Received 4 January 2024, accepted 19 January 2024, date of publication 24 January 2024, date of current version 1 February 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3358203

RESEARCH ARTICLE

Using VizDoom Research Platform Scenarios for Benchmarking Reinforcement Learning Algorithms in First-Person Shooter Games

ADIL KHAN[1], ASGHAR ALI SHAH[2], LAL KHAN[3], MUHAMMAD REHAN FAHEEM[4], MUHAMMAD NAEEM[1], AND HSIEN-TSUNG CHANG[5,6,7,8]

[1]Department of Computer Science, University of Peshawar, Peshawar, Khyber Pakhtunkhwa 25120, Pakistan
[2]Department of Computer Science, Bahria University, Islamabad 44000, Pakistan
[3]Department of Software Engineering, Foundation University Islamabad, Islamabad 44000, Pakistan
[4]Department of Computer Science, The Islamia University of Bahawalpur, Bahawalpur, Punjab 63100, Pakistan
[5]Artificial Intelligence Research Center, Chang Gung University, Taoyuan 333323, Taiwan
[6]Bachelor Program in Artificial Intelligence, Chang Gung University, Taoyuan 333323, Taiwan
[7]Department of Computer Science and Information Engineering, Chang Gung University, Taoyuan 333323, Taiwan
[8]Department of Physical Medicine and Rehabilitation, Chang Gung Memorial Hospital, Taoyuan 333323, Taiwan

Corresponding author: Hsien-Tsung Chang (smallpig@cgu.edu.tw)

This work was supported in part by the National Science and Technology Council under Grant 111-2221-E-182-058 and Grant 112-2410-H-182-026-MY2, and in part by the Chang Gung Memorial Hospital under Grant NERPD4N0231 and Grant NERPD2M0231.

ABSTRACT Advances in deep reinforcement learning have made it possible to create artificial intelligence-based agents for games that use visual information to make decisions as accurately as humans. Novel procedures are often evaluated in two-dimensional games. However, they are relatively easy compared to three-dimensional games, which have a significantly larger state and action space and, more prominently, contain partially observable states. Thus, this paper trains agents with different reinforcement learning algorithms that work fine in contradiction of human players and in-built agents by evaluating them in the first-person shooter (FPS) game Doom using the VizDoom platform. The agents learned in three different scenarios (maps): ' Defend the Center,' 'Deadly Corridor,' and 'Health gathering.' C51-DDQN, DFP, and REINFORCE algorithms have been proven effective in this study. To assess how well the trained agents performed using various reinforcement learning algorithms, we compared the results of our research with other findings in the literature. Finally, this paper presents a comparative analysis and future research directions.

INDEX TERMS Artificial intelligence, agents, game AI, reinforcement learning, VizDoom.

I. INTRODUCTION

The development of AI over the past decade has caused a revolution and abrupt change in all fields. Similarly, gaming is a well-known aspect of our cultural landscape and has existed since the time of our ancestors. Early computers were slow, and user interaction resulted in restrained basic ideas. As computers advanced in the early 1940s, programmers created novel virtual worlds and unexpected human-computer interaction methods. Later on, thanks to technological developments like GPUs [1] and TPUs [2] and the innovation in neural networks [3], It is possible to use artificial intelligence significantly in games [4]. Because of this, Atari games have used reinforcement learning algorithms and strategies to train agents. The agents played diverse arcade learning environment-based games with improved results and performances. Many ALE-based games occurred in 2D scenarios partially observable to the agents [5], [6]. Reinforcement learning algorithms try to learn how to carry out a specific, frequently challenging task. It might be about beating the opposition or finishing a level with the best score in computer games. The feedback received during the reward period is used to train algorithms. The agents are rewarded for correct decisions and punished for

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

FIGURE 1. Doom game GUI shows a first-person perspective.

mistakes. This way, the desired behaviour is reinforced [7], [8]. VizDoom is a popular research platform that allows academicians to form custom FPS maps for training and evaluating RL agents. Applying reinforcement learning (RL) algorithms to the VizDoom platform is an exciting area of research and development where artificial agents learn to play FPS games by means of RL techniques [9]. This paper uses reinforcement learning to explore training agents on the VizDoom testbed. At present, it is considered essential to use reinforcement learning algorithms in three-dimensional environments, for instance, Doom [10], an FPS game presented in FIGURE 1; Warcraft [11], [12], a third-person perspective RTS game, and sandbox open-world games like GTA-V [13] and Minecraft [14].

A. BENEFITS OF RL FOR SHOOTING 3D GAMES

1) Mastering complex tasks:

RL is best at learning complex strategies and manipulating dynamic environments, allowing AI agents in FPS games to master complex skills like cover usage, flanking, target prioritization, and weapon retreat control.

2) Adaptability and generalization:

RL algorithms can adapt to scenarios, opponent behaviours, and weapon settings without proper guidance, which makes them flexible and capable of handling diverse gameplays.

3) Emergent behaviour:

RL agents can sometimes surprise even their creators by discovering unexpected strategies and tactics, which leads to more exciting and creative AI opponents in FPS games.

4) Potential for competitive AI:

With ongoing research and development, RL-powered AI could outperform human players in FPS games, leading to even more challenging and exciting virtual opponents.

B. WEAKNESSES OF RL FOR SHOOTING 3D GAMES

1) Slow and Resource-intensive Training:

Learning through trial and error can be inefficient, requiring vast computational resources and training time, which limits the accessibility of RL for smaller researchers.

2) Sample inefficiency:

RL algorithms can be sample-inefficient, requiring much more data than supervised learning methods to achieve similar performance. This can be problematic for complex, high-dimensional tasks like 3D FPS games.

3) Sensitivity to hyperparameters:

Tuning the numerous hyperparameters involved in RL can be challenging and significantly impact the learning process and final performance.

4) Ethical concerns:

The emergence of autonomous AI agents that outclass at killing enemies raises concerns about the potential for glorification of violence in real-world applications.

RL provides exciting possibilities for developing adaptive and intelligent AI opponents in FPS games. However, it's essential to admit the challenges associated with training and ethical considerations around using such technology. The future of RL in FPS games probably rests on concentrating these challenges through improvements in efficient training techniques, hyperparameter optimization methods, and responsible practices. As these challenges are overcome, even more impressive AI-powered experiences in the future of FPS games can be expected.

As a result of the analysis of AI in games, it is assumed and considered that games are the finest testbeds to assess diverse RL algorithms before evaluating them in the practical world. Therefore, cutting-edge reinforcement learning algorithms, such as advantage actor-critic (A2C) [15], advantage actor-critic-long short-term memory (A2C-LSTM) [16], [17], double deep Q-network (DDQN) [18], deep recurrent Q-network (DRQN) [19], [20], categorical double deep Q-Network (C51-DDQN) [21], direct future prediction (DFP) [22], and REINFORCE [23], are experimented with in a three-dimensional environment VizDoom to train agents. It is recommended that such an analysis be produced so that the community studying first-person shooters could use it as a preliminary point for future study and advancements.

II. BACKGROUND

The application of Deep Q-Network (DQN) and Deep Recurrent Q-Learning Network (DRQN) in the Doom game is explored by [24], leveraging the advancements in Deep Visual Reinforcement Learning. The study draws inspiration from a publication by Lample and Chaplot in 2016 [25]. The paper introduces DQN and DRQN as two modifications of Deep Q-Learning, showcasing their application in playing Doom. It presents results from a simplified game scenario, highlighting their performance differences in predicting game features, particularly enemy positions. Moreover, a significant emphasis is placed on constructing an implementation framework for reinforcement learning algorithms, providing insights into how the testbed must be created to facilitate experimentation. One notable contribution of the proposed work is the presentation of results that demonstrate the effectiveness of the proposed architecture. Unlike previous studies, the paper achieved an impressive accuracy rate of nearly 72% in predicting enemy positions. The result emphasized the promising capabilities of DQN and DRQN in game AI, showcasing the potential for the DQN and DRQN algorithms to outperform human players when making decisions based solely on raw screen pixels.

The work proposed in [26] addresses the limitations of common built-in game agents, which often rely on pre-written scripts and potentially unfair information. Instead, the paper focuses on utilizing deep learning and reinforcement learning methods to create game agents that make decisions more flexibly, akin to human players who rely solely on the game screen. The primary approach involved implementing a game agent following the asynchronous advantage actor-critic (A3C) algorithm. The agent inputs real-time game screen data and produces discrete actions accordingly. It operates within the VizDoom environment and receives visual information to make decisions, controlling in-game characters. A significant contribution of the work is the enhancement of the A3C algorithm by integrating the 'anticipator network' into the original algorithm structure, which enables it to anticipate game outcomes prior to making decisions. It associates real-time visual information with anticipation images, providing thorough input to the A3C algorithm. Remarkably, the results proved that the A3C algorithm, with anticipation, beats the performance of the original A3C algorithm. This advancement implies that featuring anticipation mechanisms side with the agent's behaviour more closely with human players, concentrating the limitations of traditional scripted game agents.

The work proposed in [27] shows the significance of games as an influential research field for artificial intelligence. It points out that games are excellent testbeds for evaluating AI theories and concepts before evaluating them in real-world scenarios. The key driving factors behind advancements in AI within this field include grown computational potential, advancement in machine learning, in particular RL, and the expansion of neural networks. These enhancements assist agents in surpassing human performance by making decisions based solely on raw visual information. The work further focuses on applying RL, particularly deep Q-learning, to deal with agents in popular games, for example, Doom and Minecraft. It sketches building testbeds for cutting-edge algorithms by means of platforms like VizDoom, Gym-Minecraft, and Microsoft's Malmo. Moreover, the work presents some early results from the Doom game scenarios, such as predicting the positions of opponents and other related features. This work evaluates two RL variants, DQN and DRQN, in different decision processes and finally concludes that DQN outperforms DRQN in predicting the positions of opponents.

Additionally, the article extends its analysis to Minecraft scenarios, highlighting the superior performance of DRQN, especially in situations with partially observable decision processes (POMDP). It demonstrates that the planned architectures surpass built-in agents and human players in accurately precepting game features. In short, the proposed work emphasizes the growing capabilities of AI agents in mastering complex games through DRL and neural networks, making substantial contributions to AI research in the gaming domain. Some related value-based, policy-based, and actor-critic-based algorithms and the methods used in the literature are presented in Table 1.

Similarly, the work proposed by [28] addresses the limitations of built-in game agents, which typically rely on pre-written scripts and unfair information and lack the flexibility of human players who base their decisions solely on the game screen.

The main objective is to develop agents that emulate human decision-making by employing deep reinforcement learning and enhancing related algorithms. The article introduces a game agent, built using convolutional neural networks (CNN), that augments the A3C algorithm. The agent accepts visual information in real-time as input and produces the matching policies. It interacts with the VizDoom environment, reading the visual information to make decisions for managing the characters' actions. To improve the A3C algorithm's performance, an 'anticipator network' is added to the algorithm structure. This network allows the agent to generate anticipatory information before deciding. It groups real-time visual information with anticipation images to create a comprehensive input for the A3C algorithm, developing discrete actions. Significantly, this modification doesn't change the fundamental algorithm but enhances its performance. The article conducts simulations to match the implementation of the original A3C with the proposed variant, such as the A3C, with anticipation. The results indicated that the A3C algorithm with anticipation outperforms the standard A3C, demonstrating the effectiveness of the anticipator network in making game agents act more like human players. The comparisons are backed by experimental data from related studies, confirming the improvements achieved in the article.

A. DEEP REINFORCEMENT LEARNING

A deep neural network (DNN) uses multiple hidden layers [30], [31]. Each hidden layer creates a novel representation of the input from the preceding layer using a nonlinear transformation or activation function [30]. The Q-value is the expected reward when acting in state 's' of any optimal policy 'π.' Consequently, the value of Q is used to evaluate which action "a ϵ Actions" is superior to others. In this way, a better policy is worked out. This technique is called Q-learning [32]. The Q-learning approach allows the original policy 'π' to act together with its environment. The value of 'Q' can then be acquired in state 's' by acting 'a' as in (1). Methods like Temporal Difference (TD) or Monte Carlo (MC) help to identify a better policy 'π' to replace the previous actor. SARSA is close to Q-learning. Q-learning, on the other hand, is an off-policy algorithm, while SARSA is an on-policy algorithm. As a substitute for the maximum

TABLE 1. Representation of drl methods [17], [29].

	DRL Algorithms	Main Methods/Approach
Value-based	DQN	Experience Replay, Target Network, Clipping Rewards, and Skipping Frames
	Double DQN	Double Q-learning
	Dueling DQN	Dueling Neural Network architecture
	Prioritized DQN	Prioritized experience replay
	Bootstrapped DQN	Deep exploration with DNNs
	Distributed DQN	Distributional Bellman equation
	Noisy DQN	Parametric noise added to weights.
	Rainbow DQN	Combining six extensions to the DQN
	Hierarchical DQN	Hierarchical value functions
	Gorila	Asynchronous training for multi-agents
Policy-based	TRPO	KL divergence consultant
	PPO	Specialized clipping in the objective function
Actor-Critic	DEEP DPG	DNN and DPG
	TD3	Twin Delayed DDPG
	PGQ	Policy Gradient and Q-learning
	Soft Actor-Critic (SAC)	Maximum Entropy RL Framework
	A3C	Asynchronous Gradient Decent

reward for the next state, SARSA is considering a new action based on the same policy to update the Q-values as in (2).

$$Q(s,a) \leftarrow Q(s,a)+\alpha\,[r+\gamma max\alpha\ Q(s',a') - Q(s,a)](s \leftarrow s') \quad (1)$$

$$Q(s,a) \leftarrow Q(s,a)+\alpha\,[r+\gamma Q(s',a') - Q(s,a)](s \leftarrow s', a \leftarrow a') \quad (2)$$

Q-learning is used for making decisions in an environment, and its goal is to learn a policy, a mapping from states to actions, which raises the overall reward with time. Q-learning specifically focuses on learning a Q-function, denoted as Q (s, a), which stands for the expected overall reward of acting 'a' in the state 's' and then exploiting the best policy thereafter.

A step-by-step breakdown of the Q-learning update equation is as follows:

1. Initialize Q-values: Start with arbitrary Q-values for all state-action pairs. These values are typically stored in a table or a function approximator.
2. Select an action**:** In each state, choose an action based on a policy. This policy can be exploratory (exploring new actions) or greedy (exploiting the action with the highest Q-value).
3. Perform the action**:** Execute the selected action in the scenario.
4. Observe the reward and the next state**:** Receive the immediate reward from the scenario and see the new state.
5. Update the Q-value**:** Use the Q-learning update equation to adjust the Q-value for the chosen action in the present state. What each term of the equation means is presented in Fig. 2:

Where:

- Q (s, a) is the Q-value for making action 'a' in state 's'
- 'α' is the learning rate regulating the step size of updates.
- 'r' is the immediate reward gained after making action 'a' in state 's.'
- 'γ' is the discount factor, balancing the significance of immediate and future rewards.
- max$\acute{Q}$(s$'$, a$'$) represents the highest Q-value for any action in the subsequent state.

