these logs indicate the agent's situational context through variables such as isDefensible and isAlerting. They also display the name of the action that is both generated and executed to achieve the agent's set goal.

In operation, the initial agent establishes its goal as scouting. Upon encountering an enemy, the agent reconfigures its goal to achieving a state where enemiesCount==0 for defense purposes, subsequently generating and executing

Table 5. Scene 1 output log with GOAP implementation.

	Log
1	Goal : Patrol
2	isDefensible : true
3	Goal : enemiesCount==0
4	Plan : Take Gun, Target Enemy, Fire
5	Act : Take Gun
6	isDefensible : false
7	Goal : isAlerting==true
8	Plan : Find Alert, Move, Sound Alert
9	Act : Find Alert
10	Act : Move
11	Act : Sound Alert

Table 6. Scene 2 output log with GOAP implementation.

	Log
1	Goal : Patrol
2	isAlerting : true
3	Goal : Support
4	Plan : Location, Move
5	Act : Location
6	isAlerting : false
7	Goal : Patrol
8	...

an action plan. If the enemy's level escalates to a point where defense is unfeasible, the agent promptly shifts its goal to isAlerting==true, interrupts the ongoing action sequence, and formulates and executes a new plan.

If you incorporate a retreat function into GOAP, you simply add a retreat action to the existing action set and specify the conditions under which this action can be activated, along with its consequent effect. For instance, you may introduce world state variables such as enemyPowerful and isSafe, setting the pre-conditions for the retreat action to enemyPowerful==true, and the post-event to isSafe ==true. Such extension in GOAP does not necessitate significant alterations to the overall system architecture. When adding specialized functions, structural or code changes are generally unnecessary, thus enabling developers to enhance or diversify agent behavior patterns through relatively straightforward adjustments.

2.5. Implementation with Utility-Based Behavior Selection System

Utility-based behavior selection systems serve as decision-making models in machine learning, drawing on the well-established principles of utility theory. Within this theory, all potential actions and situational contexts can be quantitatively represented through normalized utility values, which signify preferences or values associated with particular actions or situations [5].

In such a system, an agent identifies and carries out the action that possesses the highest utility value among the available options in a given situation. Central to this system is the utility function, which ascertains the relative worth or preference of various behaviors according to specific situational criteria. Due to the normalization of utility values, this method facilitates straightforward comparisons across all actions. Moreover, the use of standardized values renders the statistical analysis and management of these utility values more convenient [12].

Consequently, utility-based behavior selection systems promote optimal decision-making by utilizing utility values pertinent to each specific situation.

$$U = \max(U_1, U_2, U_3). \quad (1)$$

$$U_A = \frac{\max((A_l \cdot w_l + A_h \cdot (1-w_l)) - (E_l \cdot w_l + E_h \cdot (1-w_l)), 0)}{2 + (E_l \cdot w_l + E_h \cdot (1-w_l))}. \quad (2)$$

In an agent governed by a utility-based behavior selection system, utility values for each available action are computed using state variables pertinent to the current context. These variables may include the agent's health and level, as well as the enemy's health, level, and alert status. The action with the highest utility value is subsequently selected and executed. The utility function employed in this context can be expressed as given in Equation (1). This overarching utility function is derived by evaluating three distinct sub-utility functions and selecting their maximum value. Specifically, U1 computes the utility of an action aimed at attacking an enemy for defensive purposes, U2 computes the utility of sounding an alarm, and U3 computes the utility of a supportive action. Equation (2) provides an exemplary formulation for U1. In this equation, Ah and Al are state variables denoting the agent's health and level, respectively, while Eh and El denotes the enemy's health and level. These state variables are updated in real-time and utilized in the utility function for decision-making. The variable W serves as a weight, indicating the relative importance of level values, and similarly functions as a weight for health values. Developers have the flexibility to modulate the influence of each state variable through the adjustment of these weights.

The utility values obtained can be utilized by other utility functions, thereby elucidating the interrelationships among various behavior or state variables. Nonetheless, predicting the agent's behavior, which is dynamically influenced by these state variables in diverse situations and conditions, poses a challenge [13].

Log output messages generated from executing the scenario with a utility-based behavior selection system are presented in Table 7 and Table 8. When the priority of executable actions shifts, the system is configured to output the utility value corresponding to each action.

In scenarios where the alarm is inactive, support actions

Table 7. Utility system output Log: Scene 1.

