IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 7 August 2023, accepted 3 September 2023, date of publication 11 September 2023,
date of current version 21 September 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3313737

RESEARCH ARTICLE

Character Behavior Automation Using Deep Reinforcement Learning

HYUNKI LEE, MWAMBA KASONGO DAHOUDA, AND INWHEE JOE
Department of Computer Science, Hanyang University, Seoul 04763, South Korea

Corresponding author: Inwhee Joe (iwjoe@hanyang.ac.kr)

This work was supported by the Institute of Information Communications Technology Planning Evaluation (IITP) grant funded by the Korean Government [Ministry of Science and ICT (MSIT)] (Development of the Technology to Automate the Recommendations for Big Data Analytic Models that Define Data Characteristics and Problems) under Grant 2020-0-00107.

ABSTRACT Recently, various new attempts are being made to improve the quality of media content according to the expansion of the media market. Pre-visualization is one of those attempts, and the behavior of characters (agents) in virtual space is essential for pre-visualization. In this paper, a study was conducted to automatically generate behaviors of virtual characters for more efficient visualization in pre-visualization. In particular, we propose a method to automatically produce an appropriate behavior by detecting the state of the surrounding environment with a deep reinforcement learning technique. A virtual environment is created using a game engine to configure space for reinforcement learning, and a reinforcement learning model of the training environment is configured with Python and PyTorch. The virtual environment and the model training environment are communicated with the ML-agents toolkit. In the virtual environment, the character basically moves in a straight line, and three obstacles appear at random locations in front of the character. The character senses 9 states and allows 5 actions. After that, a reward is offered according to the action to proceed with learning. For performance evaluation, reinforcement learning training was conducted using the Proximal Policy Optimization (PPO) algorithm and Soft Actor-Critic (SAC) algorithm, and performance comparisons were also conducted according to the batch size. As a result, we are able to secure a reinforcement learning model with obstacle avoidance capability. Applying the model to the character proved that the character can automatically animate according to the state of the surrounding environment without explicit programming.

INDEX TERMS Pre-visualization, deep reinforcement learning, behavior.

I. INTRODUCTION

The media environment is rapidly changing as digital media convergence based on high-speed Internet and advanced information and communication technology enables high-quality media content services using various media devices and platforms [1]. In particular, Over-the-Top (OTT), a representative platform for Internet media convergence, is expanding the content market by changing the value creation structure of the media industry and creating an active content consumption culture for users [2]. This paradigm shift in the media industry is expanding to the entire media content industry such as drama, film, entertainment, and behavior [3].

The associate editor coordinating the review of this manuscript and approving it for publication was Jiachen Yang.

Furthermore, as digital transformation is in full swing with the era of the 4th Industrial Revolution, the platform market is expected to expand further and the boundaries between the media industries are rapidly broken as next-generation media content using artificial intelligence, big-data, metaverse, and virtual reality are added [4]. Meanwhile, in order to continuously provide services that meet the rapidly changing needs of users in various media platform markets and expand the reach of media works, it is necessary to reduce production time and production cost as well as high-quality content production. Accordingly, as a way to cope with the trend of the media content industry, the importance of pre-visualization or previz work before production using computer programs is growing [5], and in this regard, pre-visualization research of 3D behavior types using computer graphics has been actively

conducted. However, although the field of application for pre-visualization has expanded and various types of application methods are being developed accordingly, related research is insufficient quantitatively and qualitatively. In addition, most characters (Agents) are programmed to respond passively under fixed virtual environment conditions, so there is a limit to making behaviors that actively respond to a given environment as in reality. Therefore, in order to further maximize the efficiency and performance of the pre-production work, a pre-visualization plan in which the character actively responds in a virtual environment similar to reality is required. Our contributions can be summarized as follows:

- The character can only travel straight in the training scenario configuration, and the three obstacles emerge in front of it at random within a predefined range in order to automatically generate the right behavior. In order to detect a total of nine states, the character was required to do five acts.
- We classify Proximal Policy Optimization (PPO), an on-policy algorithm, and Soft Actor-Critic (SAC), an off-policy method, as policy algorithms for reinforcement learning. The episode is designed to finish instantly if it encounters an obstacle, and as a result, we have devised a travel distance reward, a weighted incentive for the posture to go as quickly as possible, and a penalty policy if it moves backward.
- Using the Unity game engine, we build a virtual environment for the reinforcement learning model learning in order to verify the proposed pre-visualization model.
- We compared and examined the training results for each reinforcement learning algorithm based on the batch size in order to verify performance.

