Journal of Multimedia Information System VOL. 6, NO. 4, December 2019 (pp. 179-184): ISSN 2383-7632 (Online)
http://doi.org/10.33851/JMIS.2019.6.1.179

Procedural Behavior Model using Behavior Tree in Virtual Reality Applications

Jinseok Seo[1*], Ungyeon Yang[2]

Abstract

This paper introduces a study for procedurally generating the behavior of objects in a virtual environment at runtime. This study was initiated to enable the behavioral model of objects in virtual reality applications to evolve in response to user behavior at runtime. Our approach is to describe the behavior of an object as a behavior tree, and to make a node of the behavior tree change to another type if a certain condition is satisfied. We defined four types of node changes: "parameterized", "probabilistic", "alternate", and "variant". We experimented with a virtual environment that includes a variety of simple procedural elements to explore the possibilities of our approach. As a result of the implementation, if an optimization algorithm that can select and apply the optimized procedural elements in response to the user's behavior is complemented, it is confirmed that more intelligent objects and agents can be implemented in virtual reality applications.

Key Words: Behavior tree, Behavior model, Procedural behavior, VR authoring.

I. INTRODUCTION

In order for virtual reality to be accepted as mainstream media, it is necessary to increase the quantitative and qualitative levels of the virtual reality contents. However, planning, designing and implementing virtual reality contents requires a lot of time and money with the help of experts in various fields. In addition, development methods that rely on existing computer programming languages such as C ++ and C # have limitations in increasing content productivity. As the virtual reality market grows, it is faced with the demand for massive scalability along with the quantitative demand, which is also very important in this case.

In order to increase the productivity of contents, the virtual reality and game industries have included authoring tools based on various formalization tools in the production engine. Examples include Mecanim in Unity and Blueprint in Unreal Engine. Unity's Mecanim uses a hierarchical state machine to create animations that change as objects change their states, while Unreal's Blueprint is a massive extension of the data flow diagram that allows you to visually program the behavior of an object.

Formalization tools such as state machines and data flow diagrams can express the content in a more computational form as compared to hard coding. Therefore, in order to procedurally create an event or scenario of an object, the adoption of such a formalization tool is an essential element. For this reason, in this study, we propose a method for procedurally generating the behavior of an object using behavior tree.

Formalization tools such as state machines and data flow diagrams can express the content in a more computational form as compared to hard coding. Therefore, in order to procedurally create an event or scenario of an object, the adoption of such a formalization tool is an essential element. For this reason, in this study, we propose a method for procedurally generating the behavior of an object using behavior tree.

II. RELATED WORKS

In the game industry, research on procedural content generation (PCG) has been actively conducted to increase

Manuscript received November 28, 2019; Accepted December 20, 2019. (ID No. JMIS-19M-11-038)
Corresponding Author (*): Jinseok Seo, 176 Eomgwangno, Busanjin-gu, Busan 47340, Korea, +82-51-890-2712, jsseo@deu.ac.kr.
[1]Game Engineering Major, Dong-eui University, Busan, Korea, jsseo@deu.ac.kr
[2]Electronics and Telecommunications Research Institute, Daejeon, Korea, uyyang@etri.re.kr

the productivity of content. PCG, first proposed at the IT University of Copenhagen, is applied to a wide variety of fields, including game level [1], object appearance [2], game quest [3], and the like.

In this study, we tried to procedurally create the behavior model of virtual objects. Similar attempts can be seen in the case study for Non-Player Characters (NPCs). In [4], the quest of NPC is generated procedurally based on motivation, resources, and intimacy. In [5], FuSM (Fuzzy State Machine) is improved to adapt NPC to user's response through genetic algorithm and neural network.

As in this study, the problem of procedurally generating events and scenarios of objects is very similar to the planning problem in artificial intelligence. This is because procedurally generating events requires finding a sequence of actions to accomplish a particular purpose. A representative example of the planning algorithm is SHOP2 [6]. SHOP2(Simple Hierarchical Ordered Planner 2) is a planning system based on Hierarchical Task Network (HTN). As another example of using HTN, [2] showed that procedural scenario creation techniques using HTN in lifesaving training games can generate various procedural scenarios in an environment with the unexpected behavior of obstacles.