	Log
1	isDefensible : true
2	UT : Attack : 0.857
3	UT : Alerting : 0.325
4	UT : Support : 0
5	Act : Attack
6	UT : Take Weapon : 0.733
7	isDefensible : false
8	UT : Attack : 0.45
9	UT : Alerting : 0.682
10	UT : Support : 0
11	. . .

Table 8. Utility system output Log: Scene 2.

	Log
1	isAlerting : true
2	UT : Attack : 0
3	UT : Alerting : 0
4	UT : Support : 0.9
5	Act : Support
6	isAlerting : false
7	UT : Attack : 0
8	UT : Alerting : 0
9	UT : Support : 0
10	...

should be disregarded. Similarly in absence of enemy threats, both attack and alarm actions should be executed from consideration. This enables the agent to compute the utility values of potential actions in real-time, based on fluctuating state variables. Should the utility value of an ongoing action fall below that of previously evaluated actions, the agent may discontinue the current behavior and select an alternative course of action.

To incorporate a new feature such as retreat into a utility-based behavior selection system, one can extend the existing framework by introducing additional state variables and altering utility function formulas. However, it is important to note that these modifications may necessitate increased computational resources. Specifically, the computational complexity can grow exponentially with the addition of state variables and modifications to utility functions, particularly when calculating utility values in real-time.

2.6. Comparison

BTs enable the logical structuring of scenarios through a hierarchical and intuitive framework. However, their design limits them to pre-defined behaviors, rendering them incapable of adapting to unpredictable situations. Changes in the upper nodes of the tree structure are often necessary to adapt to new conditions, but these modifications can propagate to lower nodes, thereby constraining the system's scalability [6,14].

GOAP offers a more dynamic approach by creating action plans that consider both the surrounding environment and the agent's current situation. This allows for system expansion without affecting the existing architecture. However, predicting the flow of agent behavior within a GOAP system can be challenging. Utility-based behavior selection systems offer another avenue by quantifying game factors as state variables, which are then integrated to assess the validity of each action in the current context and to select optimal actions flexibly. Despite these advantages, such systems may not always yield optimal results due to the static nature of utility function formulas. Moreover, while adjustments to utility function formulas can facilitate system expansion, they also complicate the understanding of the agent decision-making process for developers and planners, owing to issues of system resource allocation and the relative interpretation of utility values (Table 9) [7].

Table 9. Feature comparison across AI architectures.

Criteria	AI architecture		
	Behavior tree	GOAP	Utility system
Intuitiveness	High	Low	Medium
Flexibility	Low	High	Medium
Scalability	Low	High	High

III. GOBT

In this chapter, we propose the GOBT, a framework that amalgamates the intuitiveness of BTs, the dynamic planning capabilities of GOAP, and the utility-based action selection of utility-based systems. Our analysis in Chapter 2 revealed that while BTs excel in intuitiveness, they suffer from limitations in flexibility and scalability. To address these shortcomings, GOBT enhances agent adaptability by incorporating dynamic planning within the BT structure and by fortifying goal-setting and action selection through utility-based mechanisms.

By integrating these elements, the GOBT framework allows developers to create a comprehensive logic structure for agent AI. It facilitates effective handling of nuanced action control through its dynamic planning and utility-based action selection functionalities. This integration thus offers a balanced and flexible approach to AI agent decision-making.

3.1. Methodology

The design methodology of the GOBT unfolds in several stages. Initially, a planner node incorporating dynamic planning and utility-based action selection is introduced into the BT. This node manages complex branching possibilities within the BT, selects suitable goals based on situational variables, and ensures the feasibility of these goals. Subsequently, while traditional GOAP aims to satisfy world states, GOBT prioritizes action handling for branching by choosing and executing actions specified in the planner node.

Thirdly, all actions specify their target, thereby segmenting the action set to reduce search costs associated with goal achievement. Fourth, one target action is chosen through a utility-based action selection mechanism, and an initial action satisfying the current agent state’s pre-conditions is executed. Execution then proceeds towards the target actions, navigating through pre-conditions and after-effects using utility-based behavior selection. Lastly, the planner nodes

comprise sets of target actions, executable actions, and state variables that facilitate both goal setting and behavior selection.

3.2. Dynamic Planning Function and Utility-based Action Selection Function

The extant GOAP framework employs reverse design to generate an action sequence optimizing it based on the state variables at the planning stage. However, this approach faces a limitation: it cannot ensure that the actions executed are optimal for the current situation, given that the sequence was optimized at the time of planning. To address this limitation, our Goal-Oriented Behavior Tree (GOBT) framework integrates optimal goal selection in real-time through state variables. GOBT incorporates concepts from GOAP, such as goal resetting according to situational changes, causal relationships between actions through pre-conditions and after-effects, and utility-based behavior selection mechanisms.