The paper is organized as follows: Section II presents related research and describes algorithms such as reinforcement learning used in this study and tools used such as Unity. Section III describes our proposed Design and the Implementation of the basic elements of reinforcement learning such as Environment, State, Agent, Action, and Reward. The following section IV, the performance evaluation, presents the configuration of the experimental environment, and the training results of reinforcement learning along with the comparison of training results according to the batch size. In section V, the conclusion summarizes the results of the study, the limitations of the study, and presents future research.

II. RELATED WORK

Recently, research using deep learning, a key element technology of the 4th Industrial Revolution, and deep reinforcement learning, which combines reinforcement learning, has been actively conducted [6], [7]. Reinforcement learning is a machine learning technique in which characters directly explore and recognize the state of the surrounding environment in a given virtual environment and learn through interaction with the environment [8], [9], [10], [11]; in other words, the agent interacts with the game environment, receives feedback in the form of rewards based on its actions, and learns to maximize its long-term cumulative reward. Particularly in the context of playing games, deep reinforcement learning (DRL) has been a fascinating topic of study. DRL has excelled over human performance in a number of difficult games, achieving exceptional outcomes in the gaming world. Deep reinforcement learning is effective in games because it can handle high-dimensional input, such as raw pixel data, and learn complex strategies through self-play and exploration. The deep learning model for successfully learning control rules from high-dimensional sensory input was provided by the authors in [12].

Another intriguing study was conducted in [13], where the author looked at how a stimulating environment can aid in developing complicated behavior. They taught the agents specifically in various environmental circumstances, and they discovered that doing so promoted the creation of robust behavior that excelled at a variety of tasks. When complex interaction is required, environments that mimic the physical characteristics of the target domain typically the real world are frequently required as well. For challenges where the objective is to translate a policy learned in a simulator to the actual world [14], [15], [16], as would be the case for the majority of robotics applications, this realism is crucial.

This study is meaningful in applying deep reinforcement learning to pre-visualization tasks by differentiating it from passive pre-visualization systems using existing 3D character behavior. This study proposes an automatic method of creating character behavior using deep reinforcement learning as a way to improve the efficiency of pre-visualization work. In the proposed pre-visualization model, characters can automatically create behaviors according to the state of the surrounding environment without explicit programming. In this study, we intend to create a virtual environment using a game engine to construct a space for reinforcement learning and to proceed with learning by providing rewards according to the behavior of the character.

In this case, the game engine uses the ML-Agents Toolkit 0.29.0 (Machine Learning Agents Toolkit) which allows us to create or use pre-made environments to train our agents, provided by Unity [17] while the reinforcement learning model environment consists of Python 3.9, PyTorch 1.21.1.

ML-Agents has two important components as described in Figure 1, the first is the learning environment (LE), on Unity, which contains the Unity scene and the environment elements; the second is the Python API which contains the RL algorithms (such as PPO and SAC). Therefore, we can use this API to launch training since it communicates with the Learning environment through the external communicator. Inside the Learning Environment, we have different elements: The first is the Agent, each Agent can have a unique set of states and observations, take unique actions within the environment, and receive unique rewards for events within the environment. Here, the actions of an agent are decided

by the brain which is linked to. The second element in the LE is the Brain, each Brain defines a specific state and action space, and is responsible for deciding which actions each of its linked agents will take. The Brain has 4 modes: External, Internal (Experimental), Player, and Heuristic; therefore, we used external mode because, in this mode, action decisions are made using the ML library through communication over an open socket with our Python API. The Last element in LE is the Academy, it orchestrates agents and their decision-making process.

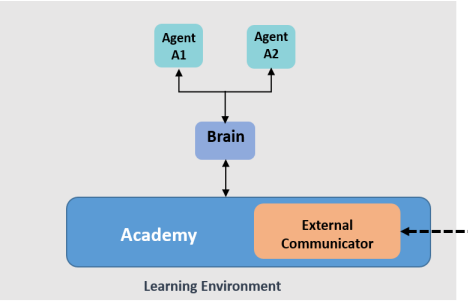

FIGURE 1. Unity ML-Agents.

We compared and analyzed the training results according to the batch size for the on-policy algorithm Proximal Policy Optimization (PPO) and Off-Policy algorithm Soft Actor-Critic (SAC) [18], [19] as policy algorithms for reinforcement learning.

III. DESIGN AND IMPLEMENTATION

In this paper, for automatic behavior generation, we present a plan divided by environment, state, agent, action, and reward, which are the basic elements of reinforcement learning. First, the environment describes the virtual environment in which reinforcement learning proceeds, second, the agent describes the type of state and the purpose of design that the agent identifies, and third, the agent describes basic actions and episode settings, decision cycles, etc. Fourth, actions describe actions that agents can take, and fifth, rewards describe reward policies designed for proper behavior generation.

