iJRASET
International Journal For Research in Applied Science and Engineering Technology



INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** XI **Month of publication:** November 2026

DOI: https://doi.org/10.22214/ijraset.2026.85703

www.ijraset.com

Call: 08813907089 | E-mail ID: ijraset@gmail.com

International Journal for Research in Applied Science & Engineering Technology (IJRASET)
ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue XI Nov 2026- Available at www.ijraset.com

From Human Hands to Host Advantage? Demonstration-Seeded vs. Vanilla DQN in ViZDoom Deathmatches

Taufik Hidayat[1], Calvanny Lambue Rezananda[2], Louis Bratawidjaja[3]
[1, 3]*School of Artificial Intelligence, Nanjing University of Information Science and Technology*
[2]*School of Computer Science and Technology, Nanjing University of Information Science and Technology, Nanjing, China*

Abstract: ***Training autonomous first-person shooter (FPS) agents from raw pixels is sample-inefficient, and small-scale multiplayer evaluations are often confounded by network roles. This paper presents a controlled case study of a deep Q-network (DQN) agent whose replay buffer is seeded with human demonstrations in ViZDoom. Agent A receives 629 human transitions; Agent B is trained with vanilla DQN. Both agents are evaluated against a random-action baseline and against each other in a custom 1v1 deathmatch across five repeated batches, with host and joiner roles swapped. After recomputing outcomes from per-episode frag counts, Agent A achieved a 68.0% win rate against the random baseline (95% CI 54.2-79.2; McNemar p = 0.006), whereas Agent B achieved 52.0% (38.5-65.2; p = 0.22) and did not significantly outperform the baseline. Contrary to the original expectation, demonstration seeding did not accelerate convergence: the vanilla agent first reached positive test scores at epoch 3, versus epoch 5 for the seeded agent, and the final plateau was nearly identical (last-10 test means 72.6 vs. 72.7). In head-to-head play, hosts won 60.0% of duels versus 19.0% for joiners, with 21.0% ties (McNemar p < 0.001), but the effect was heterogeneous: the B-host condition showed a large advantage, while the A-host condition did not. The results highlight action-space encoding, per-episode logging, and counterbalanced host/joiner designs as critical for reproducible FPS benchmarks.***
Keywords: deep reinforcement learning; deep Q-network; demonstration-seeded replay; ViZDoom; first-person shooter AI.

I. INTRODUCTION

First-person shooter (FPS) games are a demanding testbed for autonomous agents. A policy must map high-dimensional visual input to aiming, movement, and tactical decisions under partial observability and adversarial pressure [1], [2], [3], [4]. Deep Q-networks (DQNs) can learn such policies directly from pixels [5], but they are sample-inefficient, and training from scratch in sparse-reward environments can be slow or unstable [6], [7]. A natural remedy is to initialize learning with human demonstrations, either through behavioral cloning (BC) [8], demonstration-augmented replay [9], or auxiliary demonstration losses [10].

Demonstrations are not a universal shortcut. BC suffers from distribution shift because the learned policy drifts away from the states visited by the demonstrator [6]. Recent work quantifies this shift [11] and mitigates it with robust stationary-distribution corrections [12]. Replay-only seeding—simply placing demonstration transitions in the replay buffer—is a much weaker intervention than methods that add demonstration losses or pretrain the policy [9], [10]. Whether replay-only seeding helps in a small, low-dimensional FPS scenario remains an empirical question, particularly when the demonstration action encoding does not cover the full action space.

Evaluation is a second challenge. In peer-to-peer 1v1 games, the host and joiner may experience different latency, hit registration, spawning, and input timing [13], [14], [15]. If host and joiner roles are not counterbalanced, an apparent difference between agents can reflect the network role rather than policy quality. Small-scale studies often report aggregate win rates without per-episode validation, confidence intervals, or role-swapped evaluations [2], [3], [4], [16], [17], [18], [19].

This paper presents a controlled case study in ViZDoom [1], [20] with two agents. Agent A is a DQN whose replay buffer is seeded with 629 human transitions. Agent B is a vanilla DQN trained with the same architecture and hyperparameters. Both are evaluated against a random-action baseline and against each other in a custom 1v1 deathmatch across five repeated batches, with host and joiner roles swapped. We ask three questions: (1) Does replay seeding with demonstrations improve learning speed or final performance? (2) How large is the host/joiner asymmetry, and is it consistent across evaluation batches? (3) Which data-quality practices are necessary to make such comparisons reproducible?

©IJRASET: All Rights are Reserved

The contributions are as follows. First, we provide a reproducible 1v1 ViZDoom pipeline with custom training and testing scenarios and host/joiner support. Second, we recompute all duel outcomes from per-episode frag counts and show that the original aggregate tables overcounted two conditions; the corrected counts are used throughout. Third, we show that replay seeding did not accelerate convergence or improve the final plateau in this run, despite a higher random-baseline win rate for Agent A. Fourth, we show that the host/joiner asymmetry is large in aggregate but heterogeneous across conditions and batches. Fifth, we distill practical recommendations for action-space encoding, per-episode logging, seeds, and counterbalanced host/joiner designs.

The corrected results are mixed. Agent A achieved a 68.0% win rate against the random baseline and significantly outperformed it; Agent B achieved 52.0% and did not. Demonstration seeding did not accelerate convergence: the vanilla agent first reached positive test scores at epoch 3, versus epoch 5 for the seeded agent, and the final plateau was nearly identical. In head-to-head play, hosts won 60.0% of duels versus 19.0% for joiners, with 21.0% ties, but the effect was driven by the B-host condition. These findings motivate a data-validation audit and a more cautious interpretation of small-scale FPS benchmarks.

II. RELATED WORK

A. *Value-Based Deep Reinforcement Learning*

Q-learning learns an action-value function by bootstrapping from the Bellman optimality equation [21]. DQN scaled this idea to high-dimensional pixels using a convolutional network, experience replay, and a target network [5]. Sutton and Barto provide the broader RL foundation [22]. Recent surveys document the rapid expansion of deep RL in games [23], [24]. Our study deliberately uses a compact DQN without recent architectural extensions. The goal is not to maximize benchmark performance but to isolate the effect of replay seeding in a controlled 1v1 setting; the simpler architecture also makes the training and evaluation pipeline easier to reproduce.

B. *Learning from Demonstrations*

Behavioral cloning treats demonstration state-action pairs as supervised labels [8]. It is simple and sample-efficient when demonstrations are plentiful, but it suffers from distribution shift: small errors move the policy into states that were not represented in the demonstration data [6]. Recent work quantifies this shift [11] and mitigates it with robust stationary-distribution corrections [12]. A recent ViZDoom study combines deep RL and imitation learning [25], and Imitational Actor-Critic (IAC) unifies behavioral cloning with actor-critic RL [16]. Demonstration-augmented RL combines the strengths of both paradigms. Deep Q-learning from demonstrations (DQfD) adds a demonstration loss and prioritizes demonstration transitions [9]. Nair et al. use demonstrations to overcome sparse-reward exploration and to seed the replay buffer [10]. The implementation in this paper is weaker than both: it only initializes the replay buffer with demonstration transitions and does not add a BC loss, a demonstration margin, or a pretraining phase. This distinction is important because a null result for replay-only seeding does not imply that demonstration learning is ineffective.