The behavior tree is not only easy to implement in an object-oriented language-based development environment, but also has high scalability and reusability due to the modular structure. In addition, because it is natural to apply computational operations to its components, behavior tree is well suited as a means for procedural generation. In [7], they introduce the application of parameterization to behavior tree and a tool called Topiary, which enables authoring of behavior tree as a testbed for agent simulation. An example of applying computational operations to behavior tree is a case where a genetic algorithm is used to optimize the components of the behavior tree [8]. There is also an example of studies that allow NPCs or agents to learn on their own using various techniques of reinforcement learning [9].

II. SYSTEM OVERVIEW

The final goal of this study is to procedurally generate the behavior model of an object at runtime. This final goal, Procedural Behavior Engine, consists of two parts: Behavior Simulator and Behavior Mediator. Behavior Simulator analyzes and simulates the behavior model formalized by behavior tree, and Behavior Mediator evaluates the actions of a user and delivers the results to Behavior Simulator.

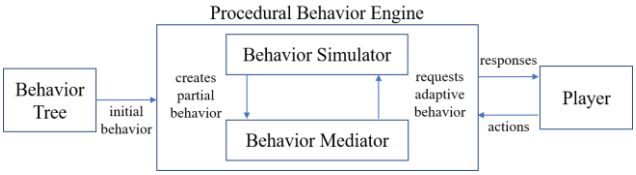

Fig. 1. Overall structure of our final system, Procedural Behavior Engine.

This paper focuses on the process of simulating procedural behavior models in Behavior Simulator. Behavior Mediator, which is responsible for evaluating the results of user behavior and requesting appropriate procedural behavior models, is left for further study. Fig. 1 shows the structure of our final system.

III. PROCEDURAL ELEMENTS OF BEHAVIOR TREE

For an object model formalized by a behavior tree to change procedurally at runtime, procedural elements must be derived. Detailed procedural elements and generation rules will be finalized when the Behavior Mediator mentioned in Chapter 3 above is completed, but this paper presents four types of procedural elements to examine the possibilities of our idea. Currently derived elements are “parameterized”, “stochastic”, “mutated” and “substitute”.

3.1. Parameterized Node

We first determined the parameterization introduced in [7] as the first procedural element. In this study, we use parameters for conditional task nodes and action task nodes as in Fig. 2. If no parameter is passed from the outside, the simulation is done with default parameters, which simulates the same as fixed task nodes in the unparameterized behavior tree. The Behavior Mediator will play a role in optimizing these parameters.

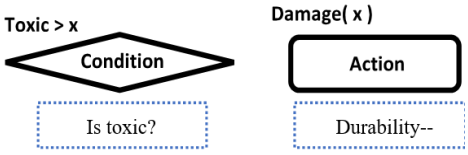

Fig. 2. Parameterized conditional node and action node in behavior tree.

3.2. Stochastic Node

Stochastic node selects one node to execute among its child nodes. Currently, one node is selected by predefined ratio, but later, the Behavior Mediator should select child nodes by stochastic reasoning based on an inference engine that will be developed later.

Journal of Multimedia Information System VOL. 6, NO. 4, December 2019 (pp. 179-184): ISSN 2383-7632 (Online)
http://doi.org/10.33851/JMIS.2019.6.1.179

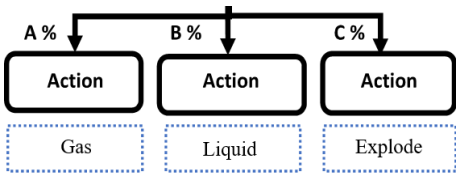

Fig. 3. Stochastic action nodes in behavior tree.

3.3. Mutated Node

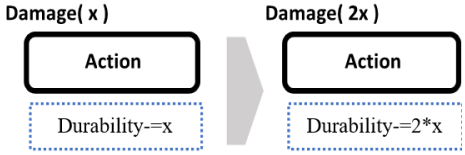

Fig. 4. Damage action node mutating at runtime.

Mutated Nodes change the values of various variables (including parameters) of the node itself. Variables that can change its values include instant, abort type, repeat forever, and end of failure. The values of these variables of mutated nodes are also currently set by random probabilities, but the Behavior Mediator will determine this later.

3.4. Substitute Node

Substitute Node is a sub-tree that can replace some node and a sub-tree rooted at it at runtime. After constructing a pool of candidate sub-trees that can replace that node, select an appropriate sub-tree from this pool and replace it with an existing node.