A. ENVIRONMENT

Environment refers to an object or problem to be solved using reinforcement learning. In this paper, the purpose is to learn to automatically generate appropriate behaviors in a given environment. In front of the character, three obstacles, an upper obstacle, a lower small obstacle, and a lower large obstacle, appeared randomly at an arbitrary position within 8 meters as described in Figure 2.

B. STATE

In this paper, we proposed to use a total of nine states for learning, and the nine states are:

1) Distance to eye height obstacles;
2) Distance from down position obstruction;

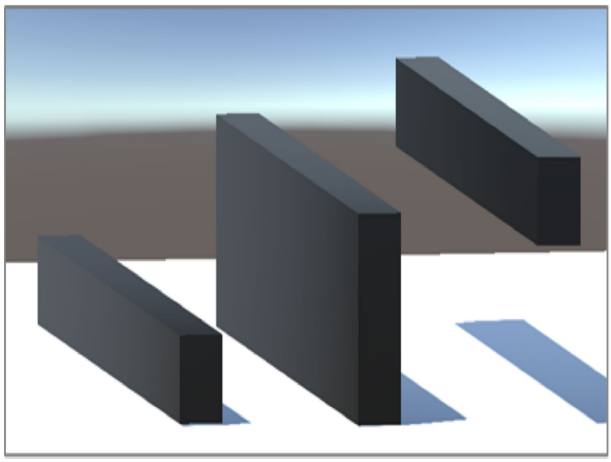

FIGURE 2. Model overview.

3) Distance from obstruction up front 30 degrees;
4) Distance from rear top 30-degree directional obstruction;
5) Upward 90-degree distance from obstruction;
6) Forward acceleration;
7) Upward acceleration;
8) Down Position Obstacle Height;
9) Current action type

Each state is (1) identifying the upper obstruction, (2) identifying the lower Obstacles, (3) determining when to take a sitting action, (4) determining when to switch actions that take place, (5) determining the duration of a sitting action, (6) calculating the appropriate forward speed, (7) calculating the appropriate jump strength, (8) jump strength calculation, (9) identifying the action the character is currently taking. In this way, the character is trained to recognize nine states and output appropriate actions and is configured to avoid them by automatically directing appropriate behaviors for a given obstacle as shown in Figure 3.

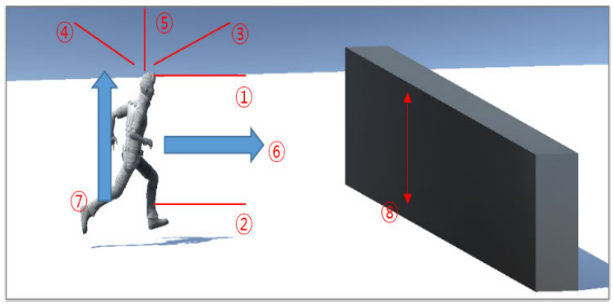

FIGURE 3. Types of states.

In our experiment process, we used the Markov Decision Process (MDP), since it represents a sequential decision problem in which behavior affects the next state and outcome. Therefore, MDPs are general and flexible enough to formulate the problem of learning goals through interaction, the same problem solved by reinforcement learning. We can

express and infer reinforcement learning problems from an MDP perspective. The MDP consists of 4-tuples (S, A, P, R). S is a state space with a finite set of states, A is an action space with a finite set of actions, and P is a transition function that defines the probability of reaching the state s' through action a from s.

$$P(s', s, a) = p(s'|s, a) \tag{1}$$

In Equation (1), the transition function is equal to the conditional probability of s' given s and a. R is a reward function that determines the value received to switch from state s to state s' after taking action. For practical purposes, assume MDP and a finite number of transfer state $(S_t, S_{t-1}, S_{t-2}, .., S_{t-k})$ can be used to solve the problem. Such a system is partially observable and its state is called observation. The final goal of MDP is to find a policy that maximizes cumulative compensation as shown in Equation (2). R is the reward obtained at each step along the policy. It is a harm to MDP when the policy does the best possible action in each state of the MDP. This policy is called the optimal policy.

$$\sum_{t=1}^{\infty} R_{\pi}(S_t, S_{t+1}) \tag{2}$$

C. AGENT

In this paper, the character is basically configured to move only straight as shown in Figure 4. As the character moves straight, it identifies the state and avoids obstacles with appropriate behavior such as walking, running, and jumping. Because the goal of the agent in this study is to move forward without being hit by obstacles for as long as possible, the learning step is set to a maximum of 5,000 times, and the episode ends if it is more than 5,000 times. The observation-determination-action-reward cycle is repeated every time an agent makes a decision request, and in this study, the decision request cycle is designed in five steps because it does not need to be controlled as finely as a robot's joint.