C. *FPS Agents and ViZDoom*

ViZDoom provides a lightweight, pixel-based FPS environment with customizable scenarios and multiplayer support [1]. Lample and Chaplot combined asynchronous actor-critic training with recurrent policies and reported superhuman performance on some deathmatch maps [2]. Recent studies train multiplayer FPS agents [3], evaluate RL algorithms in ViZDoom [4], benchmark ViZDoom scenarios [20], combine rule-based priors with deep RL [26], study human-like decision making [27], build modular FPS development frameworks [19], and use multi-modal observations [28].

More recent studies have explored imitation learning and benchmarking. Imitational Actor-Critic (IAC) combines behavioral cloning with actor-critic RL [16], and a ViZDoom study combines deep RL with imitation learning [25]. Solomon applied DQN in ViZDoom [17], Khan and Aqeel benchmarked RL algorithms [18], and Sun and Zhang introduced Gunner, a scalable deep RL system for Doom [29]. Most of these studies report aggregate performance against scripted or learned opponents. Few explicitly validate per-episode outcomes, report confidence intervals, or counterbalance host and joiner roles. Our case study addresses that gap, even though its scope is deliberately small.

D. *Broader Game AI and Evaluation*

Beyond FPS games, deep RL has achieved strong results in 1v1 fighting games with self-play curricula [30]. Recent surveys review multiagent RL in video games [24] and deep RL in game playing [23]. These systems rely on large-scale compute, league training, and carefully engineered curricula. Our study is several orders of magnitude smaller and should be read as a controlled case study rather than a competitive benchmark.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

Evaluation rigor is equally important. Recent work examines reproducibility in deep RL [7]. Network latency affects fairness and target selection in FPS games [13], [14], [15], which is directly relevant to host/joiner asymmetry. These observations motivate our per-episode validation, confidence intervals, paired tests, and batch-level plots.

III. METHODOLOGY

A. *Problem Formulation*

An RL task is a Markov decision process (MDP) defined by states, actions, transition dynamics, rewards, and a discount factor [22]. The agent follows a policy $\pi(a \mid s)$ and receives a return

$$G_t = \sum_{k=0}^{T-t} \gamma^k R_{t+k} \quad (1)$$

where T is the terminal step, γ is the discount factor, and R_t is the reward at step t. The goal is to learn a policy that maximizes the expected return. Q-learning estimates the action-value function $Q(s,a)$ and updates it toward the Bellman target [21]. DQN approximates Q with a convolutional network $Q(s,a;\theta)$ and minimizes

$$L(\theta) = \mathbb{E}[(y_t - Q(s_t,a_t;\theta))^2] \quad (2)$$

where $y_t = r_t + \gamma max_{a'} Q_{\text{target}}(s_{t+1},a')$. The gradient is

$$\nabla_\theta L(\theta) = \mathbb{E}[(y_t - Q(s_t,a_t;\theta))\nabla_\theta Q(s_t,a_t;\theta)] \quad (3)$$

In practice, a target network and experience replay stabilize training [5]. Each state is a 64x48 grayscale frame normalized to [0,1]. The action space contains eight discrete combinations of three binary buttons: MOVE_LEFT, MOVE_RIGHT, and ATTACK. The reward is the change in the game score: a kill yields 100 points, each step incurs a living reward of -1, and a rare event yields -6. Episodes end after 300 tics, which corresponds to about 25 agent decisions because actions are repeated for 12 tics.

B. *Environment and Scenarios*

Experiments use ViZDoom 1.3.0 with Freedoom assets [1]. The training scenario is a custom 448x448 arena with a stationary target opponent; the testing scenario is a custom 1v1 deathmatch map with host/joiner support. Both scenarios expose the same three buttons. The 1v1 matches use +timelimit 1, sv_spawnfarthest 1, sv_noitems 1, and switchplayermphealth 1 to reduce item and spawn variability. The evaluation scripts connect two ViZDoom instances over localhost: Player 1 hosts and Player 2 joins.

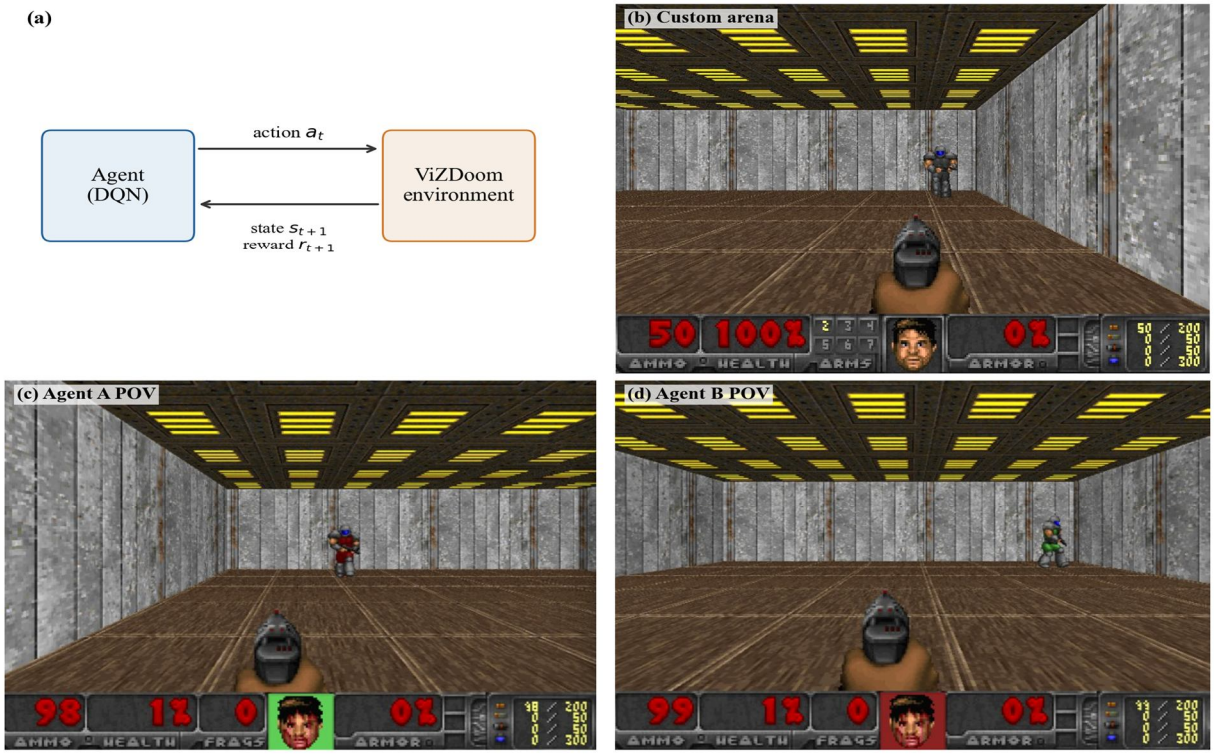


Fig. 1. ViZDoom environment and custom 1v1 arena. (a) Agent-environment interaction loop. (b) Screenshot of the custom arena. (c), (d) First-person observations from Agent A and Agent B during a duel.