In contrast to traditional dynamic planning, GOBT's dynamic planning structure is distinct. Traditional planning pre-determines an optimal action sequence for a goal and executes it accordingly. In GOBT, once a goal is set, an appropriate starting action is selected based on the current state. All plans leading to the target actions are then generated through pre-conditions and after-effects. Utility values of subsequent actions are updated in real-time through state variables and utility functions. Upon the completion of each action, the agent executes the subsequent action with the highest utility value. This process iteratively leads to the execution of target actions based on real-time optimal procedures. If the utility value of a target action changes during the process, new goals are set, and the aforementioned procedure is repeated. If the sequences is interrupted by external factors but the same target actions are selected again, agents can resume from the point of interruption.

The efficiency of this method is attributable to the characteristics of GOBT. While traditional GOAP would require significant system resources to generate all plans, GOBT optimizes this process by conducting these calculations on specific, subdivided branches of the BT through planner nodes, thus operating under lighter computational conditions.

Fig. 5 and 6 illustrate the dynamic planning and action selection process with GOBT. Fig. 5 delineates how target actions set on planner nodes are dynamically planned, while Fig. 6 shows the process of selecting and executing one of the posterior actions based on state variables and utility functions.

In summary, by amalgamating features from GOAP and utility-based mechanisms into BTs, GOBT offers developers a tool for implementing versatile and flexible action

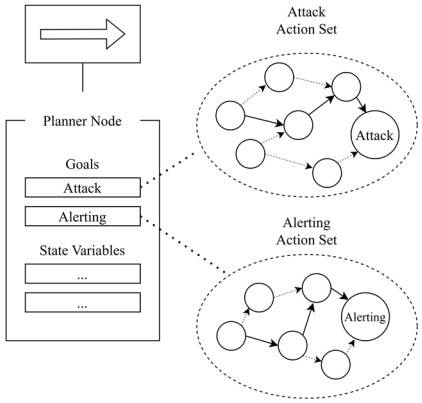

Fig. 5. Goal action planning via the planner node.

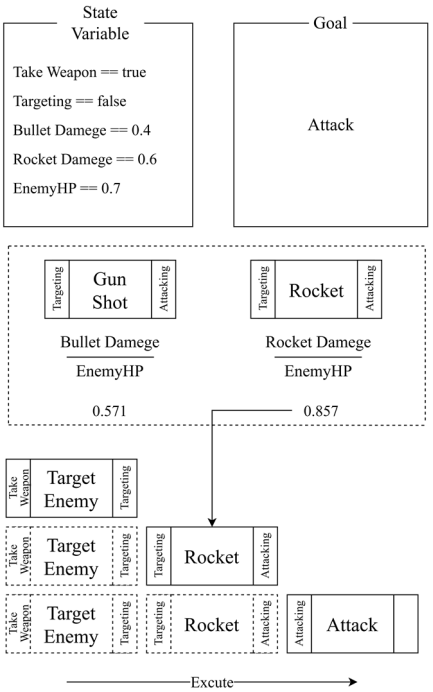

Fig. 6. Utility-based agent decision-making.

plans within hierarchical logical structures. Designing GOBT necessitates several key considerations: the clear definition of the set of actions capable of achieving the target action, as well as their pre-conditions and after-effects for feasibility and result prediction. Furthermore, given the application of utility-based mechanisms, careful deliberation is required to determine how utility values should be calculated and updated, as decision-making in GOBT is contingent upon these values.

3.3. Algorithm Complexity

The core algorithms underpinning the proposed GOBT framework are dynamic planning and utility-based action selection. This section analyzes their time and space complexities.

During the dynamic planning process, all potential paths leading to the goal action are explored. If we let b represent the average branching factor of actions and d denote the maximum depth of the search space, the worst-case time complexity becomes O(bd). This is because the entire search space might need traversal in the most complex scenarios.

For the utility-based action selection, the time complexity is linear, O(n), where n stands for the number of actions. This is due to the utility computation being directly proportional to the count of actions.

In terms of memory usage, additional space is necessitated for storing both the actions and state variables. As the number of actions and variables increases, the memory footprint grows linearly.

When implementing the GOBT framework, it's crucial to design the action complexity and depth judiciously to ensure the exploration space remains manageable. Moreover, by limiting the number of actions and variables to only essential ones, space efficiency can be optimized. With proper management of complexities, the proposed algorithms can be scaled efficiently for real-world applications.