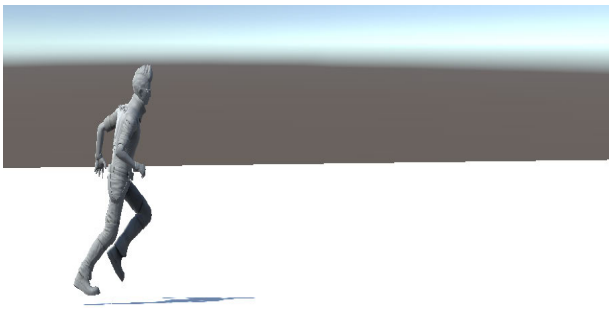

FIGURE 4. Moving character.

D. ACTION

It is the design content of the action that the character can take. There are a total of five actions that can be performed: walking, running, sitting, jumping, and jumping intensity. Each action was designed to move forward, avoid upper obstacles, and avoid lower obstacles. The character can avoid obstacles in front of him through the appropriate output of five actions.

E. REWARD

The following is about the reward strategy and the end of the episode. Rewards were designed with travel distance rewards, posture weights, and reverse penalties. The end of the episode is when an obstacle collision or maximum step is reached.

1) DISTANCE REWARD STRATEGY

The distance reward strategy is defined as a reward provided according to the distance traveled. It is a reward strategy that allows the agent to learn that it must move forward without stopping. Distance return policy is defined as follows:

$$R_{dist} = P_t - P_{t-1} \tag{3}$$

R_{dist} represents a reward for distance, P_t represents the position of the current step character, and P_{t-1} represents the position of the character in the previous step.

2) STANDING REWARD STRATEGY

It is a reward strategy that allows you to move more quickly than if you are moving in a standing position, so if you are in a standing position, you can learn to move standing up if possible by weighting the distance traveled. The Standing reward strategy is defined as follows:

$$R_a = R_a + R_{dist} * W_{stand} \tag{4}$$

R_a represents an accumulated reward, P_{dist} represents an acquisition reward for a distance, and W_{stand} represents the weight when standing.

3) PENALTY REWARD STRATEGY

It is a reward strategy that deducts rewards by giving more penalties to the distance between the characters when they go back so that they can learn to move forward. The penalty return policy is defined as follows:

$$R_a = R_a + R_{dist} * W_{back} \tag{5}$$

R_a represents the cumulative reward, R_{dist} represents a reward for distance, and W_{back} represents the penalty weight when moving backward.

4) EPISODE TERMINATING POLICY

This study aims to automatically avoid obstacles. Therefore, if the obstacle is not avoided, the episode is immediately terminated so that no more rewards are obtained. The reason why the episode was terminated without deducting the reward is that the character is taught to obtain the maximum reward. However, if the reward is deducted because the obstacle cannot be avoided, the character tends to take action that does not move. Sometimes, if the character did not move, the

character could not avoid obstacles and get a higher reward than receiving a reward deduction.

IV. PERFORMANCE EVALUATION

A. EXPERIMENTAL CONFIGURATION

The experimental environment can be divided into two main settings. First, Unity and ML-Agents were used as the virtual environment configuration for learning. The character used a virtual laser to identify the conditions such as the distance from the gaze position obstacle, the distance from the lower position obstacle, the distance from the front 30-degree obstacle, the distance from the rear 30-degree obstacle, and the distance from the 90-degree obstacle. Figure 5 shows a virtual environment implemented with Unity. The red line in the figure identifies the distance with a virtual laser. In addition, parallel training was used to shorten learning time. Figure 6 is a diagram of the configuration of parallel training with a virtual environment where agents in multiple virtual environments are linked to a single brain. It was configured so that model learning could proceed in a total of 30 identical virtual environments. The second reinforcement learning model training environment configuration used Python 3.9, PyTorch 1.21.1, and ML-Agents 0.29. Real model learning is done through PyTorch, an open-source machine learning library, which communicates with a virtual learning environment composed of Unity and through the ML-Agent Python API to conduct reinforcement learning.

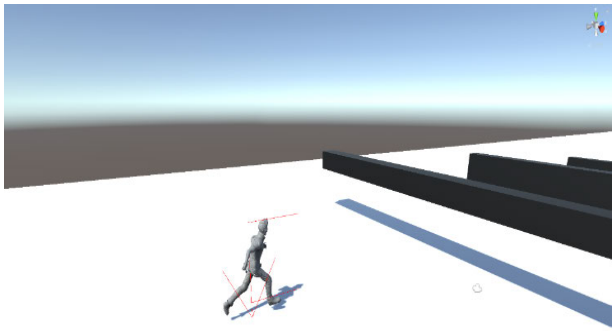

FIGURE 5. Virtual environment configuration.

