Metaverse, Virtual Environments and Game Design, Vol. 178, 2025, 26–37
https://doi.org/10.54941/ahfe1006342

Enhancing Free Walking in Virtual Environments with Warning Walls: A Pilot Study on Redirected Walking Using Machine Learning Agents

I Nyoman Natanael[1,2], Chien-Hsu Chen[1], and Wei-Ting Lu[1]

[1]Department of Industrial Design, National Cheng Kung University, Tainan, 70101, Taiwan
[2]Bachelor Program in Visual Communication Design, Universitas Kristen Maranatha, Bandung, 40164, Indonesia

ABSTRACT

Virtual Reality (VR) technology allows users to explore unlimited virtual environment spaces. Users can explore virtual spaces using a Head Mounted Display (HMD) device. Various techniques for exploring VR spaces, such as teleportation, flying, walking-in-place, and devices such as omnidirectional treadmills, are often used. Previous literature states that real walking is the most natural method to explore virtual spaces. However, the constraint to the natural walking method is the limited physical space, so the user is at risk of encountering the boundary wall of physical space. Redirected Walking (RDW) technique addressed solving the limitations of tracking space so that users can explore unlimited virtual spaces without encountering the boundaries of physical space. The RDW technique has experienced rapid development since it was first introduced more than two decades ago; until now, its development has utilized artificial intelligence technology. This pilot study aims to explore the use of AI in the Deep Reinforcement Learning framework in designing virtual environment designs through a simulation system. This study uses machine learning agents trained to explore a non-predefined pathway virtual space larger than the tracking space by applying the essential principles of redirected walking techniques such as rotation, translation, and curvature gain. The study results show that machine learning agents can learn well and explore virtual spaces larger than the size of tracking spaces, both using warning walls and without warning walls.

Keywords: Deep reinforcement learning, Machine learning agents, Redirected walking, Virtual environment

INTRODUCTION

Virtual Reality (VR) technology allows us to explore unlimited Virtual Environments (VE). Users can explore it using a head-mounted display (HMD) device. Various exploration techniques such as teleportation, flying, walking in place (Lee et al., 2018), and omnidirectional treadmill devices are often used. Real walking is considered the most natural technique for exploring virtual spaces (Nogalski & Fohl, 2017; Usoh et al., 1999).

However, the constraint to the natural walking technique is the limited physical space, so the user is at risk of encountering the boundary wall of physical space. Redirected Walking (RDW) technique addressed solving the limitations of tracking space so that users can explore unlimited virtual spaces without encountering the boundaries of physical space.

VE exploration can use waypoints and predefined paths, which provide information about predicting the user's future path to the programmers, as proposed by Chen et al. (2024). The information can be effectively exploited using predictive algorithms to change the VE based on the user's position and movement within the tracking space. On the other hand, reactive algorithms can operate without presenting predefined paths, thus providing a general solution for RDWs (Strauss et al., 2020). When the user has reached the edge of the tracking space area when he intends to go to the location in the virtual space, a reset technique is needed to prevent the user from leaving the tracking space area or hitting a wall in the physical space (Strauss et al., 2020). Various reset techniques have been developed, such as Freeze-Backup, Freeze-Turn, and 2:1-Turn (Williams et al., 2007), the use of distractors (Peck et al., 2009) and interactive portals (Freitag et al., 2014) that aim to reset the user's orientation in the tracking space area.

This study implements Artificial Intelligence (AI) technology, especially in the Reinforcement Learning (RL) framework, using machine learning agents on VE without predefined pathways. The study aims to improve the exploration of larger VE than PE through machine learning agent simulations. Specifically, the study investigated the use of Warning Walls when agents were trained to explore a VE larger than the tracking space area. Warning Walls is a system automatically generated in VE when the agent is detected too close to the boundary of the tracking space area to prevent the agent from hitting the boundary of the physical wall. This feature is commonly used in VR, such as Chaperone SteamVR (*Steam Support*, n.d.), Oculus Guardian System (*Guardian System | Meta Horizon OS Developers*, n.d.), and Cirio et al. (2009) proposed the Magic Barrier Tape to inform users about the workspace boundaries. Warning Walls appear as a marker of the reset spot, and the agent must act in the tracking space area to continue walking in the VE. The null hypothesis in this study is that there is no difference between using Warning Walls when the machine learning agent learns to explore the VE space. The alternative hypothesis of this study is that there is a difference between using Warning Walls and without Warning Walls in the learning process of agents to explore VE.