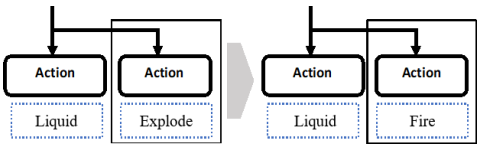

Fig. 5. “Fire” action node substituting “Explode” action node at runtime.

IV. IMPLEMENTATION RESULTS

4.1. Implementing Procedural Behavior Tree Simulator

We implemented a simple procedural behavior tree simulator to examine the possibilities of our idea. The simulator acts as a component of a game object in the Unity engine. We used Opsive's Behavior Designer [10] for the basic simulation and authoring functions of the behavior tree, which is very suitable for implementing our idea because it is an asset that provides source code. Each node of the behavior tree implemented in this study inherits from the classes provided by the Behavior Designer.

There are quite a few modules we need to implement for procedural element definition and simulation of behavior tree (for example, the asset format for procedural elements, an asset generator, an asset parser, etc.), but the most critical part is the function to replace one node with another. Node replacement is implemented in five stages such as "save progress", "stop behavior tree simulation", "replace task node", "resume behavior tree", and "restore progress".

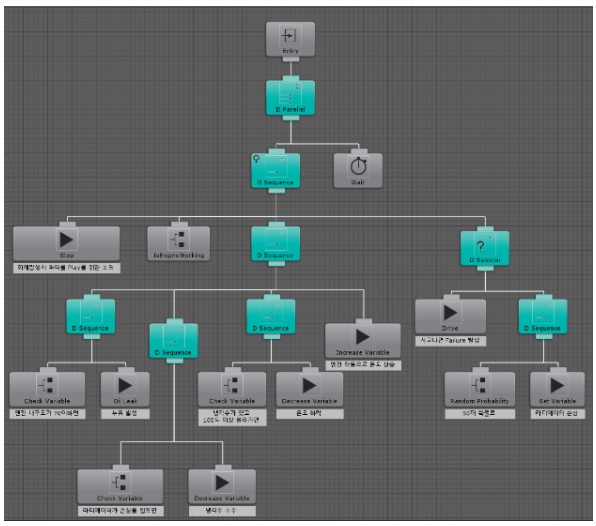

Fig. 6. Behavior tree of a car for fire drill training system.

4.2. An Example Content: Fire Drill Simulation

To test the completed procedural behavior tree simulator, we created a fire drill training content. This content is implemented by modeling various environments and objects that can cause a fire, and users are trained to cope with unexpected situations in virtual environments.

In this paper, the implementation results will be explained focusing on the car object that causes a fire due to internal causes. Fig. 6 shows the behavior tree of the car for the basic simulation. This behavior tree represents a state in which the temperature inside the car’s engine room rises when the simulation starts, but it operates safely while the radiator is not damaged.

4.3. Implementing Procedural Elements

From the behavior tree of the car described above, we derive some procedural elements that include parameterized, stochastic, and substitute nodes. Among them, parameterized and stochastic nodes are implemented by inheriting from the class library of Opsive’s Behavior Designer. Substitute sub-trees are implemented separately from the main action tree, which is stored as a separate asset so that it can be referred to as an external behavior tree when needed. Fig. 7 shows the authoring tool window for these external behavior trees, that is substitute sub-trees. In this window, we specify which candidate in the sub-tree pool can replace an existing node when certain conditions are met.

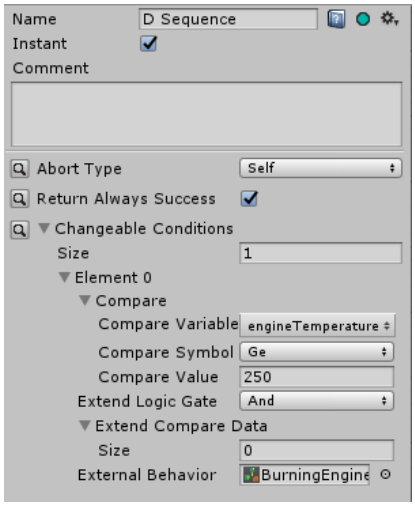

Fig. 7. Authoring tool for procedural elements.

Fig.8 shows the two substitute sub-trees we implemented. The left one is a sub-tree that first causes a fire in the engine room to spreads to another part, and the right one describes the situation where the fire spreads to the engine after the fire occurs elsewhere first.