B. EXPERIMENTS

Reinforcement learning (RL) algorithms can be broadly divided into model-free reinforcement learning and model-based reinforcement learning. Model-free RL is more widely studied; therefore, we chose a model-free RL algorithm. In addition, as described in Table 1, Model-free RL can be further divided into on-policy and off-policy approaches. The on-policy algorithm is similar to the way humans directly explore dangerous areas to determine the path of travel, rejecting hazards as much as possible, while the off-policy

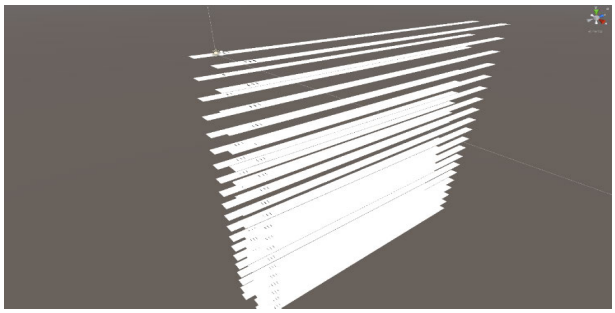

FIGURE 6. Parallel training configuration.

algorithm does not directly explore dangerous areas in a way that humans do not. The experiment was conducted with two main algorithms: PPO, which is the representative algorithm of the on-policy algorithm, and SAC, which is the representative algorithm of the off-policy algorithm. Each algorithm is characterized by a stable and general-purpose algorithm for PPO, and SAC is an algorithm that often requires 5-10 times fewer samples to perform the same task as PPO due to its high sample efficiency.

TABLE 1. Comparison between PPO and SAC.

	PPO	SAC
Base Algorithm	On-Policy	Off-Policy
Environment Type	Discrete/Continuous Behavioral Environment	Continuous Behavioral Environment
Specialities	PRG, Puzzle games	Racing, Shooting games

The following are the primary hyperparameters used by each algorithm. The hyperparameters used in learning with the PPO algorithm are shown in Table 2.

TABLE 2. PPO algorithm hyperparameters.

Num	Hyperparameters	Value
1	batch-size	1024 / 256
2	buffer-size	10240 / 2560
3	memory-size	1024 / 256
4	max-steps	5.0e6
5	learning-rate	3.0e-4
6	learning-rate-schedule	linear
7	num-layers	2
8	time-horizon	64
9	hidden-units	128
10	beta	5.0e-3
11	epsilon	0.2
12	num-epoch	3

The hyperparameters used in learning with the SAC algorithm are also shown in Table 3.

To compare the learning results for each batch size, the PPO algorithm was trained in 256 and 1024 batch sizes whereas the SAC algorithm was trained in 256, 512, and 1024 batch sizes as shown in Figure 7.

In the case of the SAC algorithm, learning was conducted in 256 and 1024 batch sizes in the same way as the PPO

TABLE 3. SAC algorithm hyperparameters.

Num	Hyperparameters	Value
1	batch-size	1024 / 512 / 256
2	buffer-size	10240 / 5120 / 2560
3	memory-size	1024 / 512 / 256
4	max-steps	5.0e6
5	learning-rate	3.0e-4
6	learning-rate-schedule	linear
7	num-layers	2
8	time-horizon	64
9	hidden-units	128
10	num-update	1

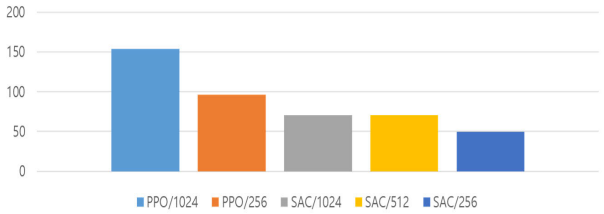

FIGURE 7. Average reward by algorithm and batch size.

algorithm, but it was not converged. So I also trained about 512 batch sizes. Analyze experimental results. In this experiment, the model was stored every 10,000 steps to check the change in the obstacle avoidance ability. As a result of learning the PPO algorithm at 1024 batch size, it was confirmed that the learning was completed and converged at about 3.5M step. As a result of learning the SAC algorithm at 1024 batch size, learning showed an unstable pattern compared to PPO. Although it seemed to converge at about 2M step, the acquisition reward was confirmed to be about 50% compared to the PPO algorithm. In addition, besides the training steps, Table 4 shows the training time, max reward, and the average reword of each algorithm based on batch size.

TABLE 4. Comparion of algorithm by batch size: BS = Batch size, TS = Training Steps, TT = Training Time, Max R = Max Reward, Avg. R = Average Reward.