6. Repeat steps 2-5 till convergence or a predetermined number of iterations.

The Q-learning update equation is derived from the Bellman equation, which expresses the recursive connection between a state's value and its successor states' values. The update process iteratively refines the Q-values, converging towards a best policy that raises the expected overall reward in the given environment.

B. DEEP Q-NETWORK

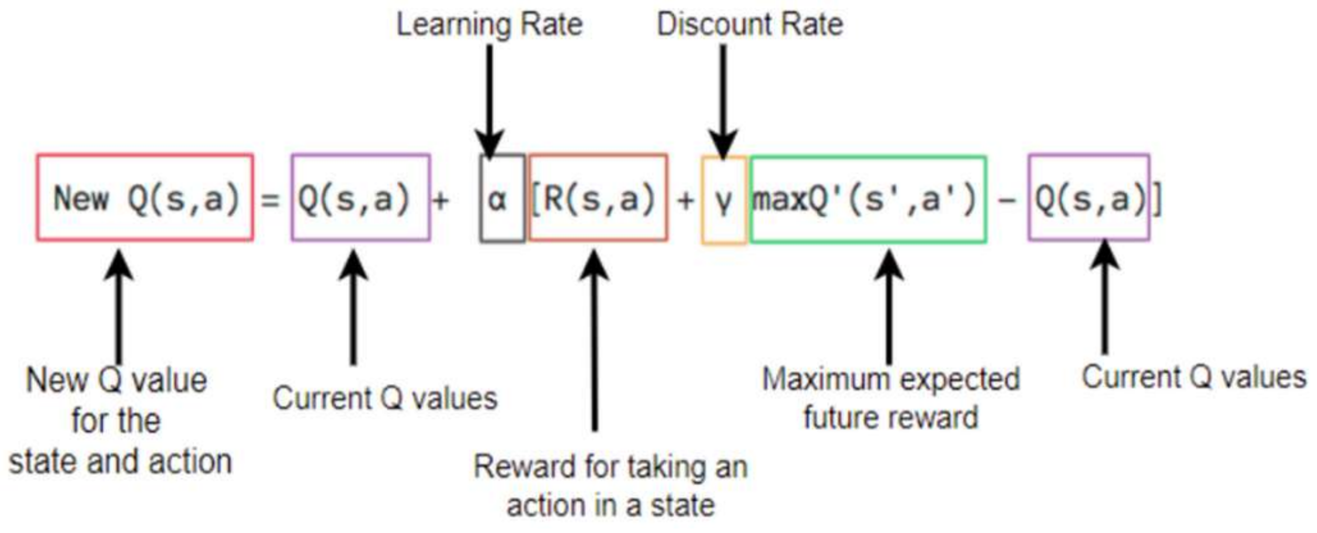
The authors [35] proposed an agent capable of superhuman performance in arcade learning games, beginning the age of Deep RL accomplishments. This network's input is an image from the game, and its output is a Q-function, one for each action. The projected model [36], [37] cannot use the rule directly and must instead calculate Temporal Difference error (TD) as in (3):

$$TD = Q(s_i, a_i) - -(r_i + \gamma max_{\alpha i}\ Q(s', a')) \quad (3)$$

Then, the authors employed a primary network to approximate the Q(s_i, a_i) term and a target network to calculate the maximum Q(s, a) term. The primary network is learned through backpropagation, whereas the value provided by the target network is set and updated regularly. The subsequent loss was used to prepare for online network (primary network) parameters and target network parameters as per (4):

$$L = \sum_i (Q(s_i, a_i; \theta) - (r_i + \gamma \max_{a'} Q\left(s_i', a'; \tilde{\theta}\right))^2 \quad (4)$$

The sum is measured across a set of samples over which the network propagates the mistake. The authors [38]



Learning Rate
Discount Rate
New Q(s,a) = Q(s,a) + α [R(s,a) + γ maxQ'(s',a') − Q(s,a)]
New Q value for the state and action
Current Q values
Reward for taking an action in a state
Maximum expected future reward
Current Q values

FIGURE 2. A term wise breakdown of the Q-learning update equation [33], [34]

proposed using experience replay memory to construct a batch comprising tuples (s, a, r, s) containing state, completed action, reward, and future state.

III. METHODOLOGY AND EXPERIMENTS

This part introduces the methodology used and presents the details of the experiments. The algorithms are compared, centering around the 3D shooter Doom. In Doom, players view the world from a first-person perspective and enjoy freedom of movement across three-dimensional surfaces. They can collect weapons and health kits and engage in combat with programmed enemies capable of performing various actions, including attacking the player. Doom's notable feature is its flexibility in custom scenarios, allowing for tasks such as editing 3D maps, programming environmental mechanics, and specifying unique triggers and end conditions. The corresponding subsections detail some of the scenarios employed in experiments.

A. SCENARIOS

The scenarios used to test the agents are described in this section. General details are offered for each scenario, for example, reward design, the agent's goals, and potential paths to achieving the objective. Additionally, settings for screen resolution, frame skipping [39], and screenshots for each scenario are provided.

B. DEFEND THE CENTER

As depicted in Fig. 3, in this scenario, an agent with a gun and restricted ammo reappears in the center of the circle, able to turn and fire only left or right. In addition, five hostile enemies approach the central agent from random locations by the wall. When an agent murders one, another appears out of nowhere.

The agent tries to kill the maximum number of opponents allowed; each enemy killed earns a +1 reward, while death earns a -1. Death is possible as the agent has an inadequate supply of ammo. Even though reward shaping has been shown to enhance agent performance, it is decided not to use it in this case.

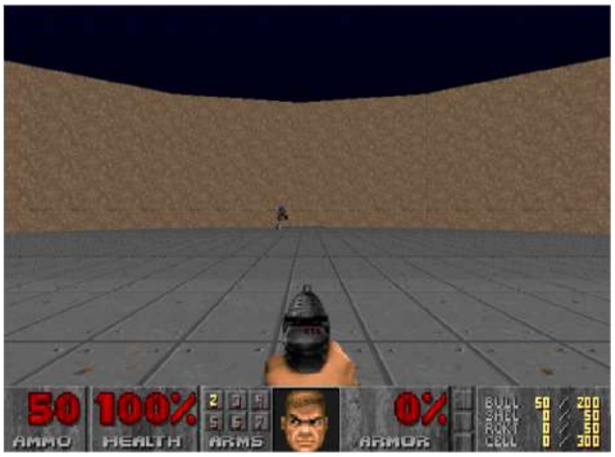


FIGURE 3. A screenshot of the 'Defend the center' scenario.

C. DEADLY CORRIDOR

This scenario is a hallway with the agent spawning at one end and a green vest placed at the other. The position of the monster pairs is depicted in 0Fig. 4 as being on either side of the corridor. The agent may receive both positive and negative rewards from his surroundings. It varies depending on how close the agent is to the vest.

D. HEALTH GATHERING

In this case, the agent appears in a room with a deadly material on the floor that continuously harms players with each step. The partially observable Markovian decision

TABLE 2. Considered hyperparameters in the experiments.

Parameters	Algorithms/Methods						
	A2C	A2C-LSTM	DDQN	DRQN	C51_DDQN	DFP	REINFORCE
Discount Factor	0.99	0.99	0.99	0.99	0.99	0.99	0.99
Learning Rates Range	0.0001 to 0.025						
Experience	60k	60k	60k	60k	60k	60k	60k
Screen Buffer	640, 480	640, 480	640, 480	640, 480	640, 480	640, 480	640, 480
Batch Size	32	32	32	32	32	32	32
Initial Decay	1.0	0.9	1.0	1.0	1.0	1.0	1.0
Final Decay	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001	0.0001
History length	4	4	4	4	4	4	4
Frames per Action	4	4	4	4	4	4	4
Time Elapsed (Hours)	50	60	70	70	72	48	61

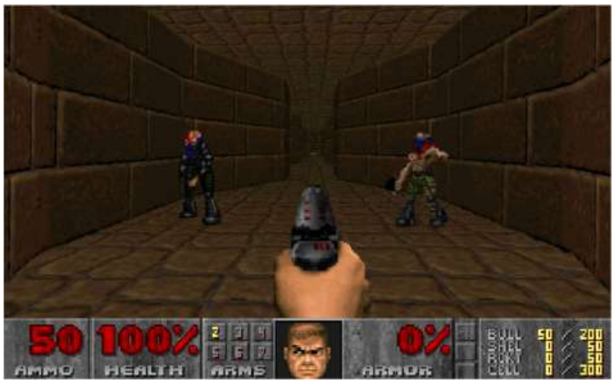


FIGURE 4. A screenshot of the 'Deadly Corridor' scenario.

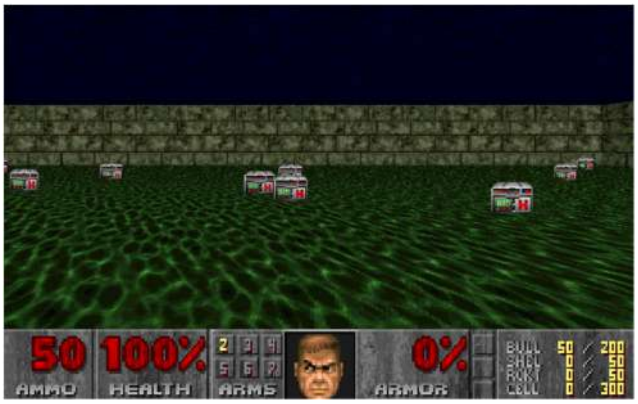


FIGURE 5. A screenshot of the 'Health Gathering' scenario.

processes (POMDP) setting that maintains the agent's health is unavailable to it. Turning left, turning right, and moving ahead are a few choices. As shown in 05, health kits on the ground help the agent stay alive and in good health when collected.

E. EXPERIMENTAL DESIGN AND ENVIRONMENT

This section describes the experimental settings and the hyperparameters (see Table 2) for training and evaluating the agents. A batch size of 32 and a sequence length of 4 are chosen. Before starting the steps, the target networks' weights are modified. The training process involves setting the parameter Epsilon to an initial value of 1.0 and linearly decreasing it after each training step until it reaches 0.0001 in the final steps. After a certain number of steps, the agents are tested with Epsilon set to 0.01, and the mean episode reward over 100 episodes is plotted. A memory (buffer) of size 60,000 is used. Only successive observation sequences are sampled from the memory, with the final observation serving as a terminal point. This modification was significant for the health-gathering scenario to prevent significant negative rewards if the agent dies during the last observation. Sometimes, several adjustments are made to improve training, including increasing the batch size and sequence length to 64 and 8, correspondingly. While the previous four experiences are utilized for training, the first four update the agent's memory. The number of steps before sampling from the memory is increased to 15, and the number of gradient steps is raised to 8000. The learning rate is also reduced to 0.0001 to fine-tune the training process. These adjustments aim to boost the training ability and effectiveness of the agents in various scenarios.

The overall implementation is carried out using PyCharm 2021 professional version, ViZDoom, Tensorflow [40], Keras [41], OpenCV [42], CMake, GCC, and Python 3.6 (64-bit) on a Windows 11 64-bit Operating System with Intel(R) Core(TM) i7-6650U CPU @ 2.20GHz and NVIDIA GTX 1070 GPU with RAM 16.00 GB (DDR), SSD 512 GB and HDD 1 TB. The algorithms learned thousands to millions of steps, playing actions, seeking conversions, and adjusting the networks.

The algorithms experimented in this paper are as follows.

F. ALGORITHMS

1) ADVANTAGE ACTOR-CRITIC (A2C)

The essential aspect of A2C is that it depends on n-step updating, which is a trade-off between Monte Carlo and Temporal Difference:

1. Monte Carlo stays till the end of an episode to update the value of an action via the sum of obtained rewards $R(s, a)$.

Algorithm 1 Advantage Actor-Critic (A2C) [15], [17], [43]

```
Initialize step counter $t \leftarrow 1$
Initialize episode counter $E \leftarrow 1$
repeat
    Reset gradients: $d\theta \leftarrow 0$ and $d\theta_v \leftarrow 0$.
    $t_{start} = t$
    Get state $s_t$
    repeat
        Perform $a_t$ according to policy $\pi\ (a_t|s_t;\theta)$
        Receive reward $r_t$ and new state $s_{t+1}$
        $t - t + 1$
    until terminal $s_t$ or $t - t_{start} == t_{max}$
    $R = \begin{cases} 0 \\ (st, \theta v) \end{cases}$
    for $i \epsilon \{t-1,\ldots, t_{start}\}$ do
        $R - r_i + \gamma R$
        Accumulate gradients wrt $\theta$ : $d\theta - d\theta + \nabla_\theta \log \pi\ (a_i|s_i;\theta)(R - V(s_i;\theta_v)) + \beta_e \partial H(\pi(a_i|s_i;\theta))/\partial\,\theta$
        Accumulate gradients wrt $\theta_v$ : $d\theta_v - d\theta_v + \beta_v(R - V(s_i;\theta_v))(\partial V(s_i;\theta_v)/\partial\theta_v)$
    end for
    Perform an update of $\theta$ using $d\theta$ and of $\theta_v$ using $d\theta_{v.}$
    $E - E + 1$
until$E > E_{max}$
```

2. Temporal difference immediately updates the action through the immediate reward $r(s, a, s')$ and estimates the rest with the value of the subsequent state $V^{\pi}(s)$.
3. N-step uses the '***n***' subsequent immediate rewards and estimates the rest with the value of the state visited'***n***' steps later, as in (5).

$$\nabla_\theta J(\theta) = \mathbb{E}_{st\sim\rho^{\pi},a\ t\sim\pi\theta}\left[\nabla_\theta \log \pi\theta(s_t, a_t)\left(\sum_{k=0}^{n-1}\gamma^k r_{t+k+1} + \gamma^n V_\varphi(s_{t+n+1}) - V_\varphi(s_t)\right)\right] \quad (5)$$

Temporal difference could, thus, be seen as a 1-step algorithm. For sparse rewards, mostly zero, +1, or -1 at the end of a game, this allows updating the 'n' last actions, which takes to a win or lose, as a substitute for only the last one in temporal difference, fasting up learning. But there is no need for limited episodes as in Monte Carlo. To be more precise, n-step approximation confirms a trade-off between bias, the wrong updates based on approximated values as in temporal difference and variance, and variability of the obtained returns as in Monte Carlo. The advantage actor-critic (A2C) algorithm is stepwise presented as follows.