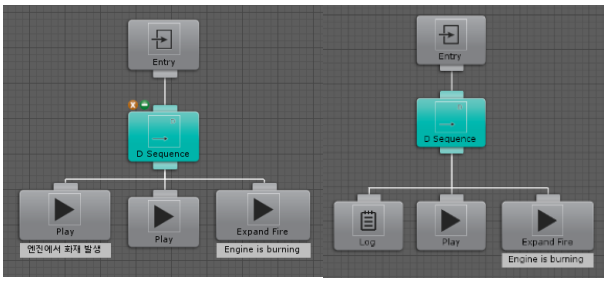

Fig. 8. Two substitute sub-trees of the car object's behavior tree.

4.4. Simulation Results

Fig. 9 is a screenshot of our simulation results. As the durability of the car decreases gradually, the radiator is damaged. Then, the internal temperature rises rapidly due to the coolant leak, causing a fire and an explosion.

Fig. 9. Screenshot of the fire drill training system.

V. CONCLUSION

In this paper, we showed a procedural method for generating behavioral models of objects. We first formalized the behavior model of an object into a behavior tree and applicable procedural sub-tree were derived from four predefined types of procedural elements. Then, based on the behavior tree and the sub-tree, we showed that in a commercial game engine Unity, the behavioral model of the object can be varied according to specific conditions at runtime to create various scenarios.

Acknowledgement

This work was supported by the National Research Foundation of Korea (NRF) grant funded by the Korea government (MSIT) (No. NRF-2019R1F1A1041854), and by Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIP) (2019-0-01347, Development of Realistic Fire Training Content Technology to Help Simulate Fire Sites and Improve Command Capabilities).

REFERENCES

[1] Shaker, N., Yannakakis, G. N., & Togelius, J., "Digging deeper into platform game level design: session size and sequential features," in *European Conference on the Applications of Evolutionary Computation*, Springer, Berlin, Heidelberg, pp. 275-284, 2012.

[2] Liapis, A., Yannakakis, G. N., & Togelius, J., "Adapting models of visual aesthetics for personalized content creation," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 3, 213-228, 2012.

[3] Lee, Y. S., & Cho, S. B., "Context-aware petri net for dynamic procedural content generation in role-playing game," *IEEE Computational Intelligence Magazine*, vol. 6, no. 2, pp. 16-25, 2011.

[4] Jung, B., Cho, S., and Kang, S., "Procedural Quest Generation by NPC in MMORPG," *Journal of Korea Game Society*, vol. 14, no. 1, pp. 19-28, 2014.

[5] Kwon J. and Jang J., "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.

[6] Nau D. S., Au T. C., Ilghami O., Kuter U., Murdock, J. W., Wu D., and Yaman F., "SHOP2: An HTN planning system," *Journal of artificial intelligence research*, vol. 20, pp. 379-404, 2003.

[7] Shoulson A., Garcia F. M., Jones M., Mead R., and Badler N. I., "Parameterizing behavior trees.," in *International Conference on Motion in Games*, Springer, Berlin, Heidelberg, pp. 144-155, 2011.
[8] Lim C. U., Baumgarten R., and Colton S., "Evolving behaviour trees for the commercial game DEFCON," in *European Conference on the Applications of Evolutionary Computation*, Springer, Berlin, Heidelberg, pp. 100-110, 2010.
[9] S Dey R., and Child C., "Ql-bt: Enhancing behaviour tree design and implementation with q-learning," in *2013 IEEE Conference on Computational Intelligence in Games (CIG)*, pp. 1-8, 2013.
[10] https://opsive.com/solutions/ai-solution/, 2019

Authors

Jinseok Seo received his BS degree from Konkuk University, Korea, in 1998 and an MS and a PhD degree in the Department of Computer Science and Engineering from Pohang University of Science and Technology (POSTECH), Korea, in 2000 and 2005, respectively. In 2005, he joined the Department of Game Engineering at Dong-eui University, Korea where he is currently a professor.

His research interests include virtual reality, augmented reality, and game AI algorithms.

Ungyeon Yang received his BS degree in computer science and engineering from Chungnam National University, Daejeon, Rep. of Korea, in 1997. He received his MS and PhD degrees from Pohang University of Science and Technology (POSTECH), Rep. of Korea, in 2000 and 2003, respectively. Since 2003, he has been a principal researcher with Electronics and Telecommunications Research Institute (ETRI).

His research interests include wearable display, information visualization, 3D user interfaces, human factors, haptics and multimodal user interaction in the field of virtual/mixed reality and ergonomics.