Algo.	BS	TS	TT	Max R	Avg. R
PPO	1024	5M	1h 33min	534.6	154.1
PPO	256	5M	1h 57min	123.2	95
SAC	1024	5M	6h 5min	333.9	70.82
SAC	512	5M	5h 13min	535.9	70.85
SAC	256	5M	3h 26min	96.08	49.9

As a result of continuous learning, it showed an unstable pattern from about 4M steps, and the acquisition reward was gradually decreasing. We applied the model learned with the PPO algorithm 1024 batch size to the character in the virtual environment and checked it. Figure 8 is the learning trend graph of the PPO and SAC algorithms, the blue is the learning trend graph of the PPO algorithm and the red is the SAC algorithm. As shown in the figure, the PPO algorithm showed a stable learning trend. The SAC algorithm showed instability in the beginning and eventually diverged. As shown in the graph, when the early learning model was applied, it was difficult to avoid obstacles, and could not take appropriate actions. When the final model was applied to the character, it showed stable avoidance of obstacles as shown in Figure 9. Based on these experimental results, it can be said that it is possible to use deep reinforcement learning to automatically generate appropriate behavior by determining the state of the point in time without separate programming.

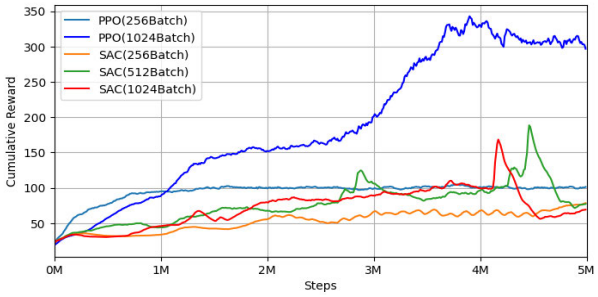

FIGURE 8. Learning progress chart.

As described above, in the case of the PPO algorithm, it was confirmed that the ability to avoid obstacles can be provided to the character when learning with a 1024 batch size. In addition, we tested the PPO algorithm to see what changes are made when changing the batch size. As a result of learning with a 256 batch size, it showed a pattern of learning at a faster pace in the beginning than with the 1024 batch size. After that, a reward was stably obtained from about 500K steps. However, the reward obtained was lower than the 1024 batch size, and the result was less than 50% compared to the learning result of the 1024 batch size. Next, we tested what changes were made when the batch size of the SAC algorithm was changed. Unlike the PPO algorithm, it was not learned and showed instability even in the 1024 batch size, so we conducted learning in 512 batch size and 256 batch size and compared the results. The 512 batch size showed similar trends in the case of the 1024 batch size and the reward acquisition score, but the learning proceeded in a more unstable manner than the 1024 batch size. Like the 1024 batch size, it showed that it did not learn the appropriate avoidance ability. The 256 batch size had a lower acquisition reward score than the 1024 batch size or 512 batch size, and from about 2M steps, learning proceeded steadily in an unstable manner. Similarly, appropriate avoidance skills have not been learned. As a result of the experiment using the PPO and SAC algorithms to adjust the size of 256, 512, and 1024 batches, it was confirmed that the obstacle avoidance problem in the virtual environment was solved by using the PPO algorithm and the 1024 batch size learning model. Table 5 is a reinforcement learning hyperparameter of the final model that can demonstrate the method proposed in this paper.

A well-trained reinforcement model with the ability to avoid obstacles actually showed stable avoidance of obstacles, as shown in Figure 9 when applied to the environment in which learning was conducted. Even when the reinforcement

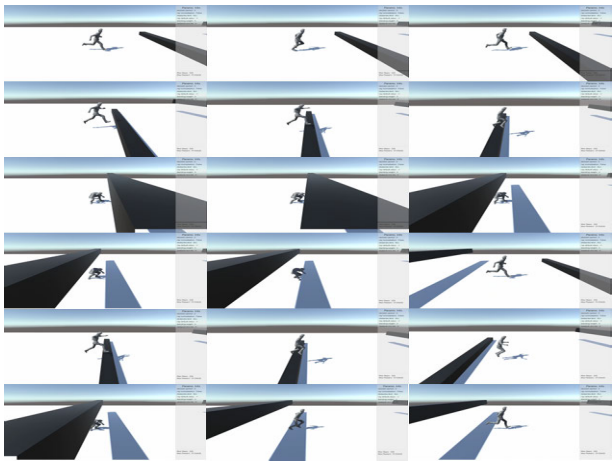

FIGURE 9. Applying final model to character.