The key steps in the advantage actor-critic (A2C) algorithm are:

1. Accumulating Gradients:

For each time-step 'I' in the episode, from 't-1' back to 't_{start}', the algorithm does two things:

- Actor Gradient: It gathers the gradient of the log probability of the considered action 'a_i' under the policy 'π' concerning its parameter vector 'θ.' This gradient is multiplied by the advantage term (R - V(s_i; θ_v)), which shows how better or worse the considered action performed compared to the approximated state value.
- Critic Gradient: It blends the gradient of the state-value function V(s_i; θ_v) regarding its parameter vector 'θ_v'. This gradient is multiplied by the advantage term multiplied by 'β_v,' which helps the critic learn faster from vast advantages.

2. Update Parameters:

- After blending gradients for the complete episode, the algorithm updates both 'θ' (actor) and 'θ_v' (critic) using the gathered gradients.

Essentially, this critical step uses the advantage term to guide both the actor toward better actions and the critic toward more accurate value estimates. Performs on-policy gradient updates, meaning the gradients are based on the actions taken by the current policy.

2) A2C-LSTM

A2C-LSTM is a robust RL algorithm that combines the strengths of A2C and LSTMs to handle sequential data and learn complex policies efficiently. The actor learns a policy to choose actions in the scenario, while the critic part estimates the value of taking a specific action in each state. A2C updates both the actor and critic based on the advantage, which is the difference between the estimated value and the actual reward received. The LSTM captures long-term dependencies in sequences. In A2C-LSTM, the critic often uses the LSTM to process sequential observations from the scenario and estimate the state value successfully. It helps the critic learn from past observations and make better value predictions in challenging scenarios. Due to the power of LSTMS, It is applied to tasks with large and complex state spaces, such

Algorithm 2 Advantage Actor-Critic-Long Short-Term Memory (A2C_LSTM) [9], [15], [16], [44]

```
Initialize the actor parameters $\theta_a$ and critic parameters $\theta_c$ of A2C-LSTM;
Initialize an empty buffer of length $T$ and the time-step $t = T + 1$;
for $i = 1$ to T do
    Randomly choose an action $a_i \in A$ and perform $a_i$;
    At the end of the $i$-th scheduling period, the agent gets the observation $p_i$;
    Append $p_i$ to the end of the buffer;
end for
repeat
    Concatenate each observation in the buffer to form $s_t = \{p_{t-T}, \cdots, p_{t-1}\}$ at the start of the $t$-th
    scheduling period;
    The state processing network takes $s_t$ as input and calculates $s'_t$;
    The policy network takes $s'_t$ as input and calculates the probability distribution of actions $\pi(a_t|s_t)$;
    The value network takes $s'_t$ as input and calculates the state value $V(s_t)$;
    The agent samples the action '$a_t$' according to $\pi(a_t|s_t)$ and performs '$a_t$';
    The agent receives the reward $r_t$ and gets the new observation $o_t$;
    Delete the first observation in the buffer and append $p_t$ to the end of the buffer;
Until The predefined maximum number of iterations has been completed.
```

as VizDoom scenarios. The pseudocode of the A2C_LSTM algorithm is as follows.

The key steps in A2C-LSTM involve the state processing and action selection at the beginning of each scheduling period

1. Concatenating Observations:
   - The algorithm extracts a sequence of past experiences ($p_{t-T}, p_{t-T+1}, \ldots, p_{t-1}$) from the memory of length'' and combines them into a single state's_t', which allows the LSTM network to capture the temporal context and dependencies between past actions and current decisions.
2. State Processing Network:
   - The state 's_t' is supplied into the state processing network, potentially an LSTM that extracts higher-level features and representations from the sequence of observations, delivering more informative input for the policy and value networks.
3. Policy Network and Action Selection
   - The processed state 's'_t' is then supplied to the policy network that produces the probability distribution over possible actions$\pi(a_t|s_t)$.
   - From this distribution, the agent samples an action'a_t' to execute for the current scheduling period.

This step (3) effectively influences the LSTM's capability to learn from sequences by:

- Memorizing past experiences: The memory stores a history of experiences, allowing the LSTM to consider prior actions and their outcomes when making new decisions.
- Temporal Dependency Modeling: The chain of experiences captures the order and relationships between past events, empowering the LSTM to learn more complex temporal changing aspects.

Thus, the pattern of state processing and LSTM networks lets A2C-LSTM make informed decisions based on its past observations and current circumstances, making it suitable for tasks with temporal dependencies.

3) DOUBLE DQN (DDQN)

DDQN is the upgraded version of the traditional DQN algorithm that assists in mitigating the overestimation issue of Q-value. The double DQN algorithm acts based on a different Q-network to fix the flaws in DQN. It does this by having two reproductions of a single Q-network and revising its weights every predetermined number of turns to synchronize the two reproductions of weights. Updates are performed based on values that the target Q-network suggests, which fixes the DQNs' flaw of either overexploiting or underexploiting their potential. Training a double deep Q-network on VizDoom scenarios is a computationally intensive task that requires substantial experimentation, hyperparameter tuning, careful monitoring, and constant patience during the entire training process. The pseudocode of the DDQN algorithm is as follows.

A brief explanation of the key steps in the DDQN algorithm is as follows:

1. Initialization:
   - Initialize the Q-networks, Q^A and Q^B with similar parameters.
   - Define the current state 's'.
2. Action Selection and Experience:
   - Choosing action 'a' based on a pattern of the estimated Q-values from networks (Q^A and Q^B) for the current state 's.'
   - Performing the chosen action 'a' in the scenario, receiving a reward 'r,' and observing the next state 's''.
3. Update Network:
   - Randomly choose one of the networks (either Q^A or Q^B) to update. This adds stochasticity and helps mitigate overestimation bias.
   - If Q^A *is to be updated*:

Algorithm 3 Double DQN (DDQN) [37], [45], [46], [47]

```
Initialize $Q^A$, $Q^B$, $s$
repeat
    Choose $a$, based on $Q^A(s, \cdot)$ and $Q^B(s, \cdot)$, observe $r$, $s'$
    Choose (e.g., random) either UPDATE(A) or UPDATE(B)
    If UPDATE(A) then
        Define $a* = arg\ max_a\ Q^A(s', a)$
        $Q^A(s, a) \leftarrow Q^A(s, a) + \alpha(s, a)\ (r + \gamma Q^B\ (s', a*)$ - $Q^A(s, a))$
    else if UPDATE(B) then
        Define $b^* = arg\ max_a\ Q^B\ (s', a)$
        $Q^B(s, a) \leftarrow Q^B(s, a) + \alpha(s, a)\ (r + \gamma Q^A(s', b) - Q^B(s, a))$
    end if
    $s \leftarrow s'$
untilend
```

  - Find the greedy action '$a*$' with the highest Q-value from the target network Q^B on the next state 's''.
  - Update the Q-value of Q^A for the current state-action pair (s, a) using the Bellman equation. This includes the reward 'r,' the discounted future Q-value from the target network Q^B using the chosen greedy action '$a*$,' and a learning rate 'α.'
- If Q^B *is to be updated*:
  - Like the update for Q^A, find the greedy action '$b*$' from the target network Q^A on the next state s'.
  - Update the Q-value of Q^B for the current state-action pair (s, a) using the Bellman equation with 'r,' discounted future Q-value from the target network Q^A using '$b*$,' and the learning rate 'α.'

4. Update State:
   - Se the current state 's' to the new state 's'' from the previous step.
5. Iteration and Target Network Updates:
   - This process repeats until a termination condition is met (e.g., episode ends, maximum steps reached).
   - Periodically, the parameters of the target networks (Q^A and Q^B) are updated by copying the weights from the main networks (also Q^A and Q^B). This allows for a slower, more stable target for learning, mitigating overestimation further.

4) DRQN

DRQN was projected to overcome the issues of partial observability [48]. In this article, combining the deep recurrent Q-network with gated recurrent unit (GRU) cells is considered [49] instead of using LSTM. GRU cells offer RNNs better convergence and require less training time than LSTM cells in RNNs, as demonstrated by [40] and [50]. The DQN consisted of GRU cells that take the observation space as an input. The GRU is configured to return the last output in the output sequence as an input for the next dense layer. The subsequent layers are significantly related to the architecture of DQN, where RELU is used as an activation function unless the output layer of the estimated Q-value is returned. This process is connected with the action policy to be performed by the agent. The key step in the DRQN algorithm is the training with action augmentation loss and backpropagation through time (BPTT). This happens when the replay memory buffer 'D' is filled, and the step reaches a multiple of its size (step mod T = 0). A brief explanation of the key steps in the DRQN algorithm is as follows:

1. Sample a sequence of length T from D: This captures the temporal dependencies between states and actions, crucial for learning long-term effects in a dynamic environment.
2. Train network 'Q_s' with action augmentation loss: This loss function combines Q-learning with the information stored in the augmented transitions (s, a, r, s'). This allows the network to learn the value of taking specific actions in states and the consequences of those actions in the future.

BPTT: This allows the network to express gradients over time by recognizing previous actions and states for their contribution to the final reward received. The algorithm continues to update the target network, constantly synchronizing it with the main network to help improve learning. The focus on action augmentation and temporal dependencies makes DRQN suitable for complex game environments and other sequential decision-making scenarios. The pseudocode of the DRQN algorithm is as follows.

Algorithm 4 Deep Recurrent Q-network (DRQN) [19], [25], [47], [51]

```
Initialize:
T $\epsilon$ N, recurrent Q-network $Q_s$, target network $Q_s$ - $\leftarrow Q_s$, dataset $D$ Simulate env E from dataset $D$
step $\leftarrow$ 1
Observe initial state '$s$' from env $E$
For each step, do
    step $\leftarrow$ step + 1
    Select greedy action w.r.t. $Q_s$(s. a) and apply to env $E$
    Receive reward r and next state $s'$ from env $E$
    Augment actions to form $T$ = ($s$, $a$,$r$,$s'$) and store $T$ in memory $D$.
    If D is filled and step mod T = 0, then
        Sample a sequence of length $T$ from $D$.
        Train network $Qs$ with action augmentation loss and BPTT
    end if
    Soft update target network $\theta^- \leftarrow (1 - r)\theta^- + r\theta$
end for
```

5) CATEGORICAL DOUBLE DEEP Q-NETWORK (C51_DDQN)

Training a categorical deep Q-Network (C51_DDQN) on VizDoom scenarios like 'defend the center,' 'deadly corridor,' and 'health gathering' is an appealing work. C51_DDQN is an extension of the standard DQN algorithm that controls the distribution of Q-values over a range of possible values instead of just estimating a single Q-value. The C51_DDQN divides the range of potential returns (cumulative rewards) into a fixed number of categories. The algorithm learns to predict the probability distribution over these categories for each state-action pair. C51_DDQN is especially useful in scenarios where the distribution of returns for different state-action pairs is not well-approximated by a single mean value. It provides a more robust and informative estimate of Q-values, which leads to improved decision-making in RL-based tasks. The pseudocode of the C51_DDQN algorithm is as follows.

Algorithm 5 C51_DDQN [9], [21], [52]

```
Input a transition $x_t, a_t, r_t, x_{t+1}, \gamma_t \in [1, 0]$
    $Q(x_{t+1}, a) := \sum_i z_i p_i(x_{t+1}, a)$
    $a^* \leftarrow arg\ max_a\ Q(x_{t+1}, a)$
    $mi = 0, i \in 0, \ldots, N-1$
    for $j \in 0, \ldots, N-1$ do
        #Compute the projection of $\mathcal{T}z_j$ onto the support
        $\mathcal{T}z_j \leftarrow [r_t + \gamma_t z_j]\,{}^{V_{MAX}}_{V_{MIN}}$
        $b_j \leftarrow (\mathcal{T} z_j - V_{MIN}) / \Delta z$ # $b_j \in [0, N-1]$
        $l \leftarrow \lfloor bj \rfloor, u \leftarrow \lceil bj \rceil$
        # Distribute probability of $\mathcal{T}z_j$
        $m_l \leftarrow m_l + p_j (x_{t+1}, a*) (u - b_j)$
        $m_u \leftarrow m_u + p_j (x_{t+1}, a*) (b_j - l)$
    end for
Output $-\sum_i m_i \log p_i(x_t, a_t)$ # Cross – entropy loss
```

It has been applied successfully in various challenging environments where traditional DQN approaches struggle to converge or generalize effectively. Bellemare et al. [52] suggest learning the value distribution (the probability distribution of returns).

They reveal that understanding the complete distribution of rewards rather than their mean improves performance on Atari games compared to more recent DQN iterations—

A brief explanation of the key steps in the C51_DDQN algorithm is as follows:

1. Input and Bellman Update:
- The algorithm takes a transition (state 'x_t,' action 'a_t,' reward 'r_t,' next state 'x_{t+1}', discount factor 'γ_t' as input.
- It computes the target Q-value for the next state and action (a∗) using the Bellman equation, and instead of a single value, it calculates a probability distribution over a range of reward values (z_i).
2. Finding Maximum Action:
- Like DDQN, it identifies the action 'a∗' with the highest expected reward based on the calculated distribution.
3. Reward Discretization:
- The algorithm discretizes the projected reward ($r_t + \gamma_t z_j$) into 'N' bins with a fixed spacing 'Δz.'
- It calculates the bin index 'b_j' for the projected reward.
4. Probability Distribution Update:
- The reward probability is distributed between two adjacent bins, 'l' and 'u,' based on the fraction ($b_j - l$) and (u - bj).
- Two variables, 'm_l' and 'm_u,' accumulate the probability mass from 'b_j' for the lower and upper bins, respectively.
5. Output and Loss Calculation:
- The output is the negative cross-entropy loss between the actual action probability $p_i(x_t, a_t)$ and the updated reward probability distribution. This encourages the network to predict the chosen action and its corresponding reward distribution accurately.

6) DIRECT FUTURE PREDICTION (DFP)

DFP prediction is a model-based reinforcement learning used to learn a predictive model of the scenario. This approach involves predicting the consequences of taking different actions in the environment and using these predictions to make better decisions. This is like traditional RL settings, where an agent learns from the feedback from acting together with the environment. In standard RL, feedback comes in the form of a scalar reward. In DFP, feedback comes as measurements (m). It could be visualized as a complex vector with various game-related characteristics represented by each element (e.g., Deaths, Armor, Health, etc.) The DFP algorithm aims to improve forecasting accuracy by directly predicting future values. —the pseudocode of the DFP algorithm is as follows.