The training and evaluation screenshots are color, but the agents receive grayscale observations. The environment uses a 640x480 screen resolution; each frame is resized to 64x48 and normalized. The custom maps were created with Ultimate Doom Builder. The training and testing maps are separate WAD files, and the testing map supports two simultaneous ViZDoom instances.

C. DQN Architecture

The network is a compact CNN. The input is a 1x48x64 grayscale frame. Conv1 uses 32 filters of size 4x4, stride 2, and padding 1, producing a 32x24x32 feature map. Conv2 uses 64 filters of size 4x4, stride 2, and padding 1, producing 16x12x64. The feature map is flattened to 12,288 values and passed to a fully connected layer with 512 units, followed by ReLU and dropout with probability 0.3. The output layer has eight units, one for each button combination. The network has 6,329,448 trainable parameters. Table II lists the layer details, and Figure 2 shows the architecture.

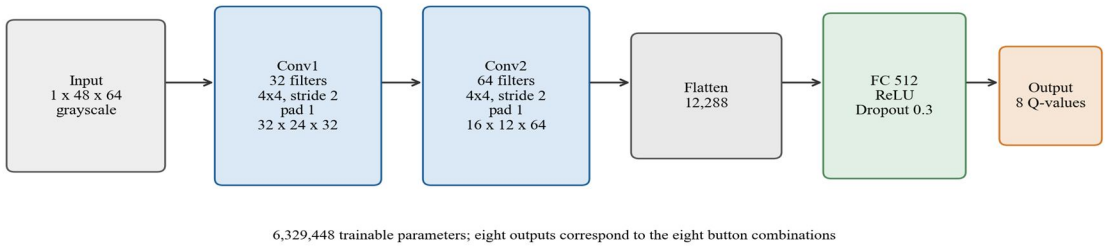


Fig. 2. DQN architecture. The network maps a 64x48 grayscale frame to eight Q-values, one for each button combination. Convolutional layers use 4x4 kernels, stride 2, and padding 1; the flattened feature size is 12,288.

Training uses a target network with the same architecture. The target is copied from the online network every epoch. The optimizer is RMSprop with a learning rate of 0.0001, and the loss is mean squared error. The replay buffer holds 100,000 transitions, and each update samples a minibatch of 64. Weights use Xavier initialization and biases start at zero. ε follows an exponential decay of 0.9996 per training step, from 1.0 to a minimum of 0.1; it reaches 0.1 by the third epoch in practice. Table I summarizes the hyperparameters.

TABLE I HYPERPARAMETERS AND TRAINING SETUP

Parameter	Value
Input	64 x 48 grayscale
Action space	8 combinations of 3 buttons
Learning rate	0.0001
Discount factor	0.99
Batch size	64
Replay memory	100,000
Training epochs	50
Steps per epoch	2,000
Test episodes per epoch	10
Epsilon start	1.0
Epsilon decay	0.9996 per step (exponential)
Epsilon minimum	0.1
Frame repeat	12
Episode timeout	300 tics
Optimizer	RMSprop

Parameter	Value
Loss	MSE
Target update	Every epoch
Dropout	0.3
Demo transitions	629
Training seed	Not fixed (single run)

TABLE II NETWORK TOPOLOGY

Layer	Kernel	Stride	Padding	Filters	Input	Output
Conv1	4 x 4	2	1	32	1 x 48 x 64	32 x 24 x 32
Conv2	4 x 4	2	1	64	32 x 24 x 32	16 x 12 x 64
Flatten	-	-	-	-	16 x 12 x 64	12,288
FC	-	-	-	-	12,288	512
Output	-	-	-	-	512	8

ReLU follows Conv1, Conv2, and FC; dropout 0.3 is applied before the output layer. Total trainable parameters: 6,329,448.

D. Demonstration-Seeded Replay

Human demonstrations were collected in spectator mode using the spectate.py script. The final buffer contains 629 transitions and 20 terminal flags. The action distribution is highly skewed: 489 transitions use action index 0, 71 use index 1, and 69 use index 2. Rewards are -1 for 607 transitions, 100 for 20 transitions, and -6 for 2 transitions. The metadata file reports 286 transitions and 10 episodes, but the actual buffer contains 629 transitions and 20 terminal flags; the paper reports the actual buffer and treats the metadata as stale. The raw predata.npz file contains 441 frames and three-element action vectors, which is a different representation from the final scalar-action buffer.

The key methodological point is that the demonstrations are used only to initialize the replay buffer. The training script does not compute a behavioral-cloning loss, does not pretrain the policy, and does not add a demonstration margin. Algorithm 1 summarizes the procedure. This is a weaker intervention than DQfD [9] or demonstration-augmented exploration [10], and it is further limited by the scalar action encoding: the demonstration actions are indices 0-2, while the training action space has eight combinations. The buffer therefore does not cover combined actions, and actions 0-2 are interpreted as combination indices 0-2. This limitation is important for interpreting the null result.

Algorithm 1 Demonstration-Seeded DQN Training

Input: demonstration buffer D, replay capacity M, batch size B, epochs E
1: Initialize $Q(s, a; \theta)$; $\theta_{\text{target}} \leftarrow \theta$
2: Load D into replay buffer R
3: for epoch = 1 to E do
4: for step = 1 to 2,000 do
5: Preprocess frame s; select a via ε-greedy policy
6: Execute a; observe r, s', done
7: Store $(s, a, r, s', \text{done})$ in R
8: Sample minibatch; compute $y = r + \gamma max_{a'} Q_{\text{target}}(s', a')$
9: Update θ by minimizing MSE; decay ε
10: end for
11: $\theta_{\text{target}} \leftarrow \theta$; test $\varepsilon = 0$
12: end for

E. Training and Evaluation Protocol

Each agent was trained for 50 epochs of 2,000 environment steps, for a total of 100,000 steps. The training script does not fix a random seed, and each agent was trained once. Training took approximately 9,320 s (2.6 h) for Agent A and 10,516 s (2.9 h) for Agent B on an NVIDIA GeForce RTX 5050 Laptop GPU, an Intel Core i5-13420H CPU, and 16 GB of RAM. The software stack is Python 3.13.13, PyTorch 2.12.0+cu132, ViZDoom 1.3.0, NumPy 2.4.4, pandas 3.0.3, Matplotlib 3.10.9, SciPy 1.17.1, and scikit-image 0.26.0 on Windows 11. Table III lists the implementation details and demonstration-buffer statistics.