RELATED WORK

The Redirected Walking (RDW) technique introduced by Razzaque allows users to explore by real walking in a larger virtual space than the limited size of the physical space (Razzaque et al., 2001). The experience of real walking feels more natural and results in a higher sense of presence when in a limited tracking space area (Strauss et al., 2020). RDW is generally applied using three main gains: translation, rotation, and curvature gains. These three gains function to manipulate translation and user rotation for VE. Gains can be defined as how tracked real-world movements are mapped

to the virtual environment (Steinicke et al., 2008). When gains are applied within certain thresholds, the discrepancy between movement in the real and virtual worlds is imperceptible (Steinicke et al., 2010). In addition to these three essential gains, there are other gains, namely bending gains. Bending gain is the same as curvature gain, which combines translation and rotation but is applied to the virtual curved path (Langbehn et al., 2017). This study does not use bending gain because VE is an open space without a predefined pathway. Predictive RDW algorithms can be used when the user's path in the tracking space is known in advance (Nescher et al., 2014; Zmuda et al., 2013). Reactive heuristic-based algorithms such as Steer-to-Center (S2C), Steer-to-Orbit (S2O), and Steer-to-multiple-targets were developed to keep the user in the tracking space (Razzaque, 2005). The S2C algorithm always steers the user to the center of the tracking space area, while S2O directs the user to walk in a circle. However, heuristic-based algorithms are designed and rely on human intuition regarding the best way to steer the users in VE. However, the Reinforcement Learning (RL) approach offers a data-driven alternative (Strauss et al., 2020).

Previous literature shows the development of various RDW methods under non-predefined pathway conditions in VE by utilizing the RL framework. Developments such as the Steer-to-Optimal-Target (S2OT) algorithm proposed by Lee et al. (2019), using Deep Q-Learning (DQN), which aims to estimate targets that steer optimally based on the Q value to avoid collisions in the path of future users. A study on RDW conducted by Strauss et al. (2020) proposed the Steer-by-Reinforcement Learning (SRL) algorithm. Using RL, they trained a neural network that directly determined manipulation parameters such as translation, rotation, and curvature gain applied based on the user's position and orientation in the tracking space area. The development of an RL-based RDW controller was also conducted by Zhao et al. (2023), which used translation, rotation, and curvature gain parameters through real-time information analysis of the tracking space area where the user is located. Zhao et al. (2023) used the Turn-to-Furthest (T2F) reset algorithm proposed by Chang et al. (2021) as an effort to redirect the user to the collision area furthest from the user's last position at the nearest obstacles or the boundary of the Physical Environment (PE). The 2:1-Turn reset technique (Williams et al., 2007) is the most commonly used (Zhao et al., 2023). This technique directs the user to rotate 180° in real space while rotating 360° in VE, thus maintaining coherence when walking in VE (Zhao et al., 2023). In this study, we did not explicitly use any reset algorithms.

METHOD

Virtual Environment

The environment used in this simulation consists of two parts. The first part represents the tracking space area or PE with a square shape. From the top view, it has a red outline measuring 4×4 meters. There are Warning Walls on each side of the tracking space boundary, which have a width of 0.5 meters. The agent can still walk inside the Warning Walls area, but if the agent continues to walk past its outer boundary, the training episode will

reset. The second part in the environment represents VE and is 10×10 meters in size and square, as seen in Figure 1 (d).