Algorithm 6 Direct Future Prediction(DFP) [9], [22], [53]

```
Define sets of model parameter values to evaluate
for each parameter set, do
    for each resampling iteration, do
        Hold-out specific samples
        [Optional] Pre-process the data
        Fit the model on the remainder.
        Predict the hold-out samples.
    for
    Calculate the average performance across hold-out predictions.
end
Determine the optional parameter set.
Fit the final model to all the training data using the optimal parameter set.
```

A brief explanation of the key steps in the DFP algorithm:

1. Define Parameter Sets:
- Identify and define a set of possible parameter values for the model to use (e.g., learning rate, number of hidden neurons).
2. Nested Loop:
- The algorithm employs a nested loop structure:
- Outer Loop: Iterates through each defined parameter set.
- Inner Loop: Performs repeated resampling within each parameter set.

3. Resampling (Inner Loop):
- Hold-out Samples: A specific subset of samples is withheld as a hold-out set for validation in each iteration.
- Optional Pre-processing: Before fitting the model, the remaining data can be pre-processed (excluding the hold-out set).
- Model Fitting: The model is trained on the pre-processed data (excluding the hold-out set) using the current parameter set.
- Predict Hold-out Samples: The trained model is used to predict the values of the hold-out samples.

4. Performance Calculation (Outer Loop):
- Across all resampling iterations within a parameter set, the average performance on the hold-out set predictions is calculated (e.g., mean squared error).

5. Optimal Parameter Set:
- After iterating through all parameter sets, the parameter set with the best average performance on the hold-out sets is selected as the optimal set.

6. Final Model Training:
- Using the optimal parameter set, the final model is trained on all the training data (including the previously held-out samples).

7) REINFORCE

REINFORCE belongs to the family of reinforcement Learning algorithms known as policy gradient algorithms. It involves creating a Policy, essentially a model that takes a state as input and produces the probability of taking specific actions as output. This policy guides the agent, recommending the best actions in different states. The REINFORCE algorithm iteratively refines this policy at each stage until it finds an approach that successfully solves the environment. This iterative process is outlined in the pseudocode of the REINFORCE algorithm.

Algorithm 7 REINFORCE [5], [15], [23]

```
  initialize $\theta$
  **For** each episode $\{s_1, a_1, r_2, \ldots, s_{T-1}, a_{T-1}, r_T\}$ $s$ sampled from policy $\pi_\theta$ **do**
  **For** $t = 1$ to $T - 1$, **do**
    $\theta \leftarrow \theta + a\nabla_\theta log\pi_\theta\,(s_t, a_t)\,G_t$
  **end for**
**end for**
```

The key steps in the REINFORCE algorithm can be summarized as follows:

1. Initialization:
- The algorithm begins by initializing the parameter vector for the policy, denoted by 'θ.'

2. Episode Loop:
- The algorithm iterates through episodes, each represented by a sequence of states (s), actions (a), and rewards (r). These

sequences are sampled from the current policy defined by 'π_θ.'

3. Action Gradient Update:
- Within each episode, the algorithm iterates through each time step (t) from 1 to T-1 (excluding the final step).
- At each time step, the algorithm updates the parameter vector 'θ' using the following steps:
  - Gradient Calculation: It calculates the gradient of the log probability of the chosen action 'a_T' at state 's_t'concerning the parameter vector 'θ.' This reflects how changing the policy parameters would affect the likelihood of taking that action in that state.
  - Return Accumulator calculates the "return" 'G_t,' the discounted sum of future rewards starting from the current time step 't.' This represents the long-term value of acting 'a_T' at state 's_T.'
  - Parameter Update: The gradient computed above is multiplied by the return 'G_t' and a learning rate 'α.' This product is then added to the current parameter vector 'θ.' This update encourages the policy to take actions that result in maximum long-term rewards.

4. Repeat:

The episode loop and action gradient update are repeated for all episodes in the training process. As the algorithm learns and updates the policy, it aims to find actions that advance to maximum overall rewards across all future episodes.

IV. RESULTS AND COMPARISON

Reinforcement learning algorithms assess the agent's performance in each scenario, relying on metrics such as remaining ammo, kill counts, score, and average health. These metrics are monitored across episodes during the training process, enabling a continuous evaluation of the agent's progress. After each training step, the goal is to achieve the desired results. To effectively showcase each agent's performance, each agent's results are visualized separately. This allows for a clear understanding of the differences between agents or facilitates comparisons between different models.

A. DEFEND THE CENTER

The agents' performances trained on the 'defend the center' scenario are detailed in this subsection, along with their learning curves. The agent's performance trained with the A2C algorithm is somewhat reasonable. The beginning performance of the agent trained with A2C remained uniform until 185000 steps; after that, the agent gathered more health packages stepwise, and at 340000 steps, the curve began reaching 77%, although the agent's average performance equals ~70%. Likewise, the agent's score, ammo left, and kill counts improved except at some steps where they declined, as presented in Fig. 6.

The A2C-LSTM algorithm performance presented in Fig. 7 is satisfactory. The arc of its health increased gradually till the last maximum number of steps, but later, it then declined to ~66%.

Moreover, the agent's score, kill counts, and ammo left in the 'defend the center' scenario are also lower at some steps at

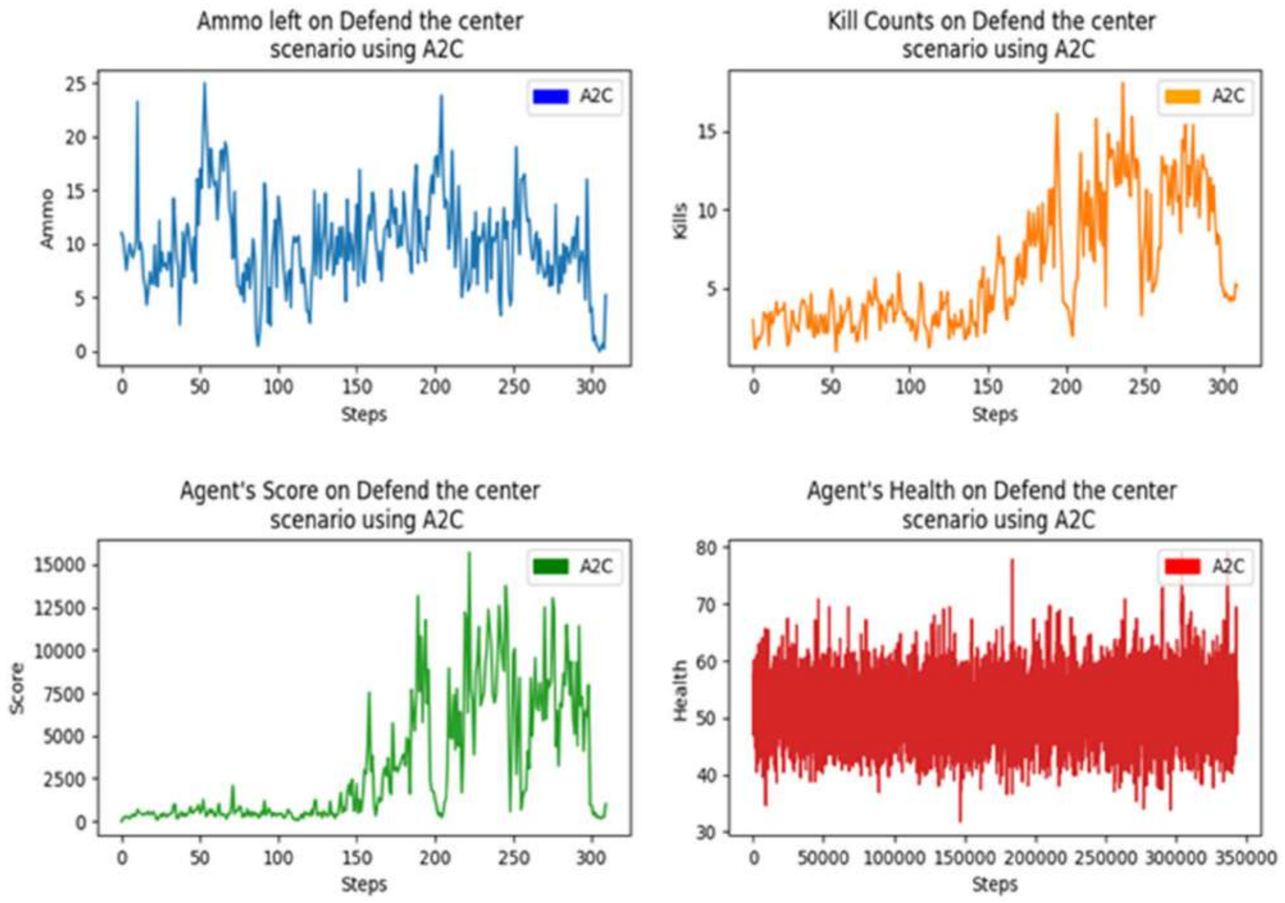


FIGURE 6. The agent act trained with A2C on the 'Defend the Center' scenario.

the end of the training. Therefore, it was supposed that A2C-LSTM could be the least preferred choice for training agents.

The agent learned with DDQN performs above average and scores satisfactorily in the scenario. Its overall kill counts and ammo left in the scenario are better and preserve health to ~85%, as shown in Fig. 8.

The performance of the DRQN agent trained on the VizDoom defend the center scenario remains good to some extent. DRQN mostly achieved results, and the crest and trough of its health most times remain between ~70% and ~82%, as shown in Fig. 9. On the other hand, the ammo left and kill counts also increased stepwise, but the agent's score declined. Also, it was concluded that DRQN could be the leading choice or priority for training game agents, specifically in the 'defend the center' scenario.

The C51_DDQN based agent is applied and analyzed on the 'defend the centre' map, which acted well initially, gaining more kills and saving enough ammo while shooting; however, it could not preserve and maintain enough of its health. Also, its score slightly dropped in the later steps, as demonstrated in Fig. 10. Thus, it can be supposed that the agent acted inadequately. Also, it was supposed that C51_DDQN could be least preferred for training agents, particularly in the 'defend the center' scenario.

The performance of the agent trained with DFP is presented in Fig. 11. The DFP agent learned and acted reasonably in gathering health packages and preserves its total health to ~98%; likewise, the agent maintains kill counts and ammo; however, the agent's scores are not much more prominent. Overall, DFP is considered the best among state-of-the-art reinforcement learning algorithms to train agents.

The agent trained with REINFORCE performed better and maintained its average health to ~80%. The score, ammo, and kill counts also increased gradually and steadily, as presented in Fig. 12. It is supposed that the agents learned with REINFORCE performed well, mainly using the 'defend the center' scenario. Moreover, REINFORCE can be considered among the best reinforcement learning algorithms for training agents.

B. DEADLY CORRIDOR

In this subsection, the learning curves and the performance of the agents trained on the 'Deadly Corridor' scenario are provided. The agent's performance trained with the A2C algorithm is somewhat reasonable. The beginning performance of the agent trained with A2C remained uniform until 175000 steps; after that, the agent gathered health with improvements steadily stepwise; at ~300000 steps,

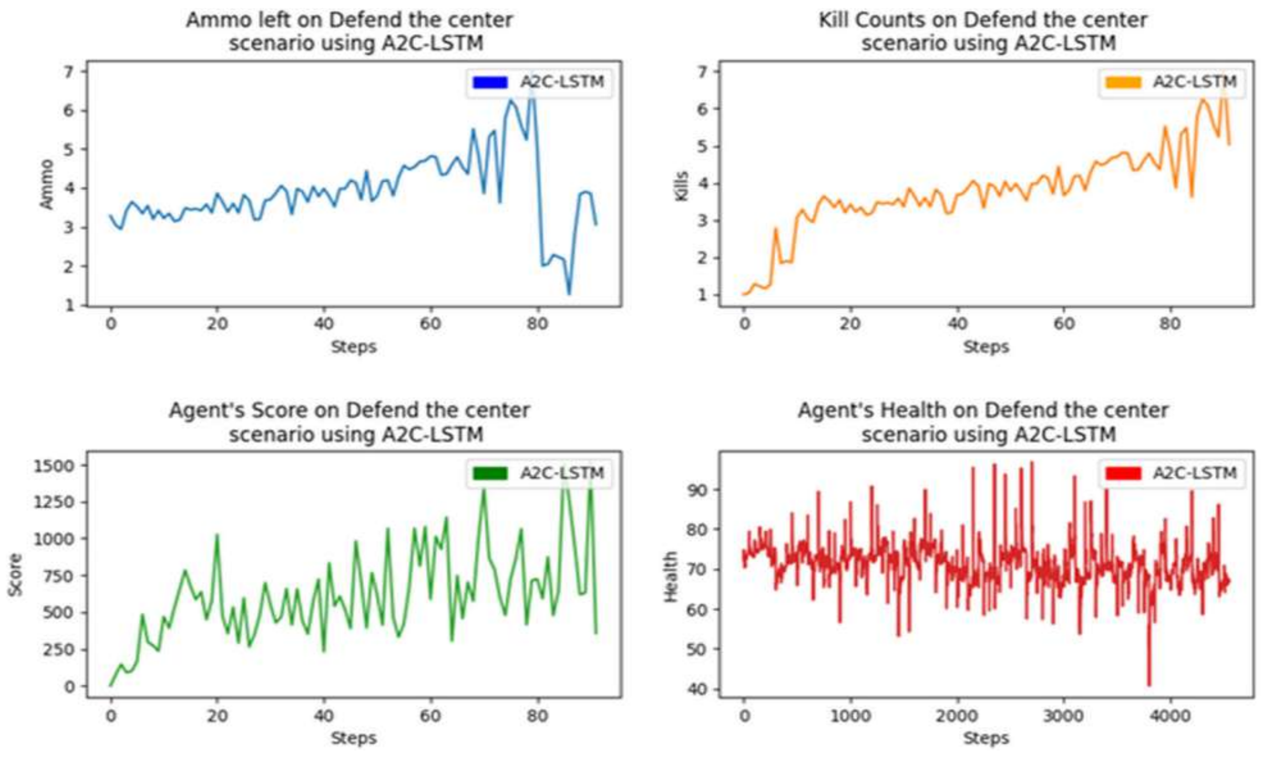


FIGURE 7. **The agent performance trained with A2C-LSTM on the 'Defend the Center' Scenario.**

