Augmenting Exploratory Testing Agents for 3D Software via Imitation Learning

Hansae Ju[1], Scott Uk-Jin Lee[*]
[1]*Dept. of Applied Artificial Intelligence, Major in Bio Artificial Intelligence*
[*]*Dept. of Computer Science & Engineering*
Hanyang University, Ansan, Republic of Korea
{sparky[1], scottlee[*]}@hanyang.ac.kr

***Abstract*—Recently, due to the emergence of the concept of metaverse, software development that provides various interactions based on 3D virtual space is increasing explosively. Accordingly, the need for 3D software testing research is increasing as well. In order to sufficiently test the interaction between various objects in 3D space, exploratory testing using experts can be effectively applied. However, it is difficult to conduct sufficient testing in the field as the cost of human resources and effort is very high. In order to solve this problem, studies have been conducted to apply the exploratory characteristics of reinforcement learning agents to 3D software testing such as games. In this paper, we propose an exploratory testing automation method in which reinforcement learning agent effectively imitate the expert's testing behaviors using imitation learning. The experimental results show that the agent applied with imitation learning shows better performance than the existing curiosity-based reinforcement learning agent in terms of defect detection and cumulative reward, and can be effectively used for exploratory testing.**

***Index Terms*—software testing, reinforcement learning, imitation learning**

I. Introduction

With the advent of the metaverse concept, the software industry based on 3D virtual space, such as Roblox, Sandbox, and Horizon Worlds, has exploded where it also increased the need for 3D software testing research. Complex interactions between various objects in 3D space are very inefficient to be covered with general specification-based tests or structure-based tests. Hence, exploratory testing approach based on experience using the expertise can be very useful effectively applied. Exploratory testing is defined as simultaneous learning, test design, and test execution where it is being used in the field due to the difficulty of designing test cases for complex functions and the need for testing from the end user's point of view [1]. Therefore, exploratory testing approach can be effectively applied for 3D software testing. However, it requires a test engineer with sufficient domain knowledge and experience, which leads to an increase in the cost of human resources and effort. Therefore, testing is not performed properly in many 3D software.

Recently, with the development of reinforcement learning, research on agents that play computer games have been actively conducted where some studies using agents for play testing are being attempted. C. Gordillo [2] showed that using curiosity as an intrinsic reward can possibly improve play testing where applying agents have improved their exploratory abilities to a 3D game environment. Curiosity is defined as the prediction error that occurs when constantly predicting the state when a specific action is taken in the current state with the features of the state. Therefore, the more unfamiliar the exploration, the higher the curiosity reward. Hence, using it as an intrinsic reward can increase the exploration ability. However, to apply such to exploratory testing, there is a problem having difficulties to reflect the intention of the tester to the agent. In this paper, we propose an efficient exploratory testing approach that uses imitation learning to incorporate the tester's intentions into the agent's explorations and at the same time shortens the learning time.

II. Agents-Driven Exploratory Testing for 3D Software

Exploratory testing for 3D software using an agent is a testing method in which a human tester trains and runs the agent while testing. There are two main purposes of 3D software testing intended in this paper. The first is the play testing to ensure that the user experience is delivered as required, and the second is the defect testing to find and track fatal defects or unintended interactions that may occur between the user and the 3D space.

For play testing, the agent must be trained to play like a human. Many reinforcement learning studies use the cumulative reward of Atari games as a performance benchmark for algorithms. Therefore, play testing of 3D software can utilize the cumulative reward as a benchmark similar to existing studies on RL. Defect testing requires the ability to test various interactions. Hence, the exploratory ability of reinforcement learning agents is important. Therefore, it is important to use a search algorithm that overcomes the exploit-exploration dilemma well in order not to degrade the performance of reinforcement learning.

In addition, the testing intent of the human tester should be reflected in the agent. If agents with various testing tendencies are used, the effect of multiple testers performing exploratory testing can be obtained.

III. Implementation

Our approach not only uses curiosity as an intrinsic reward to motivate the agent to move, but also encourages them to

imitate the tester's demonstration by adding a reward granted by Generative Adversarial Imitation Learning (GAIL) [3]. GAIL trains a discriminator implemented as a neural network to learn to distinguish whether a particular observation or action originates from an agent or a demonstration. Then, whenever the agent collects observations, it compares them to the observations and behaviors of the demo and rewards them according to their similarity. As the discriminator becomes increasingly better at discriminating, the agent becomes better at imitating the demo as it deceives the discriminator.