TABLE III IMPLEMENTATION AND DEMONSTRATION DATA

Item	Value
GPU	NVIDIA GeForce RTX 5050 Laptop GPU
CPU	13th Gen Intel Core i5-13420H (8 cores, 12 threads)
RAM	16 GB
Operating system	Windows 11 (build 26200)
Python	3.13.13
PyTorch	2.12.0+cu132
ViZDoom	1.3.0
NumPy / pandas	2.4.4 / 3.0.3
Matplotlib / SciPy	3.10.9 / 1.17.1
scikit-image	0.26.0
Training time (A / B)	9,320 s / 10,516 s
Demonstration transitions	629
Terminal flags	20
Demonstration actions	0: 489, 1: 71, 2: 69
Demonstration rewards	-1: 607, 100: 20, -6: 2
Metadata (stale)	286 transitions, 10 episodes
Raw predata	441 frames, 3-element actions

The demonstration metadata does not match the actual buffer; the table reports the actual buffer. The raw predata file uses a different action representation.

Evaluation uses five repeated batches of ten episodes per condition, for 50 episodes per condition. The evaluation scripts fix the random seed at 42 and set ε to zero. The four conditions are Agent A versus random, Agent B versus random, Agent A hosting Agent B, and Agent B hosting Agent A. A win is a higher frag count at the end of the episode; a tie is an equal frag count. We report win rate, tie rate, decisive win rate (wins divided by wins plus losses), frags for and against, and K/D where the death logs are complete. We compute 95% Wilson confidence intervals for proportions and use exact McNemar tests on discordant pairs for paired outcomes, with Holm correction across the five primary comparisons. Because the five batches use the same seed rather than independent seeds, we treat them as repeated batches and show batch-level variability rather than assuming that all episodes are independent.

F. Data Validation

All duel outcomes were recomputed from per-episode frag counts. The original aggregate tables overcounted Agent B's wins against random by two (28/7/15 instead of 26/7/17) and Agent A's wins against Agent B by one (25/8/17 instead of 24/9/17). The 3test_v.csv batch stores cumulative deaths in the per-episode death column, so its K/D values are invalid; the B-versus-random K/D is computed from the four batches with complete death logs. The demonstration metadata is stale, as noted above. The corrected counts are used throughout, and the original counts are reported in the validation notes.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

IV. RESULTS

A. Training Dynamics

Figure 3 shows training and test mean reward over 50 epochs. Both agents learn rapidly. Agent B first reaches a positive test mean at epoch 3 (18.5), while Agent A does so at epoch 5 (67.4). Agent B reaches a test mean of at least 50 at epoch 4; Agent A does so at epoch 5. Agent A reaches a test mean of at least 70 at epoch 13, versus epoch 15 for Agent B. The best test mean is 78.8 for both agents, at epochs 13 and 38, respectively. The last-10-epoch test means are 72.6 for Agent A and 72.7 for Agent B. The last-10 training means are 32.3 and 34.0.

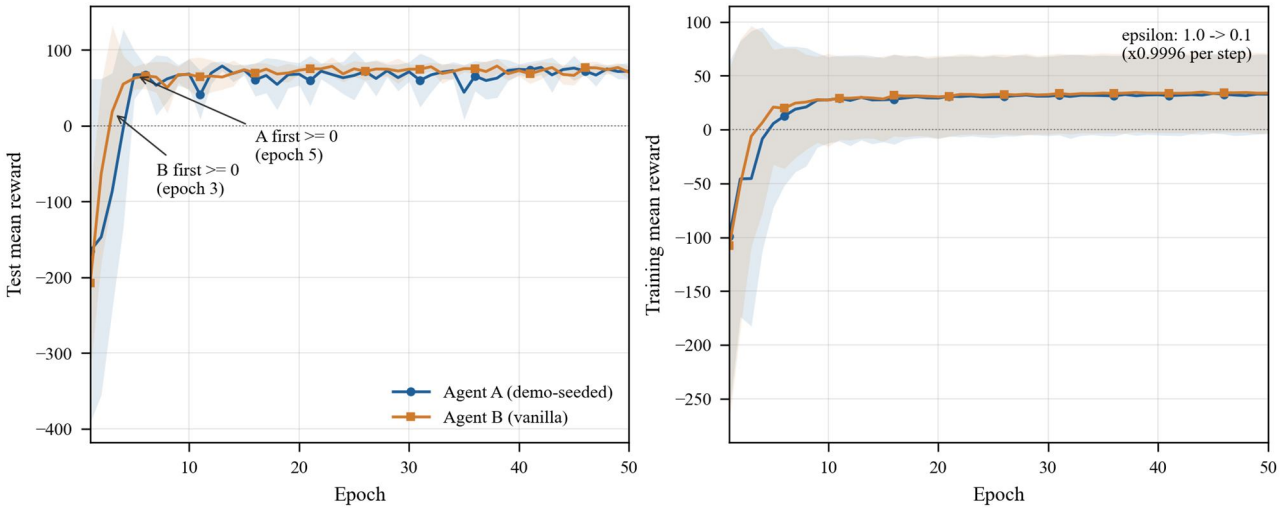


Fig. 3. Training and test mean reward over 50 epochs. Shaded bands show plus or minus one standard deviation over episodes within each epoch. The vanilla agent (B) first reaches a positive test mean at epoch 3; the demonstration-seeded agent (A) does so at epoch 5. Final plateaus are similar.

These results do not support a clear sample-efficiency advantage for demonstration seeding. Agent A has a slightly higher test score at epoch 5 (67.4 vs. 62.8), but Agent B reaches the early thresholds sooner. The final plateaus are nearly identical. The test mean is much higher than the training mean because training includes ε-greedy exploration and early episodes with large negative returns, whereas evaluation uses $\varepsilon = 0$. The two metrics should not be compared directly. The training curves are from a single unseeded run per agent, so the differences should be interpreted as descriptive rather than causal.

B. Agent Versus Random Baseline

Table IV and Figure 4 show the corrected outcomes against the random baseline. Agent A won 34 of 50 episodes, tied 2, and lost 14, for a 68.0% win rate (95% CI 54.2-79.2) and a 70.8% decisive win rate. The McNemar exact test on 34 discordant wins versus 14 losses is significant (p = 0.0055; Holm-adjusted p = 0.0165). Agent A's K/D is 1.37 (415 frags for, 304 against). Agent B won 26, tied 7, and lost 17, for a 52.0% win rate (38.5-65.2) and a 60.5% decisive win rate. The McNemar test on 26 wins versus 17 losses is not significant (p = 0.222; Holm-adjusted p = 0.444). The K/D for the four batches with complete death logs is 1.28 (284 frags for, 224 against).