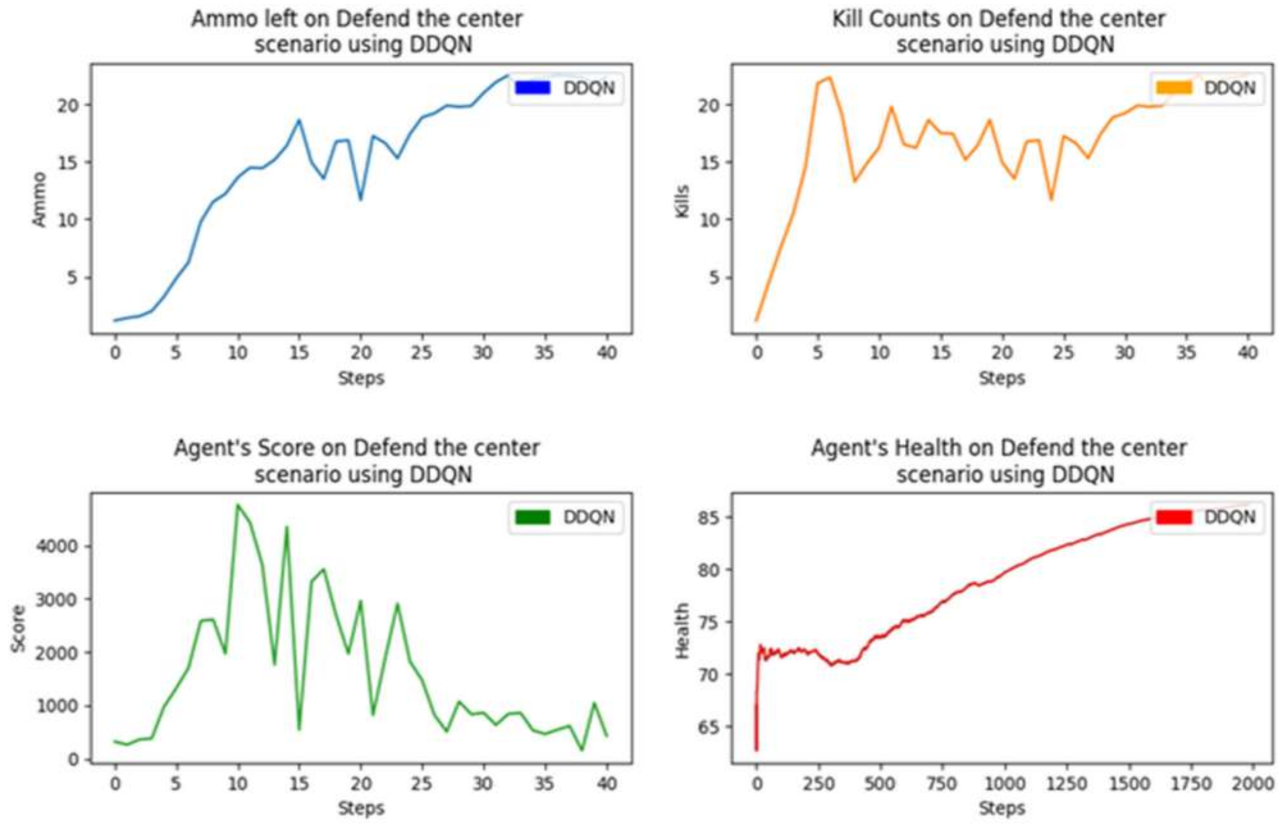


FIGURE 8. **The agent performance trained with DDQN on the 'Defend the center' scenario.**

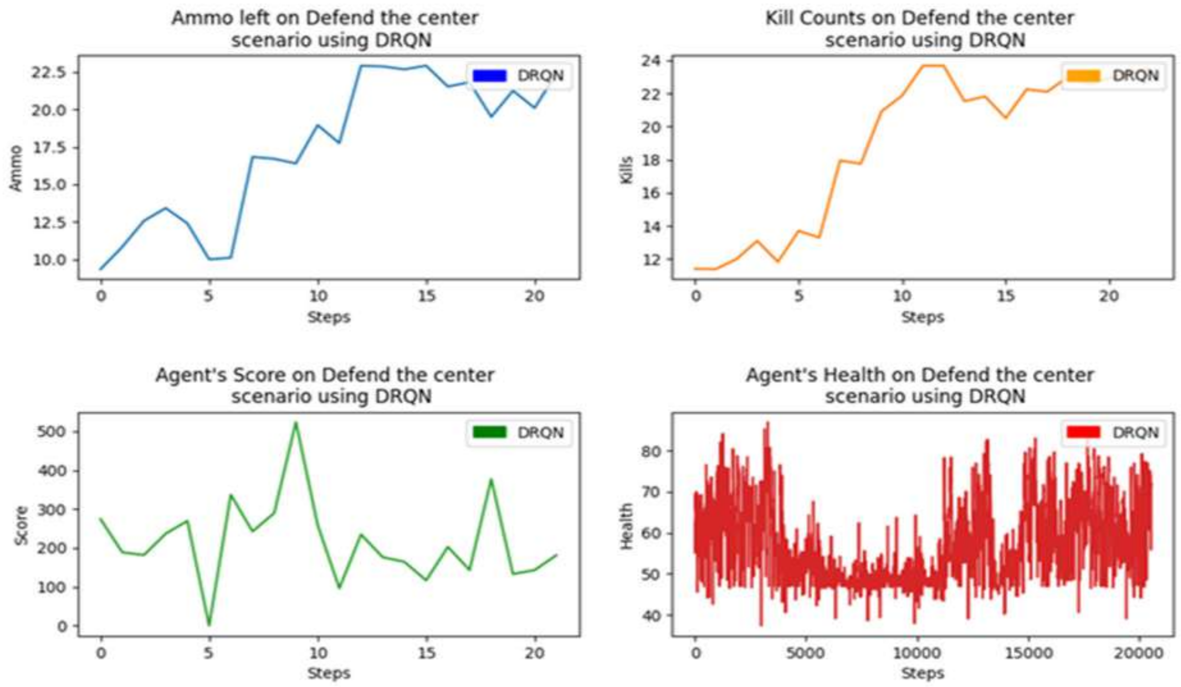


FIGURE 9. **The agent performance trained with DRQN on the 'Defend the center' scenario.**

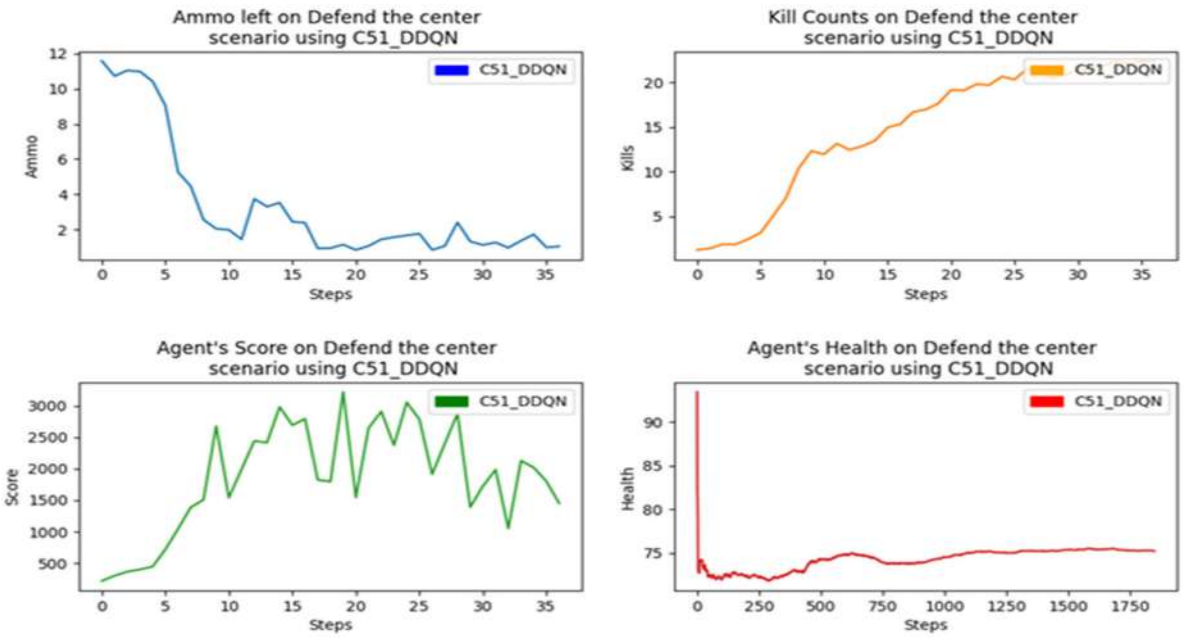


FIGURE 10. **The agent performance trained with C51_DDQN on the 'Defend the center' scenario.**

the performance arc started reaching ~71%. Likewise, the agent's score, ammo left, and kill counts improved gradually stepwise, which is presented in Fig. 13.

The A2C-LSTM algorithm acting graph is presented in Fig. 14, scores very least promising; the health usage kills, and ammo were found to be better as the health touches ~80^{+}%, whereas its score arc remained uniform till the last maximum steps and rarely highly rose or declined. Moreover, to confirm and ensure the final performance, the A2C-LSTM agent was trained repeatedly for longer steps to observe any progress in the score; though, training it for more extraordinary (millions) steps, the arc remained even. Thus, it was supposed that A2C-LSTM could be least recommended to train agents, particularly in Doom scenarios.

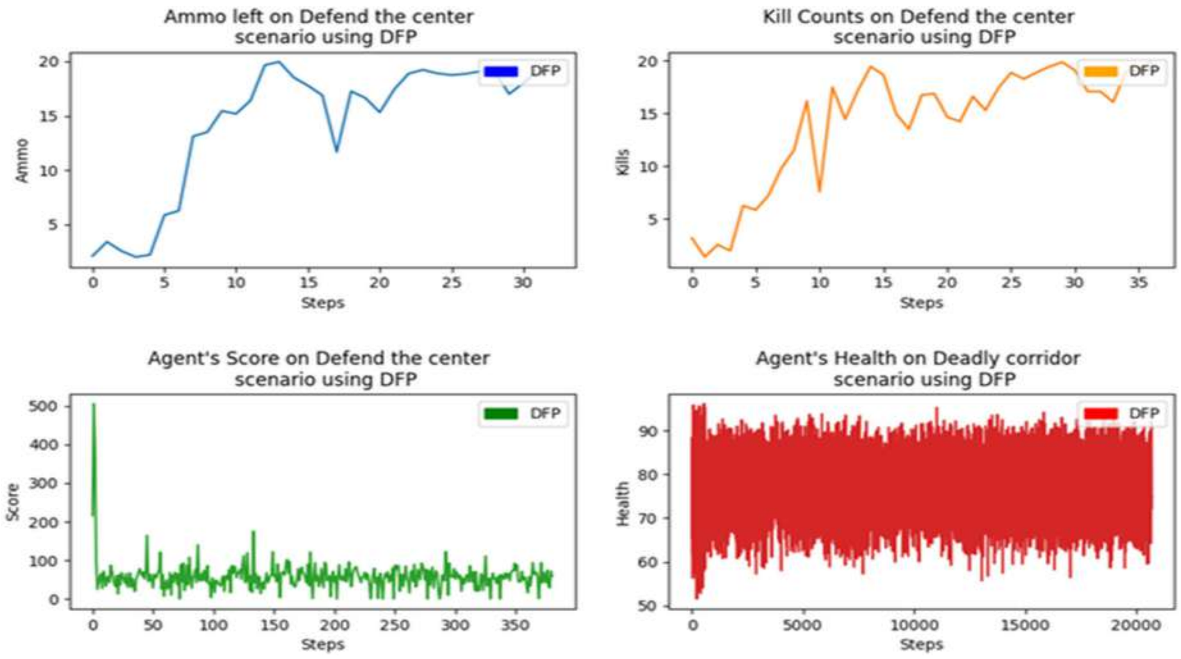


FIGURE 11. The agent act trained with DFP on the 'Defend the Center' scenario.

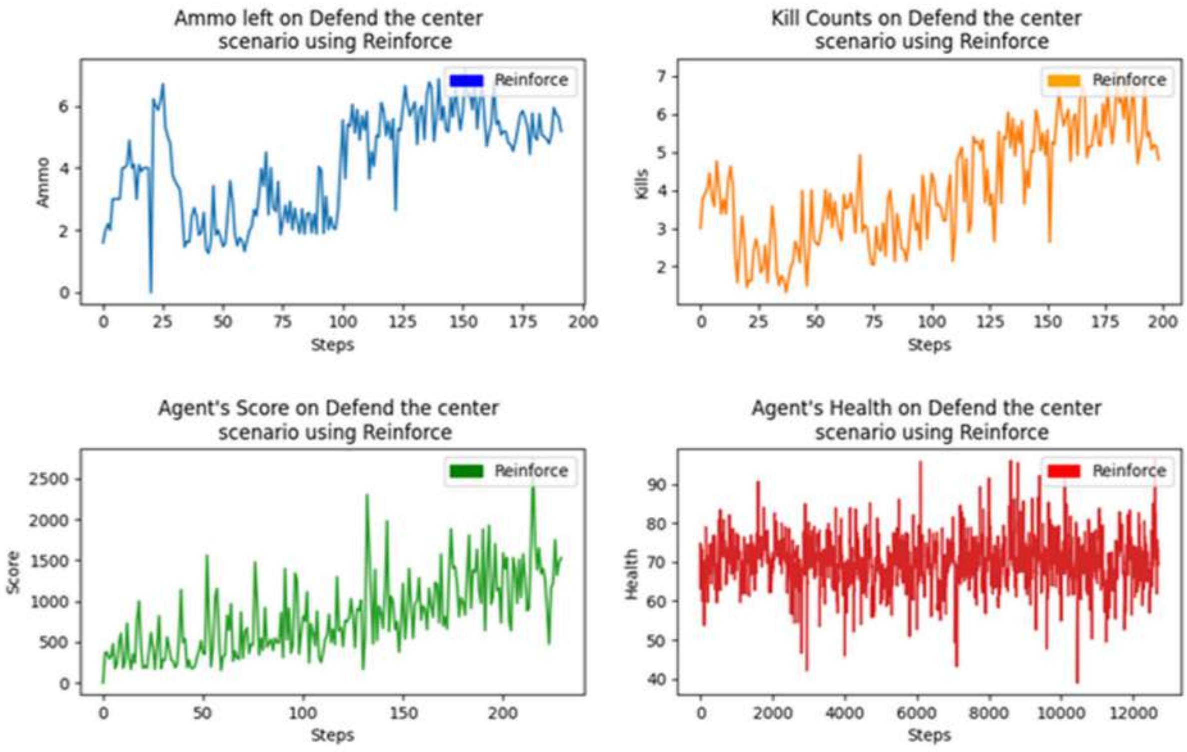


FIGURE 12. The agent performance trained with REINFORCE on the 'Defend the Center' scenario.