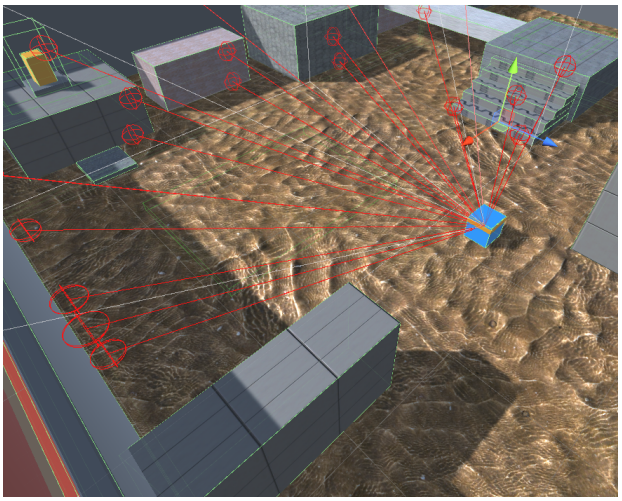


Fig. 1: Experimental Environment built in Unity

A. Environment

We constructed a simple game environment of size 100m x 100m x 15m using the Unity game engine as shown in the Fig. 1. Objects such as elevators, high terrains, bridges, and stairs are placed on the map, and players can move forward and backward, turn left and right, and jump to find randomly generated cube to get reward by clicking randomly spawned switch. We created boundaries that wrap the map for defect detection, and saved the movement paths within the episode in case the player collides with the boundary so that it can be visualized.

B. RL Algorithms

The RL agent to be used in the experiment is implemented using ML-Agents. Ml-Agents is an AI toolkit for the Unity engine that supports pytorch, tensorflow, and more as backends [4]. Proximal Policy Optimization (PPO) [5], which shows good performance in state space based on continuous observations such as position information and velocity in 3D space, is used as the basic RL Agent algorithm. Curiosity and GAIL are used together as intrinsic reward signals. The external compensation signal is set to $-1/max\_steps$ for every step and $+2$ when acquiring a cube, while the weight is set to 1. The weights of curiosity and GAIL were 0.2 and 0.1, respectively.

The observations consist of 27 ray perception sensors that fire a ray capable of detecting an object's tag, the switch's on/off state ($\mathbb{B}$), the agent's is_grounded state ($\mathbb{B}$) and velocity ($\mathbb{R}^3$). The actions of the agent are composed of keyboard movement which consists of (nothing, forward, backward, turn left and right) as well as jump actions including nothing and jump.

IV. Results

In this section, we present the cumulative reward and defect detection results of our agent and the PPO agent using only Curiosity. We also describe the agent's behaviors when using the typical user play demo and the tester play demo to find defects.

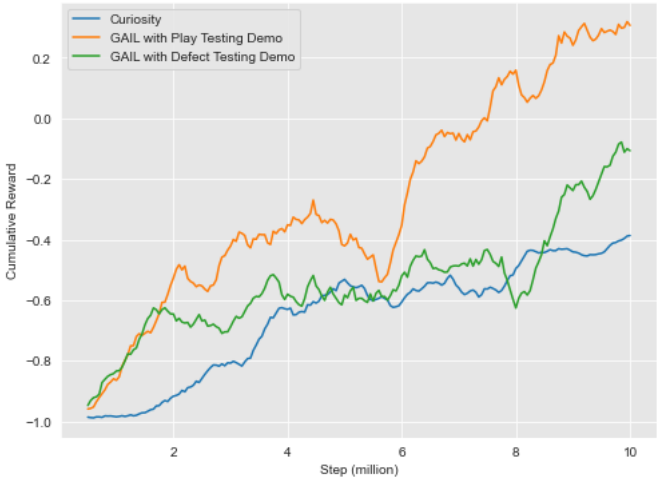


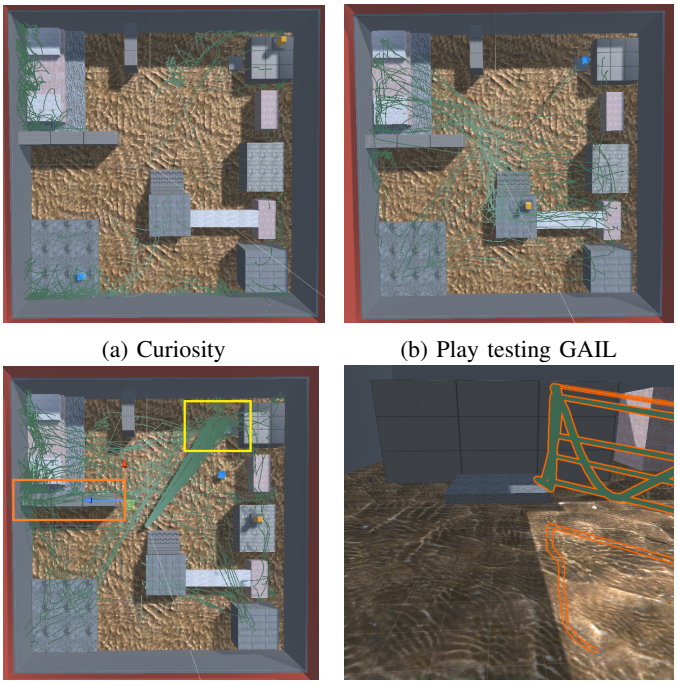
Fig. 2: Cumulative reward per step by curiosity, play testing GAIL, defect testing GAIL