TABLE IV CORRECTED RESULTS AGAINST THE RANDOM BASELINE

Agent	Role	W	T	L	Win rate, % (95% CI)	Decisive, %	Frags (for/against)	K/D
Agent A	Host	34	2	14	68.0 (54.2-79.2)	70.8	415 / 304	1.37
Random	Joiner	14	2	34	28.0 (17.5-41.7)	29.2	304 / 415	0.73
Agent B	Host	26	7	17	52.0 (38.5-65.2)	60.5	284 / 224	1.28*
Random	Joiner	17	7	26	34.0 (22.4-47.8)	39.5	224 / 284	0.78*

*K/D for B versus random is computed from the four batches with complete per-episode death logs; the fifth batch logged cumulative deaths and is excluded.

The contrast is important. Agent A clearly outperformed the random baseline; Agent B's advantage was modest and not statistically significant. The random bot is a weak baseline, so these results should not be interpreted as evidence of human-level competence. They do show that replay seeding was associated with a clearer advantage against random, even though the training curves did not show faster convergence.

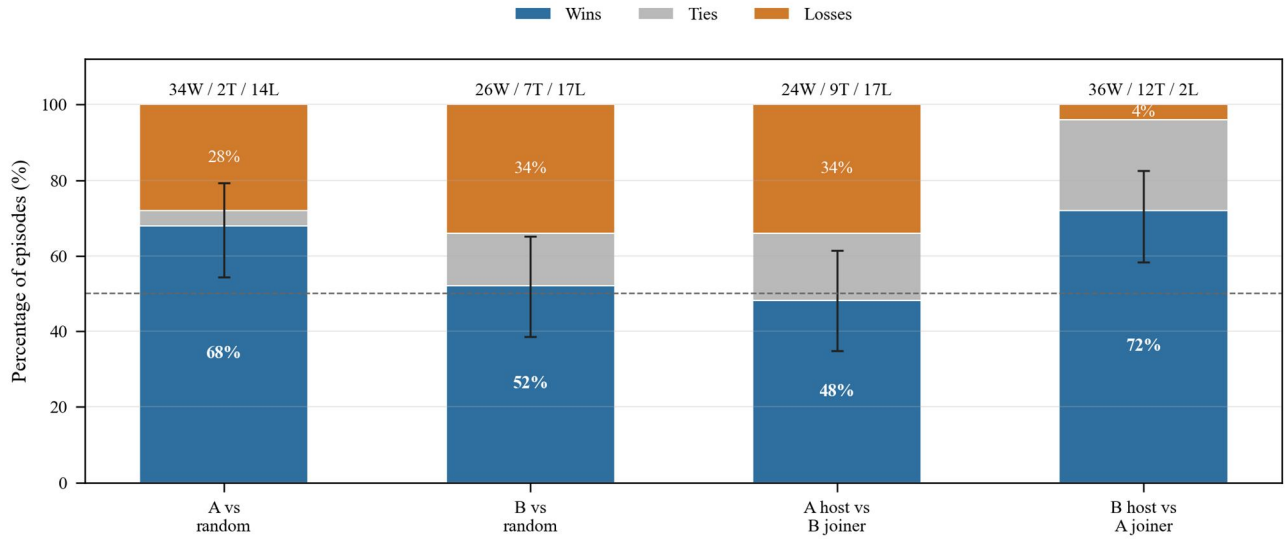


Fig. 4. Corrected win/tie/loss rates from per-episode frag counts. Error bars are 95% Wilson confidence intervals for win rates. Agent A significantly outperforms the random baseline; Agent B does not. The host advantage is large only in the B-host condition.

C. Head-to-Head Duels

Table V and Figure 4 show the head-to-head results. In Scheme 1, Agent A hosted Agent B. Agent A won 24 episodes, tied 9, and lost 17, for a 48.0% win rate (34.8-61.5); Agent B won 17, tied 9, and lost 24, for a 34.0% win rate (22.4-47.8). The McNemar test on 24 versus 17 discordant outcomes is not significant (p = 0.349). Agent A's K/D is 1.15, and Agent B's is 0.87. In Scheme 2, Agent B hosted Agent A. Agent B won 36 episodes, tied 12, and lost 2, for a 72.0% win rate (58.3-82.5); Agent A won 2, tied 12, and lost 36, for a 4.0% win rate (1.1-13.5). The McNemar test on 36 versus 2 discordant outcomes is significant (p = 5.4e-9; Holm-adjusted p = 2.2e-8). Agent B's K/D is 1.88, and Agent A's is 0.54; four deaths are unaccounted for, likely due to suicides or round resets.

TABLE V CORRECTED HEAD-TO-HEAD RESULTS

Scheme	Host	Joiner	Host W/T/L	Host win rate, % (95% CI)	Joiner win rate, % (95% CI)	Frags (host/joiner)	Host K/D	Joiner K/D
1	Agent A	Agent B	24 / 9 / 17	48.0 (34.8-61.5)	34.0 (22.4-47.8)	92 / 80	1.15	0.87
2	Agent B	Agent A	36 / 12 / 2	72.0 (58.3-82.5)	4.0 (1.1-13.5)	177 / 94	1.88	0.54*

*Four deaths are unaccounted for, likely due to suicides or round resets.

The head-to-head results are asymmetric. The A-host condition does not show a significant host advantage, while the B-host condition shows a large one. This interaction is the main empirical finding of the duel analysis.

D. Host/Joiner Asymmetry

Pooling both schemes, hosts won 60 of 100 duels, joiners won 19, and 21 were ties. The host win rate is 60.0% (50.2-69.1), the joiner win rate is 19.0% (12.5-27.8), and the tie rate is 21.0%. The McNemar exact test on 60 host wins versus 19 joiner wins is significant (p = 4.2e-6). The pooled decisive host rate is 75.9%.

The pooled effect masks substantial heterogeneity. Figure 5 shows host and joiner win rates for each batch. In the B-host batches, the host wins 70%, 80%, 100%, 90%, and 70% of episodes, while the joiner wins 0% in four of the five batches. In the A-host batches, the host wins 70%, 60%, 40%, 30%, and 40%, while the joiner wins 10%, 40%, 20%, 50%, and 50%.

In A-host batches 4 and 5, the joiner actually wins more episodes than the host. The host advantage is therefore not universal; it depends on the matchup and perhaps on the specific policies.

Several mechanisms could contribute to the asymmetry: latency and input timing [13], [14], [15], server-authoritative hit registration, spawn placement, map asymmetry, and differences in how the two policies react to the host/joiner roles. The current design cannot separate the host role from agent identity because Agent A only hosts in Scheme 1 and Agent B only hosts in Scheme 2. A fully counterbalanced 2x2 design, in which each agent hosts and joins, is needed to estimate a causal role effect. The high tie rate also complicates interpretation: ties may reflect defensive play, time limits, or evenly matched policies rather than equal skill.

E. Data Validation and Robustness

Recomputing outcomes from per-episode frag counts changed two of the four conditions. The original B-versus-random table reported 28/7/15 (56.0%), but the per-episode data give 26/7/17 (52.0%). The original A-host-versus-B table reported 25/8/17 (50.0%), but the per-episode data give 24/9/17 (48.0%). The B-host and A-versus-random counts are unchanged. The original K/D values for B versus random (0.63 and 0.39) are invalid because one batch stores cumulative deaths; the corrected K/D for the four clean batches is 1.28 and 0.78. These corrections change the interpretation: Agent B does not significantly outperform random, and the A-host condition does not show a significant host advantage.