The DDQN algorithm-based agent scored on average by gathering health packages in the scenario. The agent could not maintain its health and dropped to ∼30% at the end, which is the poorest, as shown in Fig. 15. Also, it was supposed that DDQN could be least recommended to train agents, particularly in the deadly corridor scenario.

The performance of the DRQN agent trained on the VizDoom 'deadly corridor' scenario remained steady, declined

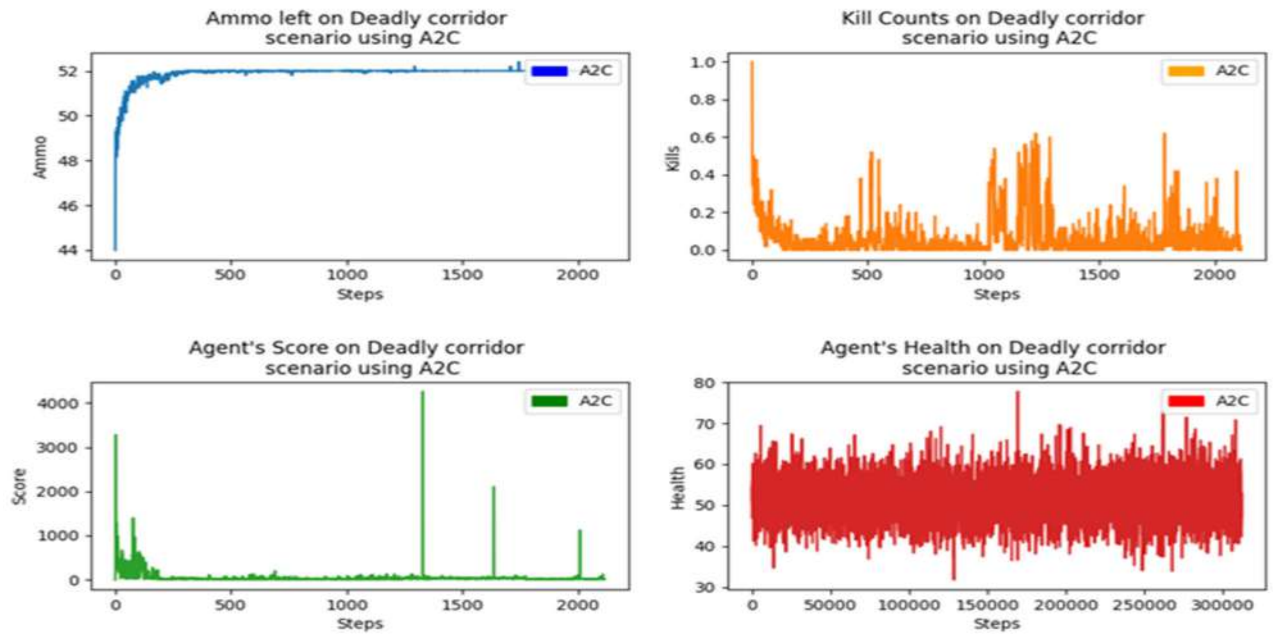


FIGURE 13. The agent act trained with A2C on the 'Deadly Corridor' scenario.

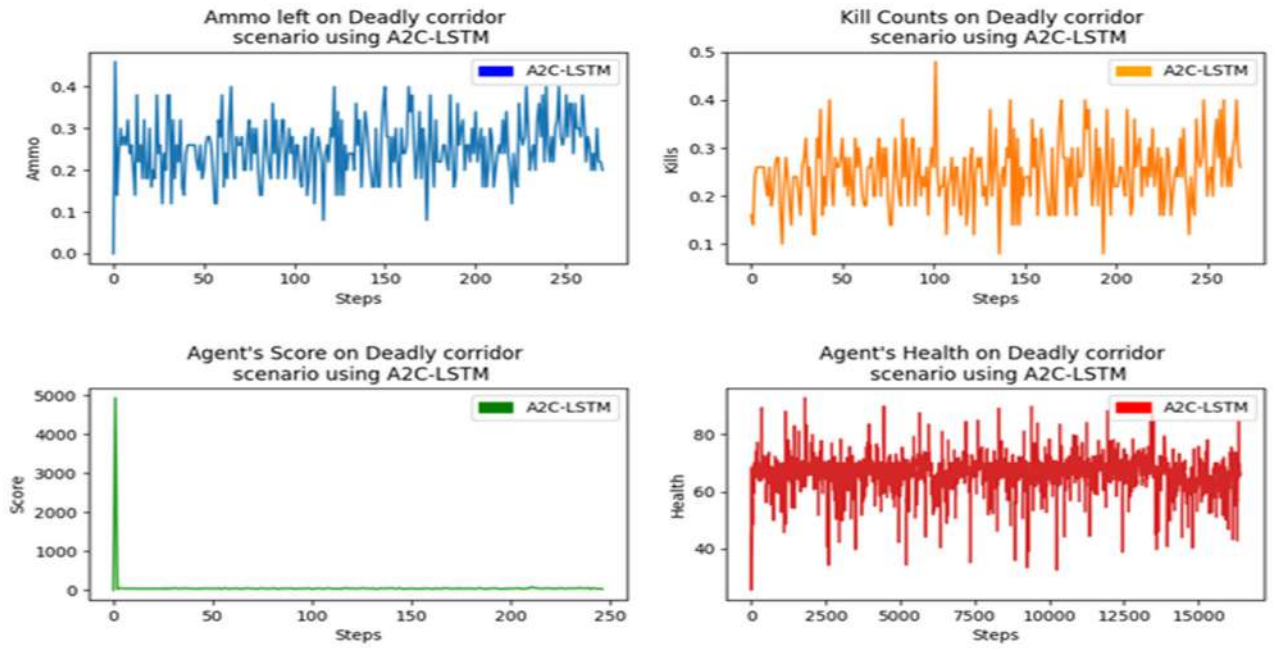


FIGURE 14. The agent performance trained with A2C-LSTM on the Deadly Corridor scenario.

later, and never managed to improve score, ammo, kills, and health to a high level. DRQN could not primarily achieve any impressive results, and the peak and low of its health often stayed between ∼50% and ∼60%, as shown in Fig. 16. The overall performance of DRQN agents trained in 'deadly corridor' scenarios is not promising. Finally, it is decided that DRQN should be less preferred for training agents.

The agent C51_DDQN is experimented with and evaluated on the 'deadly corridor' scenario. As shown in Fig. 17, the agent C51_DDQN performed much better and preserved its health to ∼89%. The agent score is steady; however, ammo usage is not too efficient, yet this performance can be considered reasonable to some extent. It is supposed that the agents learned with C51_DDQN acted well, specifically in the 'deadly corridor' scenario. Moreover, C51_DDQN can be considered among the best reinforcement learning algorithms for training agents.

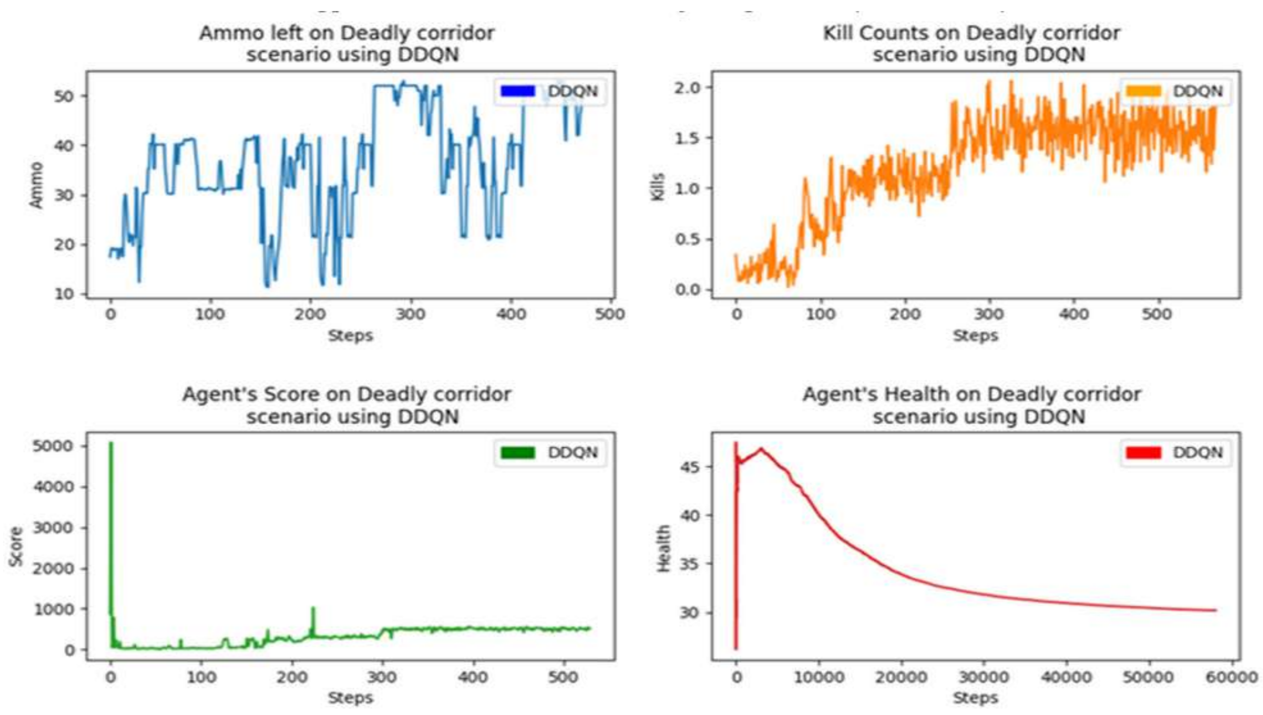


FIGURE 15. The agent performance trained with DDQN on the 'Deadly Corridor' scenario.

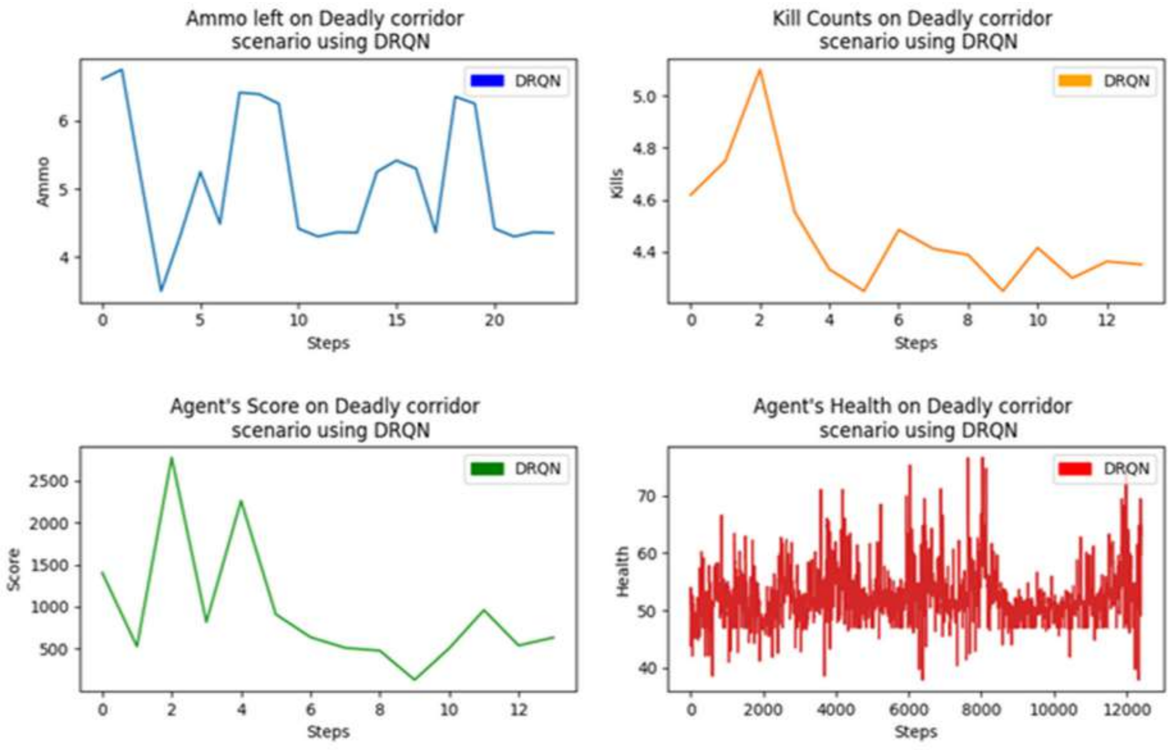


FIGURE 16. The agent performance trained with DRQN on the 'Deadly Corridor' scenario.

The performance curves of the agent learned with DFP are presented in Fig. 18, with a health preservation of almost ∼80%. The agent performed well in gathering health packs, consuming less ammo, and improving kill counts. However, the agent's score curve remains uniform. Moreover, with such statistics, DFP can be considered a cutting-edge and one of the best RL algorithms.

The agent trained with REINFORCE performed much better by preserving enough ammo and more than 80% health, with a steady score and gradual improvements; however,

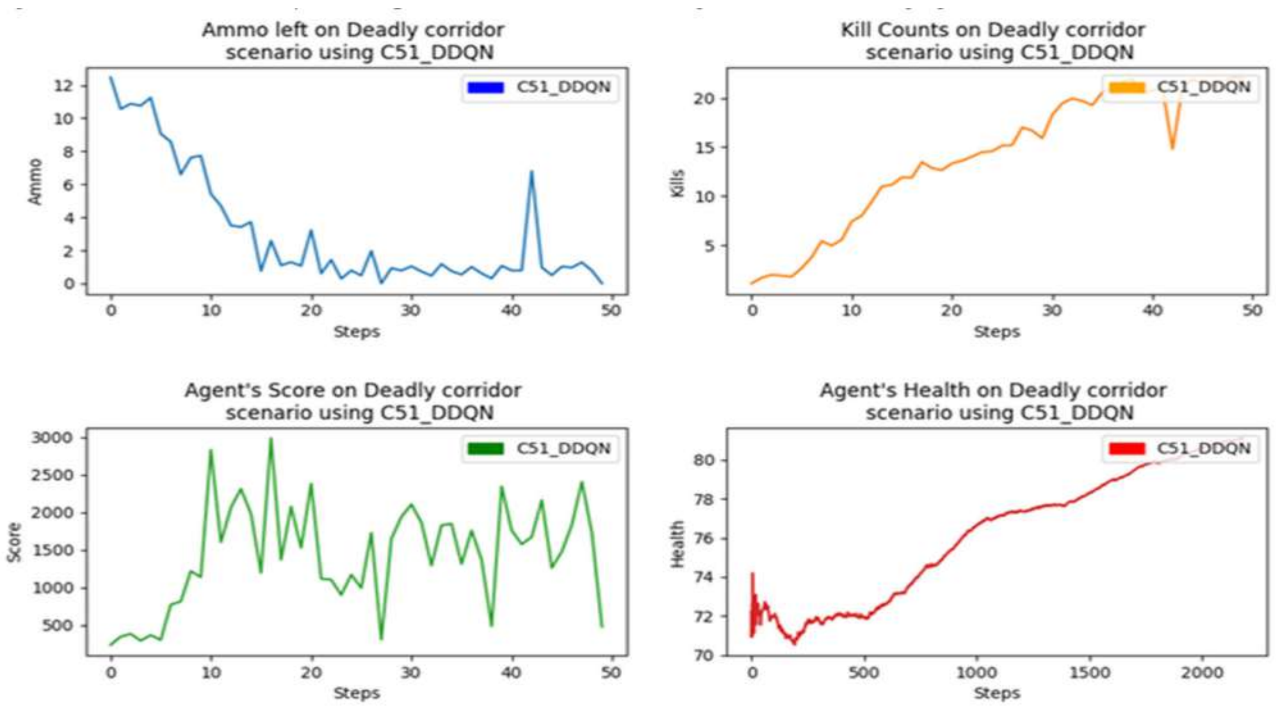


FIGURE 17. The agent performance trained with C51_DDQN on the 'Deadly Corridor' scenario.