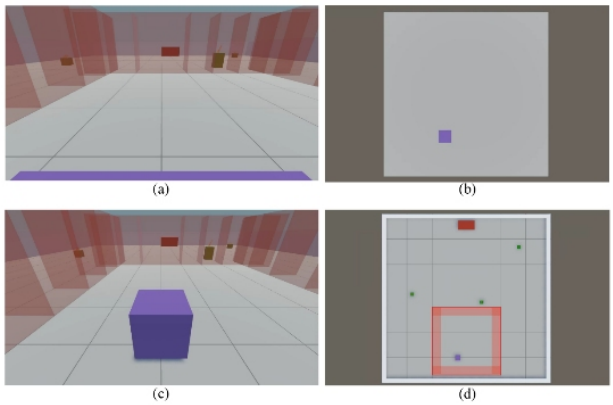

Figure 1: Snapshot of the simulation of the training agent in the environment. (a) From the agent's point of view, the first-person view shows that the red Warning Walls are active. (b) The top view display represents a tracking space area of 4x4 meters. (c) The third-person view shows the back of the agent. (d) The VE top view display overlapped the tracking space area with Warning Walls.

Target Objects

Four target objects are placed in the VE, consisting of one object placed in the North in a static position opposite the agent's position when the start is started and the other three objects placed randomly and dynamically. The position of the other three target objects will be randomly respawned when successfully touched by the agent. Random placement aims to encourage agents to explore all areas of VE and, at the same time, try to avoid overfitting conditions in agents. Overfitting conditions can occur when the agent shows optimal performance and over-masters the environment during training but fails to adapt to new variations or conditions in the environment (Zhang et al., 2018).

Machine Learning Agents

We used Unity's machine learning agent (ML-Agents) toolkit version 3.0 with Python Application Programming Interface (API) version 3.9.11 and was trained using the Proximal Policy Optimization (PPO) algorithm. The training system was built using a Personal Computer (PC) with Intel(R) i7-10700 specifications and a 2.90 GHz CPU with 16GB of RAM. The Unity ML-Agents Toolkit is an open-source project that allows researchers and developers to create simulations of environments using the Unity Editor and interact through Python API (Juliani et al., 2020). PPO is an RL algorithm Schulman et al. (2017) introduced to improve agent capabilities in complex environments. PPO focuses on optimization to maximize expected rewards based on the ratio of old and new policies. The agent (purple cube) represents

the user; it is placed in the tracking space area but can only see the VE. Agents have ray-cast sensors that allow for observations within the VE, such as tracking the position of target objects. Like those used by Strauss et al. (2020) and Zhao et al. (2023), the state agent at t time is influenced by four vectors $s_t = [x_t, z_t, \varphi_t{}^H, \varphi_t{}^L]$, where x_t and z_t are the coordinates of the agent in the tracking space area and $\varphi_t{}^H$ and $\varphi_t{}^L$ are the agent's heading angle and looking angle. We also increased the vector stacking value of the agent to 3 in the Unity Editor. This means agents can observe longer changes in a more complex and dynamic environment. After the agent observes the environment, the agent will act. The action performed by the agent is represented by three vectors $a_t = [g_t{}^R, g_t{}^T, g_t{}^C]$, where $g_t{}^R$, $g_t{}^T$, and $g_t{}^C$ are the rotation, translation, and curvature gains set during time t. We used the threshold value of each gain based on Steinickie et al. (2010), namely, $g_t{}^R \in [-0.2, 0.49]$, $g_t{}^T \in [-0.14, 0.26]$ and $g_t{}^C \in [-0.045, 0.045]$.

In RL, agents learn by receiving rewards from the environment as feedback on actions taken. In general, rewards can be divided into dense and sparse rewards. Dense rewards mean agents receive frequent and more detailed feedback after an action. In contrast, sparse rewards mean agents receive less feedback only after completing a particular task. Dense rewards will make it easier for the agent to learn because he directly receives feedback, while sparse rewards tend to cause slower training due to the lack of direct feedback. In this study, we used dense rewards. The Agent aims to touch the target object (red beam) after successfully touching as many randomized target objects (green cubes) as possible. Table 1 shows that we set the system's value as rewards that function as a reward or punishment in the agent training process.

Table 1: Rewards value system.

Action	Reward
Touching unvisited targets	10 f
Reaching the goal after all targets have been visited	50 f
Reach the goal before all targets are visited	–10f
Touching the physical wall in PE and VE	–1f
Touching the wall in VE	–1f
Touching the warning wall in VE	–0.5f
Penalty per step	–0.005f

The value of each reward in actions is determined based on previous trials and observations. We monitored the agent's behavior whenever we applied a value to specific actions. Until we see the agent exhibit behavior that we consider to be as expected. For example, with this value, the agent actively searches for object targets and explores every part of the VE space.