TABLE VI PRIMARY STATISTICAL COMPARISONS

Comparison	W/T/L	Win rate, % (95% CI)	McNemar p	Holm-adjusted p
A vs random	34 / 2 / 14	68.0 (54.2-79.2)	0.0055	0.0165
B vs random	26 / 7 / 17	52.0 (38.5-65.2)	0.222	0.444
A host vs B joiner	24 / 9 / 17	48.0 (34.8-61.5)	0.349	0.444
B host vs A joiner	36 / 12 / 2	72.0 (58.3-82.5)	5.4e-9	2.2e-8
Pooled host vs joiner	60 / 21 / 19	60.0 (50.2-69.1)	4.2e-6	-

McNemar exact tests use discordant pairs; ties are excluded. Holm correction is applied across the five primary comparisons; the pooled test is shown for reference.

Table VI summarizes the primary tests. The A-versus-random and B-host-versus-A comparisons remain significant after Holm correction; the B-versus-random and A-host-versus-B comparisons do not. The pooled host effect is significant but should be interpreted alongside the batch-level heterogeneity in Figure 5.

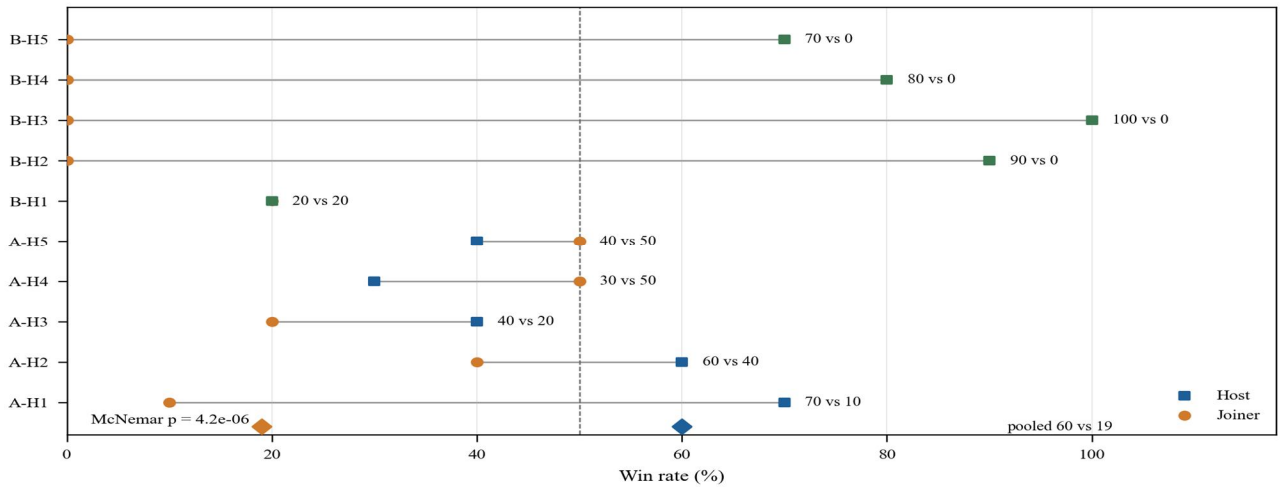


Fig. 5. Host and joiner win rates for each evaluation batch. Each row connects the joiner (circle) and host (square) win rate for one batch of ten episodes. The pooled host advantage is significant (McNemar $p < 0.001$), but the effect is heterogeneous: in A-host batches 4-5, the joiner wins more often.

V. DISCUSSION

A. *Why Demonstration Seeding Did Not Accelerate Learning*

The most surprising result is the absence of a clear sample-efficiency advantage for replay seeding. Agent B reached positive test scores and a test mean of 50 earlier than Agent A, and the final plateaus were nearly identical. Agent A had a slightly higher test mean at epoch 5 and reached a test mean of 70 two epochs earlier, but these differences are small relative to the variability across epochs.

Several factors may explain the null result. First, the implementation only seeds the replay buffer; it does not add a behavioral-cloning loss, a demonstration margin, or a pretraining phase. DQfD [9] and Nair et al. [10] use stronger demonstration signals, and behavioral cloning itself is vulnerable to distribution shift [6]; recent work quantifies and mitigates this shift [11], [12]. Second, the demonstration buffer is small (629 transitions) and covers only action indices 0-2, while the training action space has eight combinations. The buffer therefore provides limited coverage of combined actions. Third, the training scenario is simple: a stationary target in a small arena, with a dense living reward and a short episode. Exploration may be easy enough that demonstrations add little. Fourth, the training was run once per agent without a fixed seed, so the observed differences may be noise.

The higher random-baseline win rate of Agent A (68.0% vs. 52.0%) is consistent with a benefit from demonstration seeding, but it is not conclusive. The evaluation uses a different scenario and a different opponent, and the sample is small. A cleaner test would re-collect demonstrations with the full action vector, train multiple seeds, and evaluate on multiple maps.

B. *Interpreting the Host/Joiner Asymmetry*

The pooled host advantage is large (60.0% vs. 19.0%), but the batch-level and scheme-level results show that it is not a uniform law. The B-host condition is strongly asymmetric, while the A-host condition is not significant. This interaction suggests that the host role interacts with agent identity, policy style, or matchup. Possible mechanisms include latency and input timing [13], [14], [15], server-authoritative hit registration, spawn placement, and map asymmetry. The current design confounds host role with agent identity, so the pooled estimate should be read as a description of these two matchups rather than a causal effect of hosting.

The high tie rate (21.0%) also matters. Ties may reflect defensive play, time limits, or policies that avoid risk. They are not evidence of equal skill. A larger sample with explicit tie-breaking rules, latency logging, and counterbalanced roles would help separate these explanations.

C. *Limitations and Threats to Validity*

Internal validity: host role is confounded with agent identity; the five batches use the same evaluation seed rather than independent seeds; training is unseeded and single-run; no latency or spawn logs were collected; the demonstration action encoding does not cover the full action space; K/D logs are incomplete for one batch. External validity: the study uses one map, one scenario family, three buttons, one demonstrator, and 629 transitions; the random baseline is weak; no human baseline is included. Construct validity: win rate is a coarse measure of skill, ties are frequent, and K/D is sensitive to logging. Statistical validity: the sample is small, multiple comparisons are performed, and episode-level confidence intervals may be optimistic because episodes are clustered within batches. Data validity: the aggregate tables overcounted two conditions, the metadata is stale, and one batch stores cumulative deaths.

These limitations do not invalidate the case study, but they restrict its scope. The corrected episode-level analysis and the batch-level plots are intended to make the uncertainty visible rather than to hide it.