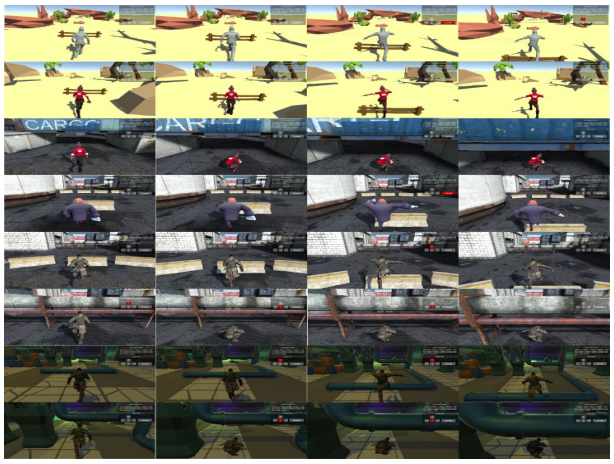

FIGURE 10. Apply the final model to different environments and characters.

TABLE 5. Final hyperparameters.

Num	Hyperparameters	Value
1	batch-size	1024
2	buffer-size	10240
3	memory-size	1024
4	max-steps	5.0e6
5	learning-rate	3.0e-4
6	learning-rate-schedule	linear
7	num-layers	2
8	time-horizon	64
9	hidden-units	128
10	beta	5.0e-3
11	epsilon	0.2
12	num-epoch	3
13	normalize	false
14	sequence-length	64
15	use-recurrent	false
16	vis-encode-type	simple
17	extrinsic-strength	1.0
18	extrinsic-gamma	0.99

learning model is applied to new virtual environments and new characters, the proposal for an automatic behavior generation method in this paper can be proved only when the obstacle avoidance ability is maintained. Thus, we created three additional virtual environments and applied the corresponding reinforcement learning model to three characters to conduct experiments on the ability to avoid obstacles. First, the three new environments added for the experiment are named Environment 1, Environment 2, and Environment 3, respectively. The characters are named Character 1 for the existing characters and Character 2, 3, and 4 for the three newly added characters, respectively. Characters 1, 2, 3, and 4 are different and unrelated 3D models. The experiment first tested the ability to avoid obstacles by applying Character 1, a character used in existing reinforcement learning, to Environment 1. Next, character 2 was applied to Environment 1 to experiment with obstacle avoidance ability, and the experiment was conducted in the order of character 2 in Environment 2, character 3 in Environment 2, and character 4 in Environment 3. Figure 4 is an experimental video of the ability to avoid obstacles by applying the final reinforcement learning model to various virtual environments and different characters. As shown in Figure 10, it was confirmed that the ability to avoid obstacles remains the same when using the PPO algorithm and a reinforcement learning model learned in a batch size of 1024 even when the environment or character changes.

V. CONCLUSION AND DISCUSSION

This study proposes a method to help advance visualization to help create more competitive content in the market by increasing the quality of content in a situation where the size of the media market is growing significantly. In particular, an in-depth reinforcement learning technique was applied to the behavior production of characters essential for pre-visualization. Using the deep reinforcement learning technique, we propose a method in which characters can automatically create behaviors by detecting the state of the surrounding environment on their own. As a method of this study, a space for reinforcement learning was constructed using the game engine Unity. Reinforcement learning model training was constructed using Python and the machine learning library PyTorch. The virtual environment for reinforcement learning composed of game engines and the model training environment composed of Python and PyTorch were communicated using the ml-agents toolkit. The character was able to detect nine states in a virtual environment and avoid obstacles through five actions. Reinforcement learning was conducted by designing a function that provides rewards according to behavior so that characters can automatically avoid obstacles. As for the learning results, it was possible to secure a convergent model when learning with a batch size of 1024 using the PPO algorithm. When learning using the SAC algorithm, convergent learning results could not be obtained even if the batch size was changed. When the model converging with the PPO algorithm was applied to the experimental environment, it was possible to actually

avoid obstacles stably. As a result of the experiment, it was confirmed that even if the environment or character changes, the character to which the reinforcement learning model is applied stably avoids obstacles. As a result of this study, the following points were proved. First, we demonstrate that deep reinforcement learning can be used to automatically generate behaviors in certain situations without the need for separate explicit programming. Second, it is proved that even if the model obtained through reinforcement learning is applied to different environments and characters from the learning environment, it can reliably generate appropriate behaviors. Third, even under the same conditions, it was confirmed that there was a significant difference in learning results depending on hyperparameters such as algorithms and batch sizes. In this study, we conducted experiments on two reinforcement learning algorithms, PPO and SAC, and further experiments on additional reinforcement learning algorithms seem to need to explore more optimal algorithms. In the case of the SAC algorithm, it seems necessary to compare the performance of the PPO algorithm if it eventually failed to learn the ability to avoid obstacles, but succeeded in learning through additional experiments through more various hyperparameter adjustments. This study is expected to contribute to the development of the media market by quickly visualizing text-based data such as movie scripts or novels to create applications that can be used in the actual media content production market.