This study used discrete actions on agents rather than continuous actions. Discrete actions are a more straightforward set of actions; agents only need to choose the direction of navigation, such as forward, backward, right turn, and left turn. Agents do not require more complex actions such as adjusting acceleration, velocity, maintaining balance, and others, so discrete actions are more likely to match the agent's condition.

Training Simulation

The training simulation was conducted in two stages. In the first stage of training, twelve iterations were carried out, during which the hyperparameter value was adjusted little by little to obtain the best reward value, which would later be used as a baseline for the next stage. The training session uses Warning Walls as the default condition, assuming that Warning Walls are an important component in the system, such as a warning if the user hits the wall or goes out of the tracking space area. Then, in the second stage, training was continued for ten iterations on conditions A (with Warning Walls) and B (without Warning Walls) based on the previous baseline. Generally, the training configuration agent values used during the first training process are the Number of Layers: 2, Number of Hidden Units: 128, Maximum Steps: 500,000 steps, Time Scale: 1, and activating Normalize Network and Normalize Reward. Normalization can ensure that the agent learning process becomes more stable and effective because it can curate problems related to different scale values (*Reinforcement Learning Reward Normalization | Restackio*, n.d.). Other training configuration value settings can be seen in Table 2.

Table 2: Hyperparameter values in the agent's training configuration.

Agent Name	Hyperparameter Values				
	Batch Size	Buffer Size	Learning Rate	Beta	Epsilon
MLagent_DisTunA1	128	10240	0.0003	0.005	0.2
MLagent_DisTunA2	512	20480	0.0003	0.005	0.2
MLagent_DisTunA3	512	40960	0.0003	0.005	0.2
MLagent_DisTunA4	32	10240	0.0003	0.01	0.2
MLagent_DisTunA5	64	10240	0.0003	0.01	0.2
MLagent_DisTunA6	128	10240	0.0003	0.01	0.2
MLagent_DisTunA7	32	20480	0.0003	0.01	0.2
MLagent_DisTunA8	64	20480	0.0003	0.01	0.2
MLagent_DisTunA9	512	10240	0.0003	0.01	0.2
MLagent_DisTunA10	64	10240	0.0001	0.01	0.2
MLagent_DisTunA11	64	10240	0.0003	0.01	0.3
MLagent_DisTunA12	64	10240	0.0003	0.01	0.1

Batch Size is the number of experiences in each iteration of gradient descent on the PPO trainer type, Buffer Size is the number of experiences to collect before updating the policy model, and Learning Rate is the Initial learning rate for gradient descent. Corresponds to the strength of each gradient descent update step. Beta regulates how much the agent depends on their knowledge compared to exploring new possibilities. Epsilon can be interpreted as an agent's level of curiosity, determining how often he tries new things compared to continuing to do a known strategy (*Training Configuration File - Unity ML-Agents Toolkit*, n.d.).

Long Short-Term Memory (LSTM) is a neural network that handles sequential data and maintains temporal context (Hochreiter & Schmidhuber, 1997). LSTM can remember information over a long period, making it more effective in processing and predicting data with sequence dependencies such as text, voice, and time series. In this study, we also tested using LSTM based on baseline iteration using varying values on Memory Size and Sequence Length, see Table 3.

Table 3: Hyperparameter value for agent's memory.

Agent	Memory Size	Memory Sequence Length
Iteration-1	32	64
Iteration-2	64	64
Iteration-3	64	32
Iteration-4	128	32
Iteration-5	256	128

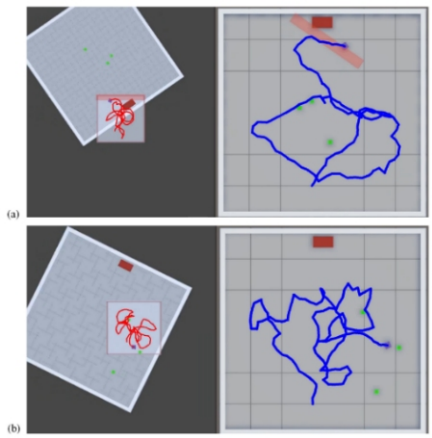

Figure 2: The training simulation snapshot shows the agent's walking trail lines in the tracking space area (red line) and the trail lines in the VE area (blue line), where the agent can walk in the VE, which is larger than the tracking space area without a predefined pathway. (a) The training process in Condition A where the Warning Walls (represented by a transparent red rectangle) will be activated when the agent is in a position close to the limit of the tracking space area. (b) Warning Walls are not activated even though the agent is within the boundary of the tracking space.