D. *Recommendations*

First, log the full action vector for every transition, not a scalar index. This is necessary for demonstration replay and for analyzing action coverage. Second, log per-episode wins, ties, losses, frags, deaths, and latency; do not rely on cumulative scoreboard columns. Third, fix and report training seeds, run multiple seeds, and report variability. Fourth, counterbalance host and joiner roles across agents and batches, and use dedicated servers or localhost with latency instrumentation. Fifth, report confidence intervals and batch-level results alongside pooled estimates. Sixth, if demonstration learning is the goal, use a method that adds a demonstration loss or pretraining, such as DQfD [9] or IAC [16], rather than replay seeding alone. Seventh, evaluate on multiple maps and opponents, including human baselines, before drawing general conclusions.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

VI. CONCLUSIONS AND FUTURE WORK

This paper presented a controlled case study of demonstration-seeded versus vanilla DQN in ViZDoom 1v1 deathmatches. The corrected episode-level results show that Agent A significantly outperformed the random baseline (68.0% win rate), while Agent B did not (52.0%). Contrary to the original expectation, replay seeding did not accelerate convergence: the vanilla agent reached early test thresholds sooner, and the final plateaus were nearly identical. In head-to-head play, hosts won 60.0% of duels versus 19.0% for joiners, but the effect was heterogeneous: the B-host condition showed a large advantage, while the A-host condition did not. The study also exposed data-quality issues—aggregate table overcounts, cumulative death logging, stale demonstration metadata, and an action-encoding mismatch—that are easy to miss in small-scale FPS benchmarks.

The results suggest three lessons. First, replay-only seeding is not equivalent to demonstration learning; stronger methods may be needed to realize the expected sample-efficiency gains. Second, host/joiner asymmetry can dominate 1v1 comparisons and must be counterbalanced. Third, per-episode logging and action-space validation are essential for reproducible results.

Future work will re-collect demonstrations with full three-button action vectors, train multiple seeds with DQfD or IAC, evaluate on multiple maps, and use a counterbalanced host/joiner design with latency logging. Extending the study to self-play and league training [24] and to recent modular FPS frameworks [3], [19] would provide a stronger basis for competitive FPS agents.

REFERENCES

[1] M. Kempka, M. Wydmuch, G. Runc, J. Toczek, and W. Jaskowski, "ViZDoom: A Doom-based AI research platform for visual reinforcement learning," in Proc. IEEE Conf. Comput. Intell. Games (CIG), 2016, pp. 1-8, doi: 10.1109/CIG.2016.7860433.

[2] G. Lample and D. S. Chaplot, "Playing FPS games with deep reinforcement learning," in Proc. AAAI Conf. Artif. Intell. (AAAI), 2017, pp. 4460-4466, doi: 10.1609/aaai.v31i1.10827.

[3] P. Almeida, V. Carvalho, and A. Simoes, "Reinforcement learning as an approach to train multiplayer first-person shooter game agents," Technologies, vol. 12, no. 3, p. 34, 2024, doi: 10.3390/technologies12030034.

[4] A. Khan and M. Naeem, "Evaluating reinforcement learning algorithms in first-person shooter games using VizDoom," Multimedia Tools and Applications, vol. 84, no. 15, pp. 15053-15075, 2024, doi: 10.1007/s11042-024-19658-y.

[5] V. Mnih et al., "Human-level control through deep reinforcement learning," Nature, vol. 518, no. 7540, pp. 529-533, 2015, doi: 10.1038/nature14236.

[6] S. Ross, G. J. Gordon, and J. A. Bagnell, "A reduction of imitation learning and structured prediction to no-regret online learning," in Proc. Int. Conf. Artif. Intell. Stat. (AISTATS), 2011, pp. 627-635.

[7] T.-A. Paleu and C. Pascal, "Reproducibility in deep reinforcement learning with maximum entropy," in Proc. 27th Int. Conf. Syst. Theory, Control Comput. (ICSTCC), 2023, pp. 428-433, doi: 10.1109/ICSTCC59206.2023.10308431.

[8] D. A. Pomerleau, "Efficient training of artificial neural networks for autonomous navigation," Neural Computation, vol. 3, no. 1, pp. 88-97, 1991, doi: 10.1162/neco.1991.3.1.88.

[9] T. Hester et al., "Deep Q-learning from demonstrations," in Proc. AAAI Conf. Artif. Intell. (AAAI), 2018, pp. 3223-3230, doi: 10.1609/aaai.v32i1.11757.

[10] A. Nair, B. McGrew, M. Andrychowicz, W. Zaremba, and P. Abbeel, "Overcoming exploration in reinforcement learning with demonstrations," in Proc. IEEE Int. Conf. Robot. Autom. (ICRA), 2018, pp. 6292-6299, doi: 10.1109/ICRA.2018.8463162.

[11] O. S. Mulekar, R. Bevilacqua, and H. Cho, "Metric to evaluate distribution shift from behavioral cloning for fuel-optimal landing policies," Acta Astronautica, vol. 203, pp. 421-428, 2023, doi: 10.1016/j.actaastro.2022.12.011.

[12] S. Seo, B.-J. Lee, J. Lee, H. Hwang, H. Yang, and K.-E. Kim, "Mitigating covariate shift in behavioral cloning via robust stationary distribution correction," in Proc. Adv. Neural Inf. Process. Syst. (NeurIPS), vol. 37, 2024, pp. 109177-109201, doi: 10.52202/079017-3466.

[13] S. S. Tokey et al., "The effects of network latency on the peeker's advantage in first-person shooter games," in Proc. 19th Int. Conf. Found. Digit. Games (FDG), 2024, pp. 1-10, doi: 10.1145/3649921.3650002.

[14] S. Liu and M. Claypool, "The impact of latency on target selection in first-person shooter games," in Proc. 14th ACM Multimedia Syst. Conf. (MMSys), 2023, pp. 51-61, doi: 10.1145/3587819.3590977.

[15] C. Michaelides and B. Bellalta, "Demystifying the peeker's advantage in multiplayer first-person shooter games," IEEE Transactions on Games, vol. 18, no. 1, pp. 197-206, 2026, doi: 10.1109/TG.2026.3656267.

[16] Y. Ying, "Mastering first-person shooter game with imitation learning," in Proc. Int. Conf. Netw., Commun. Inf. Technol. (NetCIT), 2022, pp. 587-591, doi: 10.1109/NetCIT57419.2022.00139.

[17] E. Solomon, "Reinforcement learning for autonomous agents via deep Q networks (DQN) within ViZDoom environment," in Proc. Int. Conf. Artif. Intell., Blockchain, Internet Things (AIBThings), 2024, pp. 1-5, doi: 10.1109/AIBThings63359.2024.10863438.

[18] A. Khan and A. Aqeel, "Benchmarking reinforcement learning algorithms in first-person shooter games using ViZDoom," Entertainment Computing, vol. 55, p. 101031, 2025, doi: 10.1016/j.entcom.2025.101031.

[19] S. Lee and H. Sung, "A modular reinforcement learning framework for iterative FPS agent development," Electronics, vol. 15, no. 3, p. 519, 2026, doi: 10.3390/electronics15030519.