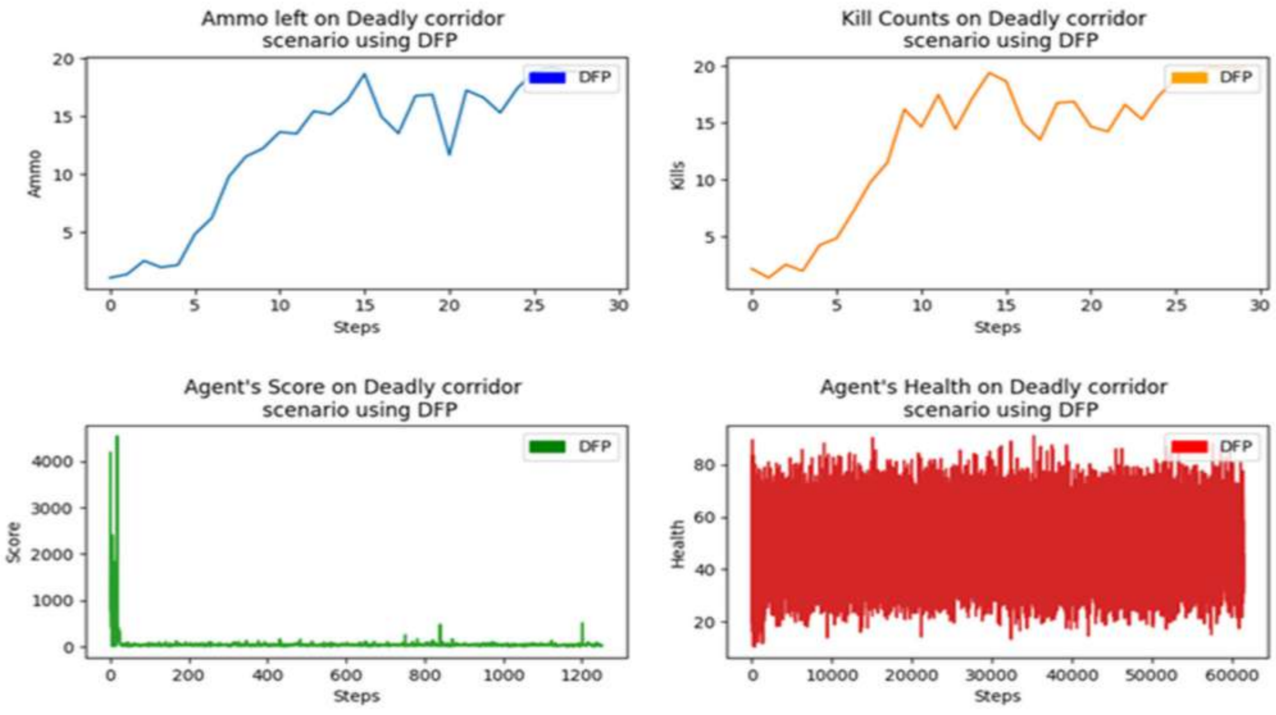


FIGURE 18. The agent performance trained with DFP on the 'Deadly Corridor' scenario.

it was less efficient in kills. Overall, with such performance, including score, ammo, health, and kills, as presented in Fig. 19, it is assumed that the agents learned with REINFORCE acted well, specifically in the deadly corridor scenario. Moreover, REINFORCE can be considered the best reinforcement learning algorithm to train agents.

C. HEALTH GATHERING

In this part, learning curves of the agents trained on the 'Health Gathering' scenario are provided. The agent's performance learned with the A2C algorithm is reasonable to some extent because we care less about other parameters, such as ammo and kills, in health-gathering scenarios.

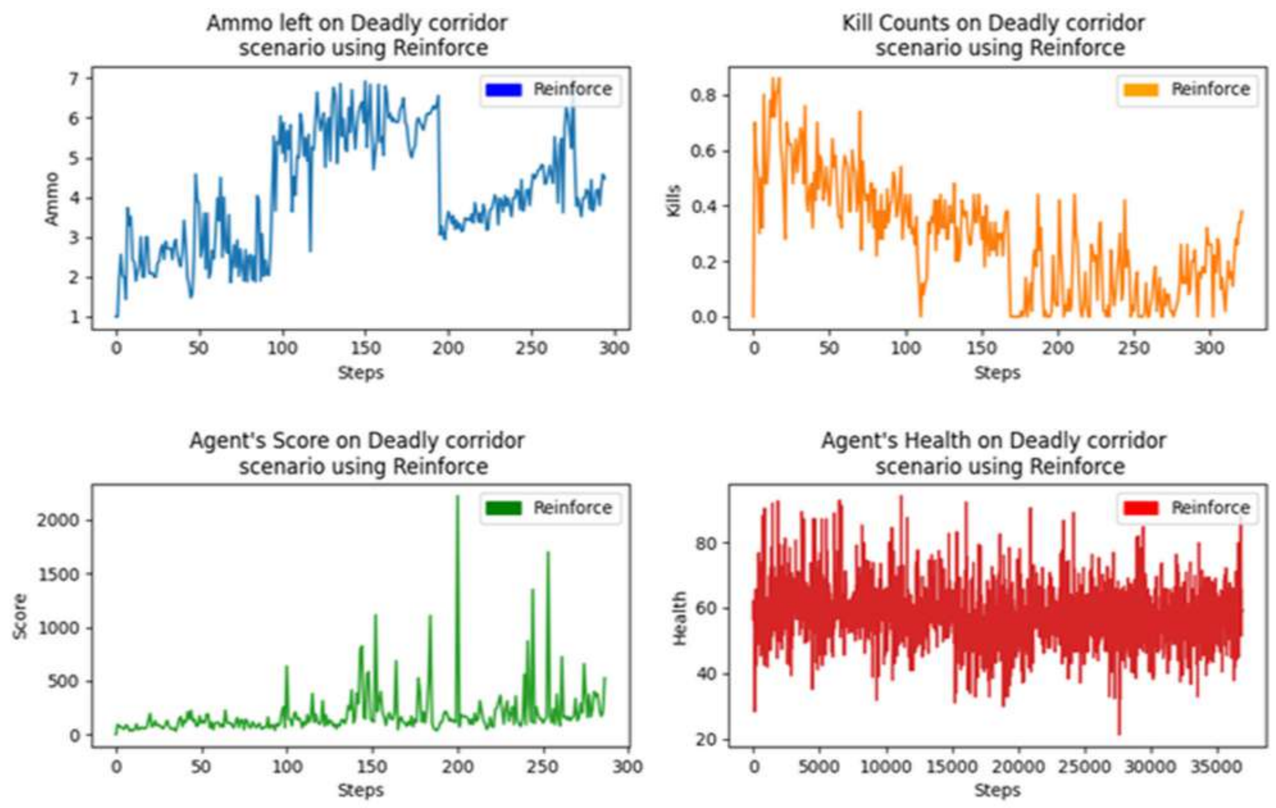


FIGURE 19. The agent performance trained with REINFORCE on the 'Deadly Corridor' scenario.

However, the beginning act of the agent's health learned with the A2C algorithm remained uniform until 30000 steps; after that, the agent health gathering act developed progressively stepwise, and at 60000 steps, the performance act began reaching 88%. Likewise, the agent's overall score improved stepwise except with a rare, declined step, which is presented in Fig. 20.

The A2C-LSTM algorithm is presented in Fig. 21. The agent score curve was initially impressive; however, it later remained consistent till the last maximum steps. It rose slightly and could not decline highly. Moreover, the agent also could not preserve enough health, which stayed on average in the range of 50 to 55 %. Thus, it is assumed that A2C-LSTM could be less recommended to train agents.

The agent learned with DDQN persists with a reasonable score in gathering health packages on the health gathering scenario. The agent can preserve its health to ∼55% on average, as shown in Fig. 22. It is an average performance in overall experiments. Thus, it is assumed that DDQN could be less recommended to train agents, particularly in the 'health gathering' scenario.

The act of the DRQN agent learned on the 'health gathering' scenario remains good except with a very rare decline in a step or so. However, it manages to improve it to a satisfactory level. DRQN achieved results primarily in gathering health packs, as shown in Fig. 23. Moreover, the health of the DRQN agent is much better. It is decided that DRQN can be among the top priorities for developing agents.

The C51_DDQN based agent is presented in Fig. 24. The agent C51_DDQN performed the least well on the 'health gathering' scenario and preserved its health to ∼59%, which is somehow influential. It is supposed that the agents learned with C51_DDQN acted the least well, specifically in the 'health gathering' scenario. Moreover, it is supposed that C51_DDQN can be, on average, the priority for training agents.

The act of the agent learned with the DFP algorithm is presented in Fig. 25, with a health preservation of almost ∼96%. The agent performed well in gathering health packages. The agent's score climbed further later. Moreover, DFP is found to be a valuable RL algorithm to train agents. DFP is among the state-of-the-art approaches, particularly in the 'health gathering' scenario.

The agent learned with REINFORCE was somehow reasonable and preserved its overall health to ∼60%, as shown in Fig. 26. Likewise, the score of the REINFORCE agent gains its high level, stays uniform to some extent, and then again climbs up. It infers that the agents learned with REINFORCE behave reasonably well, specifically in the 'health-gathering' scenario. Moreover, REINFORCE should be less preferred

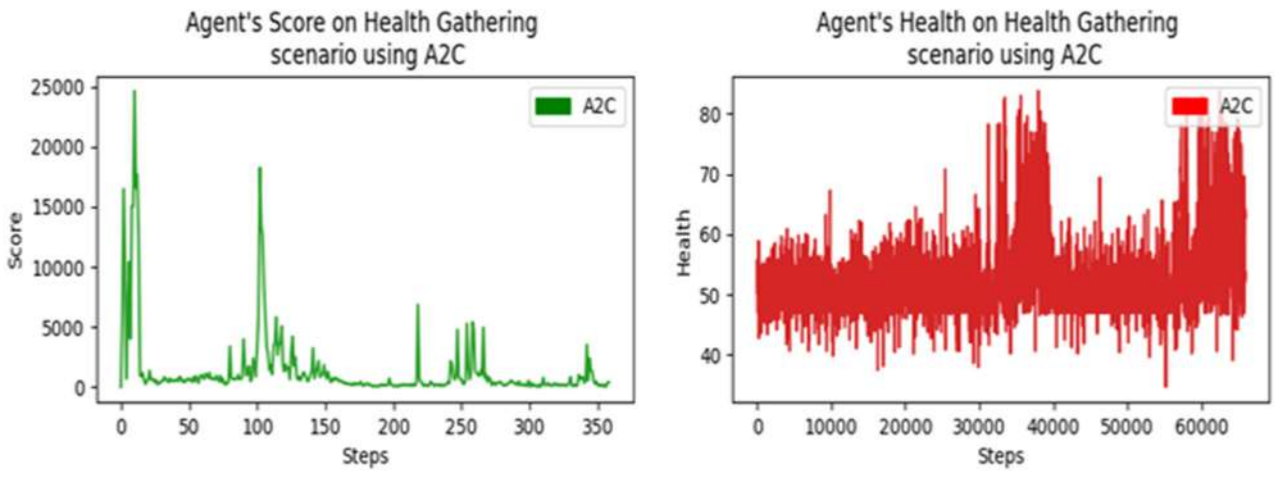


FIGURE 20. The agent performance trained with A2C on the 'Health Gathering' scenario.

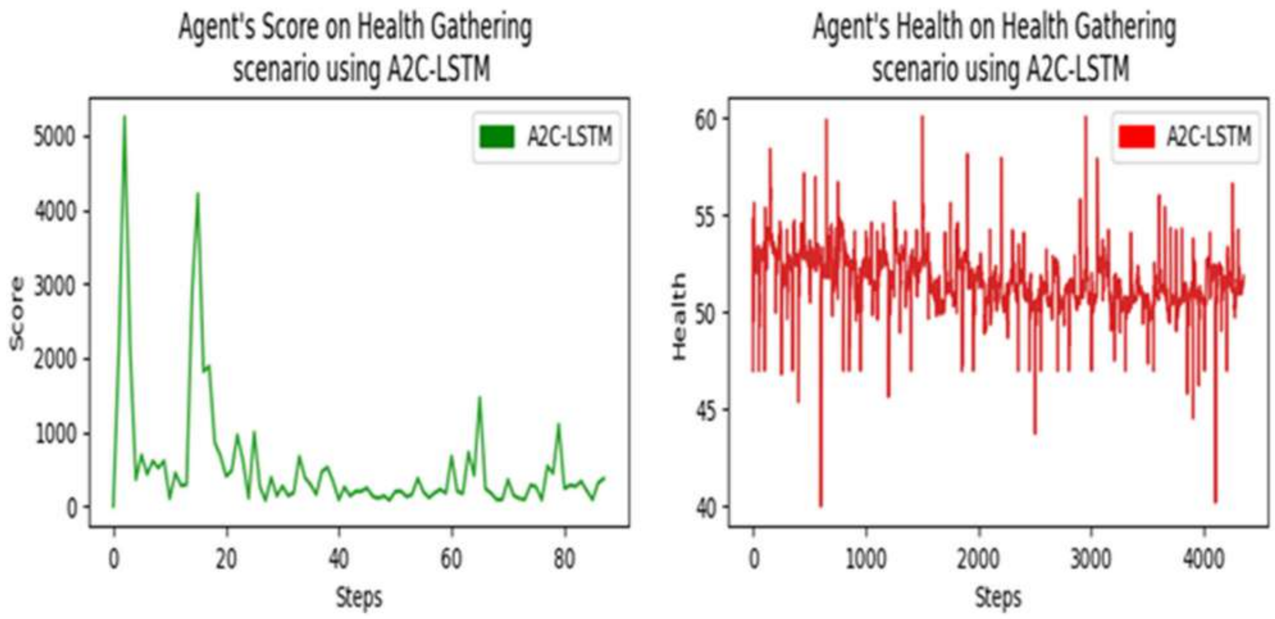


FIGURE 21. The agent performance trained with A2C-LSTM on the 'Health Gathering' scenario.

for training agents, particularly in the 'health-gathering' scenarios.