Furthermore, ten training iterations are conducted in two conditions in the second stage. In Condition A, Warning Walls will be automatically activated on the VE if the agent is in a margin area of 0.5 meters from each side of the tracking space boundary. In Condition B, all Warning Walls are disabled, so they do not appear even if the agent is in a margin area of 0.5 meters in the tracking space.

RESULT

The results of the first stage of training simulation obtained from TensorBoard showed that the number of cumulative rewards in the fifth (MLagent_DisTunA5) training session was the highest compared to eleven other training sessions with a total cumulative reward value of 291,103 or a mean reward value of 29,110. Thus, agent parameter five was chosen as the baseline for further training testing in Conditions A and B, see Figure 3 and Table 4.

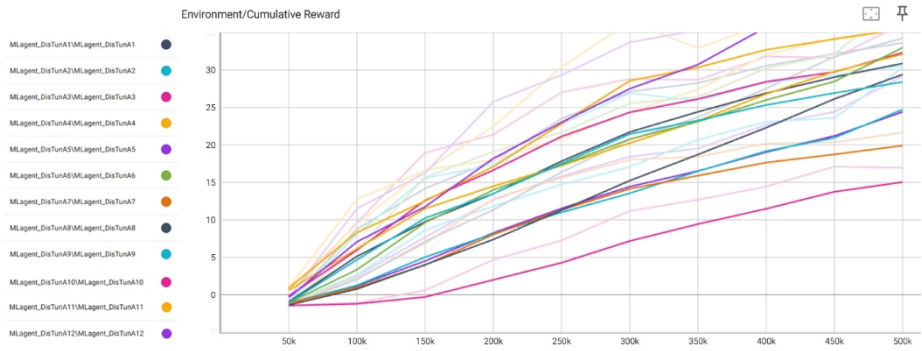

Figure 3: The results of the training in the first stage on TensorBoard.

Table 4: The results of mean reward value for the first stage of training agents.

	Steps										
Agent/ Reward Value	50k	100k	150k	200k	250k	300k	350k	400k	450k	500k	Mean Reward Value
1	-0.872	8.836	14.240	17.909	23.519	27.116	28.274	30.556	32.178	33.608	**21.536**
2	-1.024	2.741	8.690	11.870	14.763	17.107	20.720	23.115	23.604	30.324	**15.191**
3	-1.367	-0.993	0.666	4.737	7.307	11.237	12.721	14.481	17.137	16.987	**8.291**
4	0.688	9.503	16.703	18.017	20.872	24.399	27.442	32.057	34.201	35.535	**21.942**
5	-0.281	11.548	16.299	25.787	29.326	33.697	35.308	42.981	45.477	50.960	**29.110**
6	-0.972	6.043	15.730	19.125	21.693	25.554	26.437	30.217	32.110	39.640	**21.558**
7	-1.041	2.288	6.924	12.854	15.656	18.067	18.448	20.218	20.376	21.669	**13.546**
8	-1.288	2.119	7.190	11.384	16.415	20.722	23.724	27.527	31.796	34.226	**17.381**
9	-0.969	8.155	15.616	17.378	22.769	26.917	25.897	28.315	29.301	30.586	**20.396**
10	-0.100	9.690	18.990	21.398	27.028	28.794	28.692	31.794	31.649	36.143	**23.408**
11	0.988	12.796	16.577	22.498	30.367	36.440	32.922	36.109	36.219	37.485	**26.240**
12	-1.008	2.608	7.756	12.716	15.847	18.471	19.542	22.713	24.431	29.158	**15.223**

Based on the baseline iteration, the results in five iterations of the test showed that the Mean Reward Value collected by the agent with LSTM (5a-5e) was lower than those of training agents without LSTM or baseline. Therefore, in this study, we did not use memory on the agent for the next testing stage, see Table 5.