[20] A. Khan, A. A. Shah, L. Khan, M. R. Faheem, M. Naeem, and H.-T. Chang, "Using VizDoom research platform scenarios for benchmarking reinforcement learning algorithms in first-person shooter games," IEEE Access, vol. 12, pp. 15105-15132, 2024, doi: 10.1109/ACCESS.2024.3358203.

[21] C. J. C. H. Watkins and P. Dayan, "Q-learning," Machine Learning, vol. 8, no. 3-4, pp. 279-292, 1992, doi: 10.1007/BF00992698.

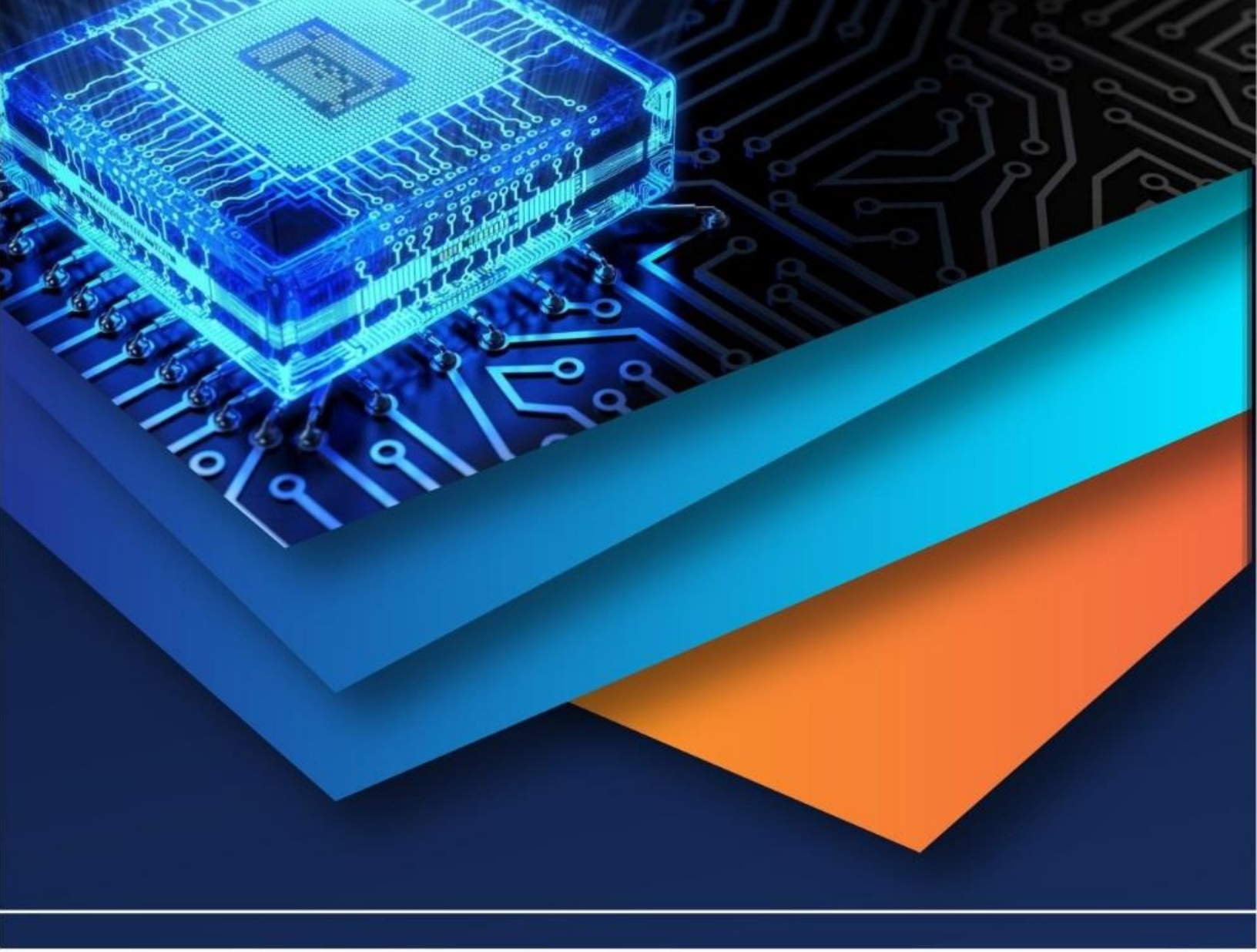
[22] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[23] S. Patankar, C. Usakoyal, P. Patil, and K. Raut, "A survey of deep reinforcement learning in game playing," in Proc. MIT Art, Design and Technology School of Computing Int. Conf. (MITADTSoCiCon), 2024, pp. 1-5, doi: 10.1109/MITADTSoCiCon60330.2024.10575819.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |

[24] Z. Li, Q. Ji, X. Ling, and Q. Liu, "A comprehensive review of multiagent reinforcement learning in video games," IEEE Transactions on Games, vol. 17, no. 4, pp. 873-892, 2025, doi: 10.1109/TG.2025.3588809.
[25] Y. Xu, "Deep reinforcement learning and imitation learning based on VizDoom," in Proc. 6th Int. Conf. Electron. Inf. Technol. Comput. Eng. (EITCE), 2022, pp. 1700-1706, doi: 10.1145/3573428.3573729.
[26] Z. Fang, J. Zhao, W. Zhou, and H. Li, "Implementing first-person shooter game AI in WILD-SCAV with rule-enhanced deep reinforcement learning," in Proc. IEEE Conf. Games (CoG), 2023, pp. 1-8, doi: 10.1109/CoG57401.2023.10333171.
[27] K. Futatsuike and T. Goto, "Game AI with human-like decision making using reinforcement learning in first-person shooter games," in Proc. 28th ACIS Int. Winter Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput. (SNPD-Winter), 2024, pp. 82-86, doi: 10.1109/SNPD-Winter67015.2024.00023.
[28] A. Sayapin and I. Makarov, "Multi-modal deep reinforcement learning in ViZDoom with audio component," in Proc. IEEE 20th Jubilee Int. Symp. Intell. Syst. Informat. (SISY), 2022, pp. 437-442, doi: 10.1109/SISY56759.2022.10036289.
[29] Y. Sun and L. Zhang, "Gunner: Playing FPS game Doom with scalable deep reinforcement learning," PeerJ Computer Science, vol. 11, p. e3410, 2025, doi: 10.7717/peerj-cs.3410.
[30] I. Oh, S. Rho, S. Moon, S. Son, H. Lee, and J. Chung, "Creating pro-level AI for a real-time fighting game using deep reinforcement learning," IEEE Transactions on Games, vol. 14, no. 2, pp. 212-220, 2022, doi: 10.1109/TG.2021.3049539.

©IJRASET: All Rights are Reserved | SJ Impact Factor 7.538 | ISRA Journal Impact Factor 7.894 |



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Call : 08813907089 (24*7 Support on Whatsapp)