REFERENCES

[1] J. M. Bauer and M. Latzer, *Handbook on the Economics of the Internet*. Cheltenham, U.K.: Edward Elgar Publishing, 2016.
[2] J. Sujata, S. Sohag, D. Tanu, D. Chintan, P. Shubham, and G. Sumit, "Impact of over the top (OTT) services on telecom service providers," *Indian J. Sci. Technol.*, vol. 8, no. S4, p. 145, Feb. 2015.
[3] L. Jong-Won, "Extension and competition of media platform," KISDI, Jincheon, South Korea, KISDI Premium Rep. 8, 2021.
[4] M. Song, "Over-the-top (OTT) platforms' strategies for two-sided markets in Korea," *Int. J. Internet, Broadcast., Commun.*, vol. 13, no. 4, pp. 55–65, 2021.
[5] P. Seongho, "A study on the role of 3D animated pre-visualization for VFX film production," in *Cartoon Animation Research*, no. 51. 2018, pp. 293–319.
[6] R. S. Sutton and A. G. Barto, "Reinforcement learning: An introduction," *IEEE Trans. Neural Netw.*, vol. 9, no. 5, p. 1054, Sep. 1998.
[7] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (Adaptive Computation and Machine Learning series). Cambridge, MA, USA: MIT Press, 2018.
[8] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," 2015, *arXiv:1509.02971*.
[9] S. Fujimoto, H. van Hoof, and D. Meger, "Addressing function approximation error in actor-critic methods," in *Proc. Int. Conf. Mach. Learn.*, 2018, pp. 1587–1596.
[10] H. Tang, Z. Meng, J. Hao, C. Chen, D. Graves, D. Li, C. Yu, H. Mao, W. Liu, Y. Yang, W. Tao, and L. Wang, "What about inputing policy in value function: Policy representation and policy-extended value function approximator," 2020, *arXiv:2010.09536*.
[11] P. Li, H. Tang, J. Hao, Y. Zheng, X. Fu, and Z. Meng, "ERL-Re^2: Efficient evolutionary reinforcement learning with shared state representation and individual policy representation," 2022, *arXiv:2210.17375*.
[12] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with deep reinforcement learning," 2013, *arXiv:1312.5602*.
[13] N. Heess, S. Sriram, J. Lemmon, J. Merel, G. Wayne, Y. Tassa, T. Erez, Z. Wang, S. M. A. Eslami, M. Riedmiller, and D. Silver, "Emergence of locomotion behaviours in rich environments," 2017, *arXiv:1707.02286*.
[14] A. A. Rusu, M. Vecerik, T. Rothörl, N. Heess, R. Pascanu, and R. Hadsell, "Sim-to-real robot learning from pixels with progressive nets," 2016, *arXiv:1610.04286*.
[15] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," 2017, *arXiv:1703.06907*.
[16] M. Andrychowicz, B. Baker, M. Chociej, R. Jozefowicz, B. McGrew, J. Pachocki, A. Petron, M. Plappert, G. Powell, A. Ray, J. Schneider, S. Sidor, J. Tobin, P. Welinder, L. Weng, and W. Zaremba, "Learning dexterous in-hand manipulation," 2018, *arXiv:1808.00177*.
[17] A. Juliani, V.-P. Berges, E. Teng, A. Cohen, J. Harper, C. Elion, C. Goy, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," 2018, *arXiv:1809.02627*.
[18] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor," 2018, *arXiv:1801.01290*.
[19] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," 2017, *arXiv:1707.06347*.

HYUNKI LEE received the M.S. degree in computer science from Hanyang University, Seoul, South Korea. Since 2014, he has been a Software Engineer with CJ Olivenetworks, Seoul. His current research interests include deep learning, reinforcement learning, generative AI, computer vision, digital human, and augmented reality.

MWAMBA KASONGO DAHOUDA received the B.S. degree in information system engineering from the University Protestant of Lubumbashi, Lubumbashi, Democratic Republic of the Congo, and the M.S. degree in software engineering from Hanyang University, Seoul, South Korea, in 2020, where he is currently pursuing the Ph.D. degree in software engineering. His research interests include artificial intelligence, deep learning, wireless-powered communication networks, and non-terrestrial networks.

INWHEE JOE received the B.S. and M.S. degrees in electronics engineering from Hanyang University, Seoul, South Korea, and the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, GA, USA, in 1998. Since 2002, he has been a Faculty Member with the Department of Computer Science, Hanyang University. His current research interests include the Internet of Things, cellular systems, wireless-powered communication networks, embedded systems, network security, machine learning, and performance evaluation.