Table 5: The results of agent training with LSTM.

Agent/ Reward Value	Steps										
	50k	100k	150k	200k	250k	300k	350k	400k	450k	500k	Mean Reward Value
5a	-1.269	-0.166	4.527	11.517	13.388	17.614	16.583	19.726	19.782	20.655	**12.236**
5b	-0.001	7.104	13.731	14.489	19.418	22.295	24.900	22.606	24.938	24.079	**17.356**
5c	-0.689	9.060	11.952	16.137	15.921	18.181	22.026	23.123	24.474	23.297	**16.348**
5d	-0.899	9.522	13.878	17.274	18.492	21.426	25.796	27.267	26.335	31.901	**19.099**
5e	-0.892	10.465	12.893	15.701	17.996	18.918	22.463	25.807	25.259	33.430	**18.204**

Before statistical analysis, we collected data from 10 training iterations from the Tensorboard dashboard: Walltime and Mean Reward Value at every 50,000 steps to a maximum of 500,000 steps (10 summary frequency). Walltime data is obtained in Unix time format; for example, 1733722346, converted into a human-readable time format, means 12/9/2024 5:32:26 AM UTC. We converted all Walltime data into a human-readable format, then adjusted to the location of the time zone in which the study was conducted (UTC + 8). After that, we calculate the delta time in minutes, the difference in the time of recording the summary frequency. Then, we add up all the delta times to get the total training time and convert it into units of hours in each iteration. Then, we also calculate the mean reward value of each iteration, see Table 6. This is important to ensure that the data is the same as displayed on the Tensorboard dashboard.

Table 6: Calculation data delta time and mean reward Condition A and B of the second stage.

Iteration	Delta Time Hour		Mean Reward Value	
	Condition A	Condition B	Condition A	Condition B
1	4.407	5.852	24.974	21.645
2	3.967	4.386	24.473	23.518
3	3.918	4.536	22.386	22.733
4	3.894	4.552	20.097	21.610
5	4.052	4.167	21.417	26.787
6	4.239	4.009	24.293	25.694
7	4.219	4.086	17.820	21.139
8	3.981	3.996	18.500	22.341
9	4.203	4.155	26.689	20.288
10	4.253	4.053	21.998	27.146

Next, we perform descriptive statistical analysis calculations. Start by conducting a normality test on the DeltaTime variable and Mean Reward Value in Conditions A and B to determine whether the data distribution is normal. The results of the DeltaTime variable normality test in Condition A based on Kolmogorov-Smirnov (K-S) showed a value of $p = 0.200$ ($p > 0.05$) and Shapiro-Wilk (S-W) showed a value of $p = 0.362$ ($p > 0.05$). Meanwhile, the results of the normality test on Condition B based on K-S showed a value of $p = 0.027$ ($p < 0.05$), and S-W showed a value of $p = 0.001$ ($p < 0.05$). Thus, the DeltaTime data is normally distributed in Condition A

but not normally distributed in Condition B. The results of the normality test of the Mean Reward Value variable on Condition A based on K-S showed a value of $p = 0.200$ ($p > 0.05$), and S-W showed a value of $p = 0.837$ ($p > 0.05$). The results of the normality test of the Mean Reward Value variable for Condition B based on K-S showed a value of $p = 0.200$ ($p > 0.05$), and S-W showed a value of $p = 0.201$ ($p > 0.05$). Thus, the Mean Reward Value data is normally distributed in both Condition A and B.

Based on the normality test results, the statistical analysis was continued by conducting a non-parametric test on the DeltaTime variable and a parametric test on the Mean Reward Value variable. The results obtained in the non-parametric test using Mann-Whitney U show Asymp. Sig. (2-tailed): $p = 0.199$ ($p > 0.005$). Thus, the results state no significant difference between Condition A and Condition B for the DeltaTime variable.

The results of the parametric test on the Mean Reward Value variable for Condition A and Condition B using the Independent Samples T-Test showed that the homogeneous data (Levene's Test) with a value of $p = 0.639$ ($p > 0.05$) and a value of Sig. (2-tailed): $p = 0.402$ ($p > 0.05$). Thus, the results show no significant difference between the Mean Reward Value in Condition A and Condition B.