In conclusion, the dynamic planning in GOBT employs a goal-directed search complemented by utility guidance, ensuring flexible adaptation. Through hierarchical decomposition, the integrated algorithm adeptly balances between responsiveness and scalability.

IV. GOBT AUTHORING TOOL AND SIMULATOR

To validate the efficacy of the GOBT framework, we developed a prototype tool and simulator using Behavior Designer. As GOBT integrates both BT authoring and simulation functionalities, we incorporated dynamic planning and utility-based action selection features in accordance with the framework's design.

4.1. Implementation of Dynamic Planning Function

The planner node, depicted in Fig. 7, serves as crux of the GOBT framework. This node is implemented by inheriting from the terminal node class utilized in Behavior Designer. Fig. 8 illustrates a screenshot where planner nodes

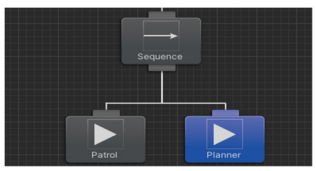

Fig. 7. The blue node in the bottom right is a planner node.

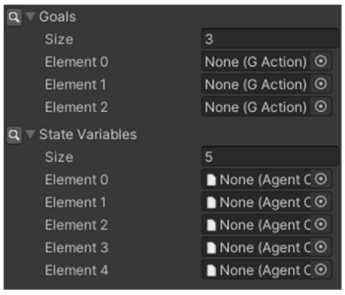

Fig. 8. Unity editor's inspector pane of a planner node.

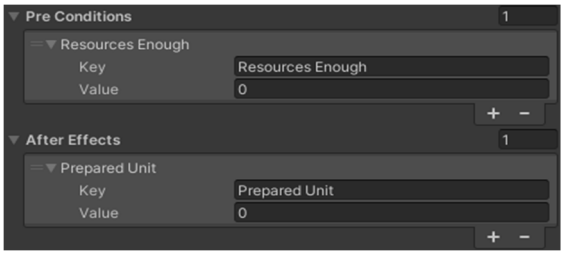

Fig. 9. Inspector pane for an action object composed of pre-conditions and after-effects.

are situated within the authoring tool. To configure a planner node, developers are initially required to establish a goals list and a state variables list. Both of these elements are made accessible via the Unity editor's inspector window, thereby facilitating their setup.

Within the planner node, the optimal target action is selected from an action set, followed by the determination of a starting action based on current state variables. Subsequently, all plausible paths from the starting to the target action are mapped out. It is noteworthy that both the target action and the planning sequence components are objects inherited from the GAction class, which itself is a descendant of the MonoBehaviour class. This allows for the setting of pre-conditions and after-effects directly within the Unity editor's inspector window. As depicted in Fig. 9, for instance, Resources Enough is set as a pre-condition, while Prepared Unit is defined as an after-effect. Unlike GOAP, where actions are selected based on predefined costs, our approach relies on utility as elaborated in the following section.

4.2. Implementation of Utility-Based Behavior Selection Function

In the GOBT framework, state variables configured in the planner node facilitate utility-based action selection. These state variables are created by developers through inheritance from an abstract class termed AgentConsideration, which itself inherits from Unity's ScriptableObject class. As such, each state variable object is a descendant of ScriptableObject and can be managed as an asset within Unity projects. When configuring a planner node, developers merely need to select from these pre-saved state variable assets. This asset-based approach enables different planner

nodes to share the same considerations.

Developers implementing AgentConsideration must furnish methods that update or return values for use as state variables. For example, in the study presented herein, state variable objects were created for both the number and health of defensive units, as well as enemy units.

Finally, child classes of GAction class must implement a utility function, referred to as the UpdateUtility method. This function calculates and returns utility values based on the aforementioned state variables. Utility-based action selection in planner nodes is realized by invoking these specialized utility functions. In scenarios where multiple actions yield the highest utility value, a random selection among these actions is made.

V. IMPLEMENTING AN EXAMPLE SCENARIO WITH GOBT

In this chapter, we elaborate upon and operationalize the example scenario initially introduced in Chapter 2. We employ the implemented authoring tool and simulator to evaluate the performance of the GOBT framework, and we scrutinize the log messages generated during the simulation.