A. Cumulative Reward

The Fig. 2 shows a cumulative reward graph when learning 10M steps of three agents which are curiosity PPO agent, GAIL agent imitating play testing demo, and GAIL imitating defect testing demo. The demos consist of 17 episodes each. In the play testing demo, play was recorded as a normal user, and whereas in the defect testing demo, the play was recorded to find defects mainly near elevators and high terrain that were intended to have defects. Looking at the graph, it is obvious that the agents using GAIL increase the accumulation reward faster than the agent using only curiosity. The agent using the play testing demo learned most effectively, and effectively imitated the user's play method as illustrated in the graph. The agent using the defect testing demo made better use of the elevator, which was not the case with the previous two agents, and increased the frequency of jumping to high terrain, but got a lower cumulative reward than the play testing demo agent. However, even in this case, it learned more effectively than the curiosity-based agent.

B. Defect Detection

The curiosity agent detected defects which are mainly occurring at the edge of the map, as shown in the Fig. 3a. The



(a) Curiosity (b) Play testing GAIL

(c) Defect testing GAIL (d) Elevator Defect

Fig. 3: Detected defect paths (green line) and critical elevator defect. In 3c, Orange box is high terrain and yellow is elevator.

agent did not tend to move over high terrain where there is a risk of going out of the map, and the detected defect coverage was lower than that of GAIL agents. The play testing GAIL agent detected defect episodes throughout the map. The Fig. 3c shows that the GAIL agent imitating the defect testing demo has detected many defects on the high terrain and in elevator.

During training, the curiosity agent had many duplicate defect episodes. However, the defect testing demo GAIL agent had fewer duplicates, and the play testing demo agent had the least duplicate defect episodes. With such results it can be confirmed that GAIL is much more efficient in finding the optimal policy.

C. Inference Testing

Testers can directly check the agent's play by inference the trained agents, where it can be used as a test case to correct the defects found. During the inference testing, We also discovered unintended interactions where the agent jumped between walls and climbed terrain that could only be reached by an elevator. Defect episodes that could not be detected (saved) during the learning with the setting of the detection boundaries can also be detected effectively by the tester in exploratory testing with the inference of the agent.

V. Conclusions

In this paper, we applied imitation learning to reinforcement learning agents to achieve effective exploratory testing automation of 3D software which consists of complex interactions. The agent using the GAIL-based imitation learning algorithm on the existing Curiosity-based agent was able to learn sample efficiently by receiving the tester's demo as input and was able to find more complex defect episodes. In addition, it was possible to train agents with different testing propensities by mimicking various test demos. Agents used for testing by inference after learning can be a great help for testers to perform exploratory testing. The proposed testing method is expected to significantly reduce the resource and effort cost in exploratory testing for 3D software.

As a future plan, we plan to introduce the Multi-agent RL algorithm so that the proposed exploratory testing technique can be applied to 3D software where interactions between multiple players occur.

References

[1] J. Itkonen and K. Rautiainen, "Exploratory testing: a multiple case study," in *2005 International Symposium on Empirical Software Engineering, 2005.*, 2005, pp. 10 pp.–.

[2] C. Gordillo, J. Bergdahl, K. Tollmar, and L. Gisslén, "Improving playtesting coverage via curiosity driven reinforcement learning agents," *CoRR*, vol. abs/2103.13798, 2021. [Online]. Available: https://arxiv.org/abs/2103.13798

[3] J. Ho and S. Ermon, "Generative adversarial imitation learning," *CoRR*, vol. abs/1606.03476, 2016. [Online]. Available: http://arxiv.org/abs/1606.03476

[4] A. Juliani, V. Berges, E. Vckay, Y. Gao, H. Henry, M. Mattar, and D. Lange, "Unity: A general platform for intelligent agents," *CoRR*, vol. abs/1809.02627, 2018. [Online]. Available: http://arxiv.org/abs/1809.02627

[5] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *CoRR*, vol. abs/1707.06347, 2017. [Online]. Available: http://arxiv.org/abs/1707.06347