DISCUSSION

The results of the statistical analysis test in this study showed no significant difference between the time variables and rewards for Condition A (using Warning Walls) and Condition B (without Warning Walls). These results also follow visual observations made during the training process. The agent can walk to target objects that have randomly placed positions within the VE. Although agents still hit the warning walls quite often or get out of the tracking space area, they can also learn to walk in the VE, which is larger than the tracking space area. The agent learned how to perform a reset like 2:1-Turn without using an explicit algorithm to continue walking in the VE. Through this study, we obtained a fascinating finding: Machine learning agents can learn under conditions without visualizing Warning Walls in VE. We believe that exploring VE without warning walls is impossible for humans because humans rely on their vision to determine physical areas' boundaries in virtual spaces.

CONCLUSION

This pilot study aims to improve exploration in VE spaces that are larger in size compared to PE through machine learning agent simulations by using redirected walking techniques. The VE space is conditioned as an open space without a predefined pathway so agents can walk freely. The simulation was conducted by adding Warning Walls, which act as a warning system to prevent agents from hitting the boundaries of the tracking space area. The study results show that machine learning agents can learn to explore two VE conditions with and without Warning Walls. There is no significant difference in usage; thus, the null hypothesis in this study is acceptable.

REFERENCES

Chang, Y., Matsumoto, K., Narumi, T., Tanikawa, T., & Hirose, M. (2021). Redirection Controller Using Reinforcement Learning. *IEEE Access*, *9*, 145083–145097. https://doi.org/10.1109/ACCESS.2021.3118056

Chen, C.-H., Hsieh, H. Y., Liu, C.-Y., Li, W., Wu, Y.-P., & Lin, Y.-C. (2024). Infinite Museum: Implementation and Evaluation of Redirected Experience. *2024 IEEE 13th Global Conference on Consumer Electronics (GCCE)*, 928–931. https://doi.org/10.1109/GCCE62371.2024.10760805

Cirio, G., Marchal, M., Regia-Corte, T., & Lécuyer, A. (2009). The magic barrier tape: A novel metaphor for infinite navigation in virtual worlds with a restricted walking workspace. *Proceedings of the 16th ACM Symposium on Virtual Reality Software and Technology*, 155–162. https://doi.org/10.1145/1643928.1643965

Freitag, S., Rausch, D., & Kuhlen, T. (2014). Reorientation in virtual environments using interactive portals. *2014 IEEE Symposium on 3D User Interfaces (3DUI)*, 119–122. https://doi.org/10.1109/3DUI.2014.6798852

Guardian System | Meta Horizon OS Developers. (n.d.). Meta for Developers. Retrieved January 14, 2025, from https://developers.meta.com/horizon/documentation/native/pc/dg-guardian-system/.

Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, *9*(8), 1735–1780. https://doi.org/10.1162/neco.1997.9.8.1735

Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., & Lange, D. (2020). *Unity: A General Platform for Intelligent Agents* (No. arXiv:1809.02627). arXiv. https://doi.org/10.48550/arXiv.1809.02627

Langbehn, E., Lubos, P., Bruder, G., & Steinicke, F. (2017). Bending the Curve: Sensitivity to Bending of Curved Paths and Application in Room-Scale VR. *IEEE Transactions on Visualization and Computer Graphics*, *23*(4), 1389–1398. https://doi.org/10.1109/TVCG.2017.2657220

Lee, D.-Y., Cho, Y.-H., & Lee, I.-K. (2019). Real-time Optimal Planning for Redirected Walking Using Deep Q-Learning. *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*, 63–71. https://doi.org/10.1109/VR.2019.8798121

Lee, J., Ahn, S. C., & Hwang, J.-I. (2018). A Walking-in-Place Method for Virtual Reality Using Position and Orientation Tracking. *Sensors*, *18*(9), Article 9. https://doi.org/10.3390/s18092832

Nescher, T., Huang, Y.-Y., & Kunz, A. (2014). Planning redirection techniques for optimal free walking experience using model predictive control. *2014 IEEE Symposium on 3D User Interfaces (3DUI)*, 111–118. https://doi.org/10.1109/3DUI.2014.6798851