The extended scenario is designed as follows: The agent possesses two attack tools, a gun and a rocket. Initially, the gun's attack power is configured to be stronger than that of rocket. However, during actual combat, the rocket's attack power is dynamically adjusted to surpass the gun's. Should the firing action be successfully executed, the enemy's ability score increases, consistent with the original scenario.

To assess the adaptability of the GOBT framework, we configured the scenario so that the optimal attack method would fluctuate based on specific state variables. We compared the performance of GOBT and GOAP to quantitatively evaluate the improvements in flexibility offered by GOBT. Fig. 10 depicts the implementation of the expanded scenario using GOBT, while Fig. 11 illustrates the causal relationships between the actions employed in the scenario.

For this extended scenario, new state variables were introduced to represent the attack power of the gun and the rocket. These state variables assign utility values to both

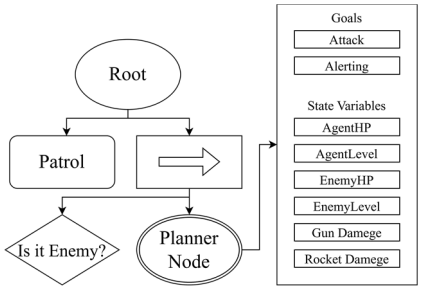

Fig. 10. Extended scenario implemented with GOBT.

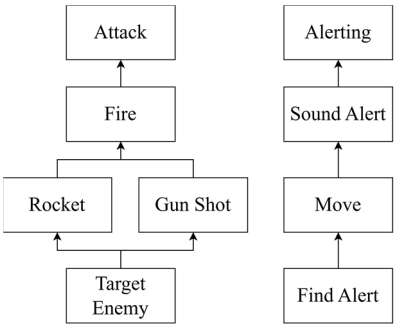

Fig. 11. Causality between actions in GOBT.

weapons, thereby influencing the agent's choice during attack sequence. As detailed in Chapter 4, target actions and state variable lists were configured in the inspector window of the planner nodes. This modularization allows for a streamlined expression of complex decision logic compared to scattering it throughout the BT structure. To substantiate the dynamic planning capabilities of GOBT, we juxtaposed this extended scenario against a traditional GOAP implementation. The comparative analysis is presented in Table 10 and Table 11.

Upon reviewing the output log messages, a marked distinction emerges between GOBT and traditional GOAP in terms of planning and execution. GOAP generates an optimal action sequence predicated on pre-established costs, and adheres to this sequence unless the goal changes. For instance, even if the gun's attack power is reduced during the Target Enemy action, the Gun Shot action continues as planned. In contrast, GOBT perpetually monitors the utility values associated with target actions. Should a change in priority occur, It is GOBT that adjusts the action sequence, not GOAP. This demonstrates that utility-based behavior selection in GOBT ensures a higher degree of responsiveness

Table 10. GOAP extended output Log.

	Log
1	Goal : Patrol
2	isDefensible : true
3	Goal : enemiesCount==0
4	Plan : Target enemy, gun shot, fire, attack
5	Act : Target enemy
6	Act : Gun shot
7	Act : Fire
8	isDefensible : false
9	Goal : isAlerting==true
10	Plan : Target enemy, gun shot, fire
11	Act : Sound alert
12	Act : Move
13	...

Table 11. GOBT extended output log.

	Log
1	Act : Patrol
2	Dec : is it enemy? : True
3	isDefensible : True
4	UT : Attack : 0.685
5	UT : Alerting : 0.492
6	Goal : Attack
7	Start action : Target enemy
8	Act : Target enemy
9	UT : Rocket : 0.75
10	UT : Gun shot : 0.23
11	Act : Rocket
12	Act : Fire
13	isDefensible : true
14	UT : Attack : 0.42
15	UT : Alerting : 0.822
16	Goal : Alerting
17	Start action : Find alert
18	...

compared to the dynamic planning in traditional GOAP.

VI. CONCLUSION

The GOBT framework offers enhanced situational adaptability, similar to the GOAP architecture, by incorporating dynamic planning and utility-based action selection into traditional Behavior Trees (BTs). This integration is realized through planner nodes, which quantify the utility of prospective actions within the current context. These utility values, alongside real-time situational factors, inform the agent's logical decision-making process. The metric for action selection, represented by the utility value, is determined by a range of state variables configured by developers. This configuration permits fine-tuned behavioral adjustments according to specific circumstances. In terms of scalability and maintainability, the addition of new conditions or actions to planner nodes is straightforward, thereby obviating the need for comprehensive structural revisions as is often required in classical BTs.

This study proposes the GOBT framework, a hybrid system that integrates salient features from GOAP and utility-based architectures with traditional BTs. We conducted evaluations of established AI frameworks, including BTs, GOAP, and utility-focused action selection systems. Our analysis identified areas of potential improvement in existing BTs and substantiated the effectiveness of the GOBT framework in realistic gaming environments.

Future work will focus on extending and diversifying the applications and maintenance of the GOBT framework. Specific challenges within the framework will be addressed, with an emphasis on mitigating real-time action selection delays attributed to utility calculations, potentially through the use of heuristic functions. Additionally, exploring the dimensions of cooperation and competition in multi-agent configurations, as well as incorporating real-time data analysis and machine learning, will be critical for enhancing an agent's learning capabilities.

ACKNOWLEDGMENT

This work was supported by Dong-eui University Grant (202301420001).

REFERENCES

[1] B. Merrill, *Building Utility Decisions into Your Existing Behavior Tree*. Boca Raton, FL: CRC Press, 2019.

[2] M. Dawe, S. Gargolinski, L. Dicken, T. Humphreys, and D. Mark, *Behavior Selection Algorithms an Overview.* Boca Raton, FL: CRC Press, 2015.

[3] D. Hilburn, *Simulating Behavior Trees*. Boca Raton, FL: CRC Press., 2019.

[4] M. Dawe, *Real-World Behavior Trees in Script*. Boca Raton FL: CRC Press., 2019.

[5] D. Graham, *An Introduction to Utility Theory.* Boca Raton Fla: CRC Press., 2019.

[6] M. Colledanchise and P. Ögren, *Behavior Trees in Robotics and AI: An Introduction*. Boca Raton FL:CRC Press., 2018.

[7] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Upper Saddle River NJ: Pearson, 2020.

[8] R. H. Abive, I. Günsel, N. Akkaya, E. Aytac, A. Cagman, and S. Abizada, "Robot soccer control using behaviour trees and fuzzy logic," *Procedia Computer Science*, vol. 102, pp. 477-484, 2016.

[9] O. Ilghami, D. Nau, H. Muñoz-Avila, and D. Aha, "Learning preconditions for planning from plan traces and HTN structure," *Computational Intelligence*, vol. 21, 2005.

[10] N. Nejati, T. Könik, and U. Kuter, "A goal- and dependency-directed algorithm for learning hierarchical task networks," in *Proceedings of the ACM Conference,* 2009, pp. 113-120.

[11] J. Orkin, "Agent architecture considerations for real-time planning in games," in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment,* 2005, pp. 105-110.

[12] M. J. Kwon and J. S. Seo, "Game AI agents using deliberative behavior tree based on utility theory", *Journal of Korea Multimedia Society*, vol. 25 no. 3, pp. 432-439, Feb. 2022.

[13] J. G. Han and S. J. Cheon, "Improving the accuracy of

cold item recommendations by applying recommendation diversification methods," *Journal of Korea Multimedia Society*, vol. 25, no. 8 pp. 1242-1250, Aug. 2022.

[14] M. Colledanchise and P. Ögren, "How behavior trees generalize the teleo-reactive paradigm and or-trees," in *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots Systems* (IROS), Boca Raton, 2016, pp. 5045-5052.

[15] Behavior Designer-Behavior Trees for Everyone, https://assetstoreunity.com/packages/tools/visual-scripting/behavior-designer-behavior-trees-for-everyone-15277, 2008.

[16] J. Orkin, 2004. "Symbolic Representation of Game World State: Toward Real-Time Planning Games," http:// alumni.media.mit.edu/~jorkin/WS404OrkinJ.pdf

[17] Goap, 2021. https://assetstore.unity.com/packages/tools/ behavior-ai/goap -252687

AUTHORS

Yoosung Hong is presently a student in the Game Engineering at Dong-eui University. He completed his secondary education at Yeoncho High School, graduating in February 2019.

Tianhao Yan is presently enrolled in a graduate program in Digital Media Engineering at Dong-eui University. He completed his undergraduate studies in Game Animation Engineering at the same institution, receiving his degree in December 2021.

Jinseok Seo received his BS degree in 1998 from Konkuk University, Korea. He subsequently obtained his MS and PhD degrees from the Department of Computer Science and Engineering at Pohang University of Science and Technology (POSTECH), Korea, in 2000 and 2005, respectively. Since 2005, he has been affiliated with the Department of Game Engineering at Dong-eui University, Korea, where he currently holds the position of a professor. His primary research interests include in virtual reality, augmented reality, and game AI algorithms.