Nogalski, M., & Fohl, W. (2017). Curvature gains in redirected walking: A closer look. *2017 IEEE Virtual Reality (VR)*, 267–268. https://doi.org/10.1109/VR.2017.7892279

Peck, T. C., Fuchs, H., & Whitton, M. C. (2009). Evaluation of Reorientation Techniques and Distractors for Walking in Large Virtual Environments. *IEEE Transactions on Visualization and Computer Graphics*, *15*(3), 383–394. https://doi.org/10.1109/TVCG.2008.191

Razzaque, S. (2005). *Redirected Walking* [Ph. D., The University of North Carolina at Chapel Hill]. https://www.proquest.com/docview/305393643/abstract/B5B32B121D1A4478PQ/1

Razzaque, S., Kohn, Z., & Whitton, M. C. (2001). *Redirected Walking*. https://doi.org/10.2312/egs.20011036

Reinforcement Learning Reward Normalization | Restackio. (n.d.). Retrieved December 21, 2024, from https://www.restack.io/p/reinforcement-learning-answer-reward-normalization-cat-ai.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). *Proximal Policy Optimization Algorithms* (No. arXiv:1707.06347). arXiv. https://doi.org/10.48550/arXiv.1707.06347

Steam Support: SteamVR Chaperone FAQ. (n.d.). Retrieved January 14, 2025, from https://help.steampowered.com/en/faqs/view/30FC-2296-D4CD-58DA.

Steinicke, F., Bruder, G., Jerald, J., Frenz, H., & Lappe, M. (2010). Estimation of Detection Thresholds for Redirected Walking Techniques. *IEEE Transactions on Visualization and Computer Graphics*, *16*(1), 17–27. https://doi.org/10.1109/TVCG.2009.62

Steinicke, F., Bruder, G., Kohli, L., Jerald, J., & Hinrichs, K. (2008). Taxonomy and Implementation of Redirection Techniques for Ubiquitous Passive Haptic Feedback. *2008 International Conference on Cyberworlds*, 217–223. https://doi.org/10.1109/CW.2008.53

Strauss, R. R., Ramanujan, R., Becker, A., & Peck, T. C. (2020). A Steering Algorithm for Redirected Walking Using Reinforcement Learning. *IEEE Transactions on Visualization and Computer Graphics*, *26*(5), 1955–1963. https://doi.org/10.1109/TVCG.2020.2973060

Training Configuration File—Unity ML-Agents Toolkit. (n.d.). Retrieved December 21, 2024, from https://unity-technologies.github.io/ml-agents/Training-Configuration-File/.

Usoh, M., Arthur, K., Whitton, M. C., Bastos, R., Steed, A., Slater, M., & Brooks, F. P. (1999). Walking > walking-in-place > flying, in virtual environments. *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques - SIGGRAPH '99*, 359–364. https://doi.org/10.1145/311535.311589

Williams, B., Narasimham, G., Rump, B., McNamara, T. P., Carr, T. H., Rieser, J., & Bodenheimer, B. (2007). Exploring large virtual environments with an HMD when physical space is limited. *Proceedings of the 4th Symposium on Applied Perception in Graphics and Visualization*, 41–48. https://doi.org/10.1145/1272582.1272590

Zhang, C., Vinyals, O., Munos, R., & Bengio, S. (2018). *A Study on Overfitting in Deep Reinforcement Learning* (No. arXiv:1804.06893). arXiv. https://doi.org/10.48550/arXiv.1804.06893

Zhao, J., Zhu, W., & Zhu, Q. (2023). Redirection controller-based reinforcement learning for redirected walking. In F. Wen, C. Zhao, & Y. Chen (Eds.), *Fourth International Conference on Artificial Intelligence and Electromechanical Automation (AIEA 2023)* (p. 43). SPIE. https://doi.org/10.1117/12.2684589

Zmuda, M. A., Wonser, J. L., Bachmann, E. R., & Hodgson, E. (2013). Optimizing Constrained-Environment Redirected Walking Instructions Using Search Techniques. *IEEE Transactions on Visualization and Computer Graphics*. *19*(11), 1872–1884. https://doi.org/10.1109/TVCG.2013.88