D. COMPARATIVE ALGORITHMS WITH BETTER PERFORMANCES

After training agents on the different scenarios, such as 'defend the center,' 'deadly corridor', and 'health gathering,' of the VizDoom platform using various RL algorithms, the act of the following three algorithms was found to be healthier, and more truthful as presented in Fig. 27. The x-axis denotes the training steps in thousands to millions. In contrast, the y-axis subsequently denotes the (ammo, score, kills, and health) of the agents in percentage. The agents that learned with the DFP algorithm manage approximately 90% to gather health packages and, on average, preserve their health, showing the best performance regarding ammo, kills, and average score. This is better than the other algorithms and remains the best choice for training agents. The agents learned with the REINFORCE algorithm fall after DFP in the act, which preserves, on average, almost 88% health in gathering health packages in the scenarios. It means the REINFORCE falls next-best RL algorithm to DFP to train agents, but it holds REINFORCE significantly longer to get those results. Further, the REINFORCE agent performs less satisfactorily in maintaining a score than the other algorithms until it takes longer steps. The C51_DDQN is the third-best RL algorithm that acted well explicitly, with an average performance of ∼86%.

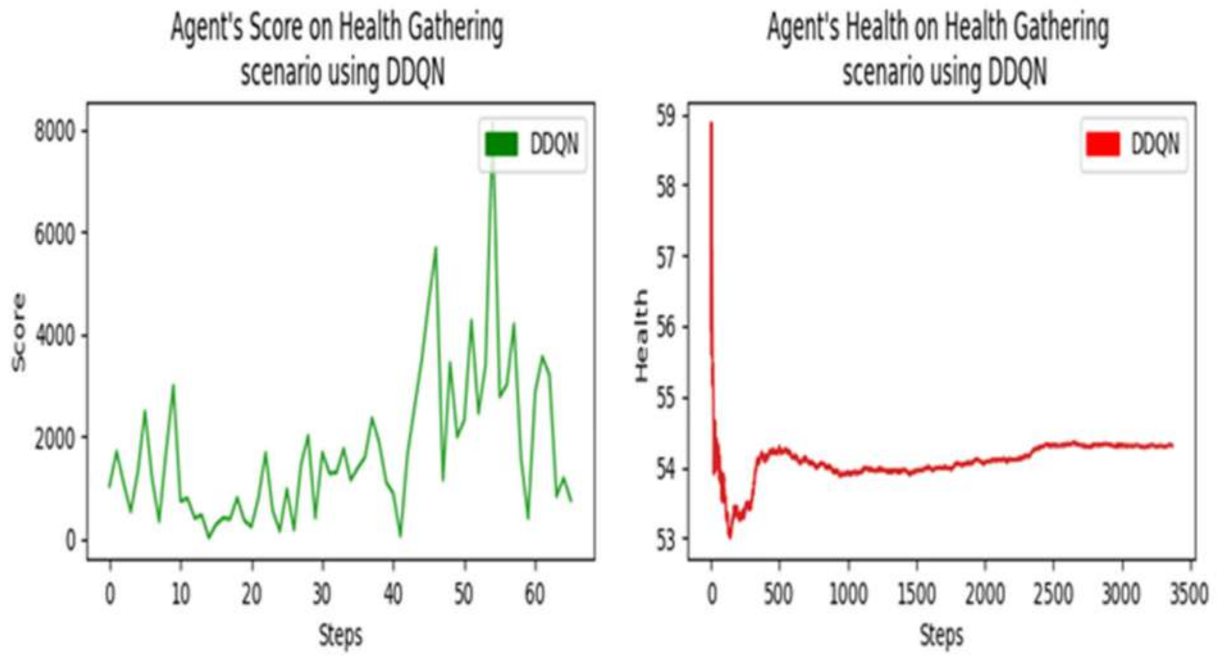


FIGURE 22. The agent performance trained with DDQN on the 'Health Gathering' scenario.

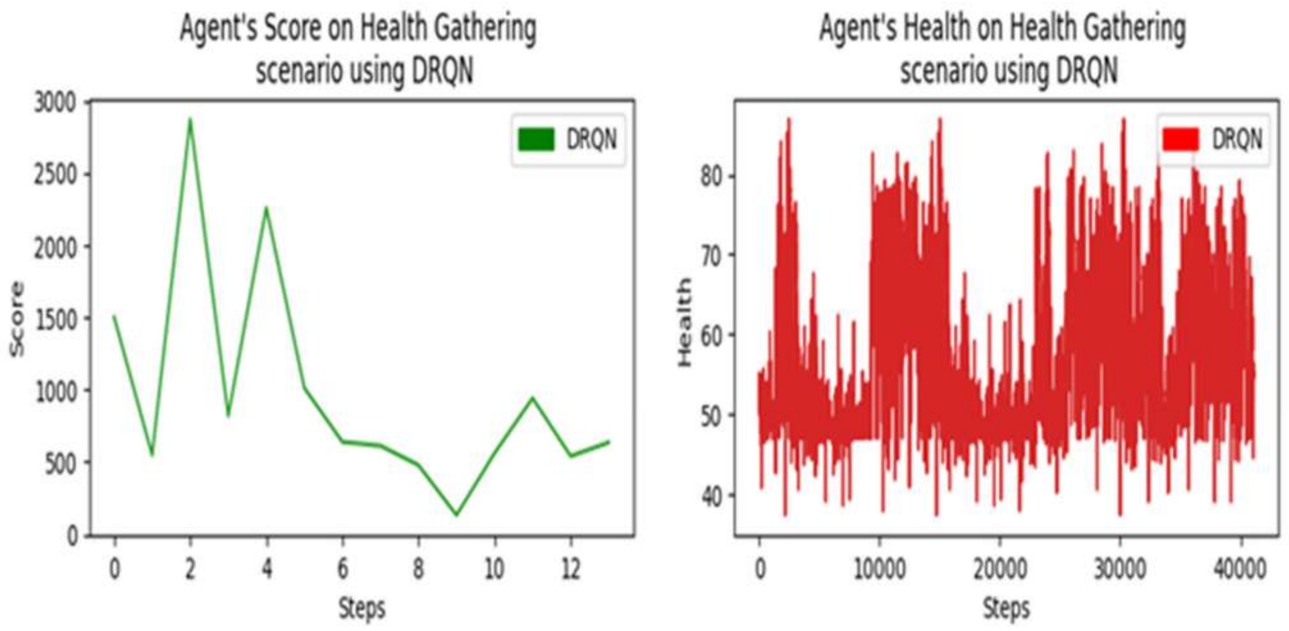


FIGURE 23. The agent act trained with DRQN on the 'Health Gathering' scenario.

E. COMPARISON WITH PREVIOUS STUDIES

Our proposed work is compared with the results of Niels Justesen et al. in article [54] and Paulo Bruno et al. in article [48], which are worthy examples for comparison. Table 3 shows the overall comparison. We have explored various algorithms, including A2C, A2C_LSTM, DDQN, DRQN, C51-DDQN, DFP, and REINFORCE. Justesen*et* al. focused on A2C and a variant with the rarity of events (RoE) approach. Paulo Bruno*et* al. used deep Q-Learning (DQN). Our work evaluated agent' performance in the 'Defend the Center,' 'Deadly Corridor,' and 'Health Gathering' scenarios. Justesen*et* al. used'Deathmatch,' 'Health Gathering,' 'Deadly Corridor,' and'My Way Hom' scenar-

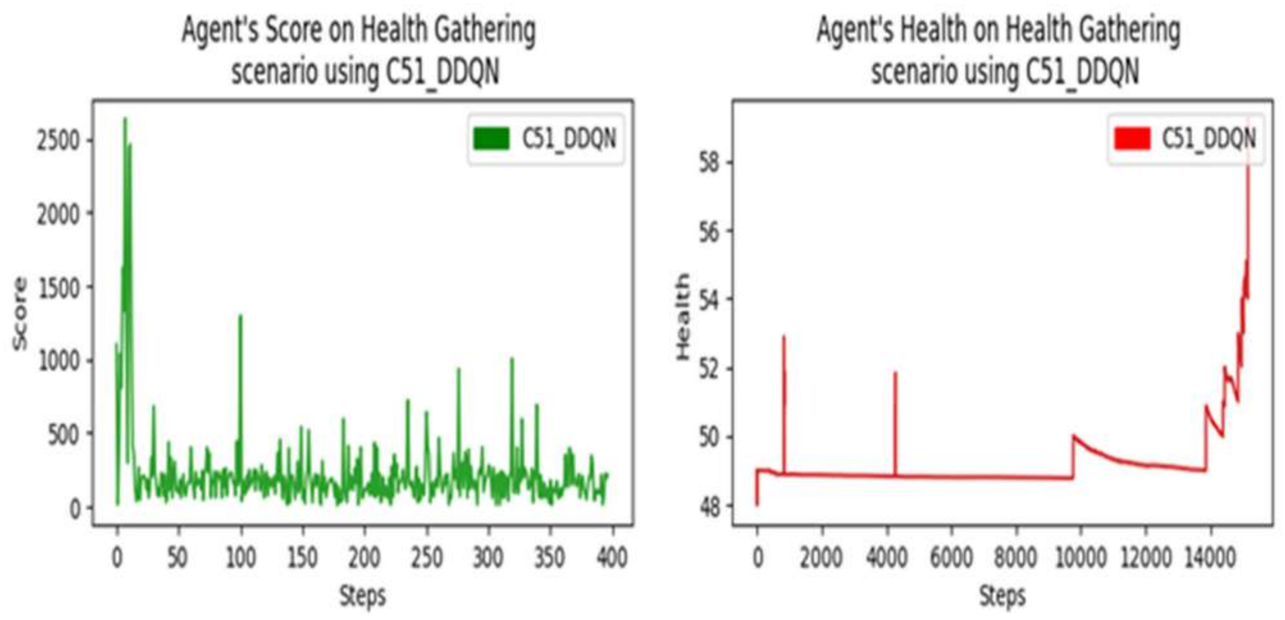


FIGURE 24. The agent performance trained with C51_DDQN on the 'Health Gathering' scenario.

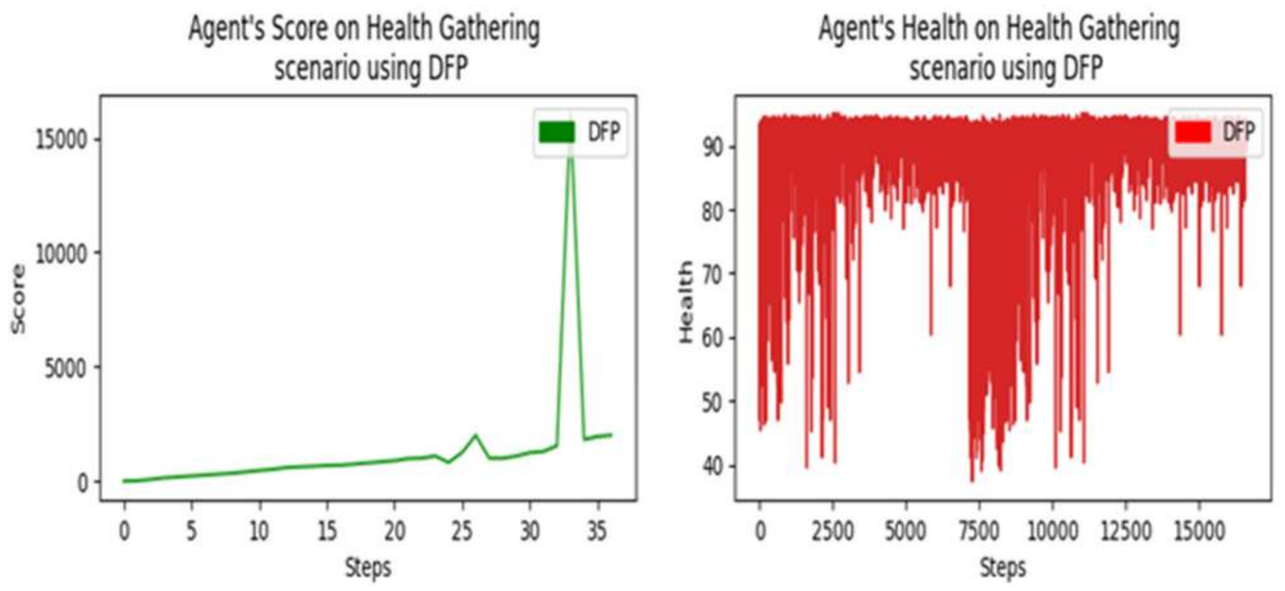


FIGURE 25. The agent performance trained with DFP on the 'Health Gathering' scenario.

ios. Paulo Bruno et al. tested 'Basic,' 'Defend the Line,' and'Medikits and Poiso' scenarios. Our work considered multiple metrics like ammo, kills, score, and health. Justesen et al. focused on the score, kills, medkit, and armour pickups. Paulo Bruno et al. used only the score as the evaluation metric. Our work employed experience replay to improve learning. Justesen et al. introduced RoE, a technique to enhance rare, valuable events' exploration and learning efficiency. Paulo Bruno et al. utilized experience replay and dropout for regularization. Our work used $\gamma = 0.99$, $\alpha = 0.0001$ to 0.01, batch size = 32. Justesen et al. employed $\gamma = 0.99$, $\alpha = 0.0007$, batch size = 64, and Bruno et al.: Set $\gamma = 0.99$, $\alpha = 0.0001$, batch size = 64. Our work and Justesen*et* al. used four frames per action, while Paulo Bruno *et* al. used eight frames per action. Our work converted images to 108×60 grayscale. Justesen*et* al. used 80×80 grayscale images, while Paulo Bruno et al. downscaled images to 64×48 grayscale. Our Work reported mean scores for each algorithm on each scenario, providing detailed performance comparisons. Justesen et al. provided mean scores for A2C and A2C+RoE in each scenario, along with standard deviations indicating variability. Paulo

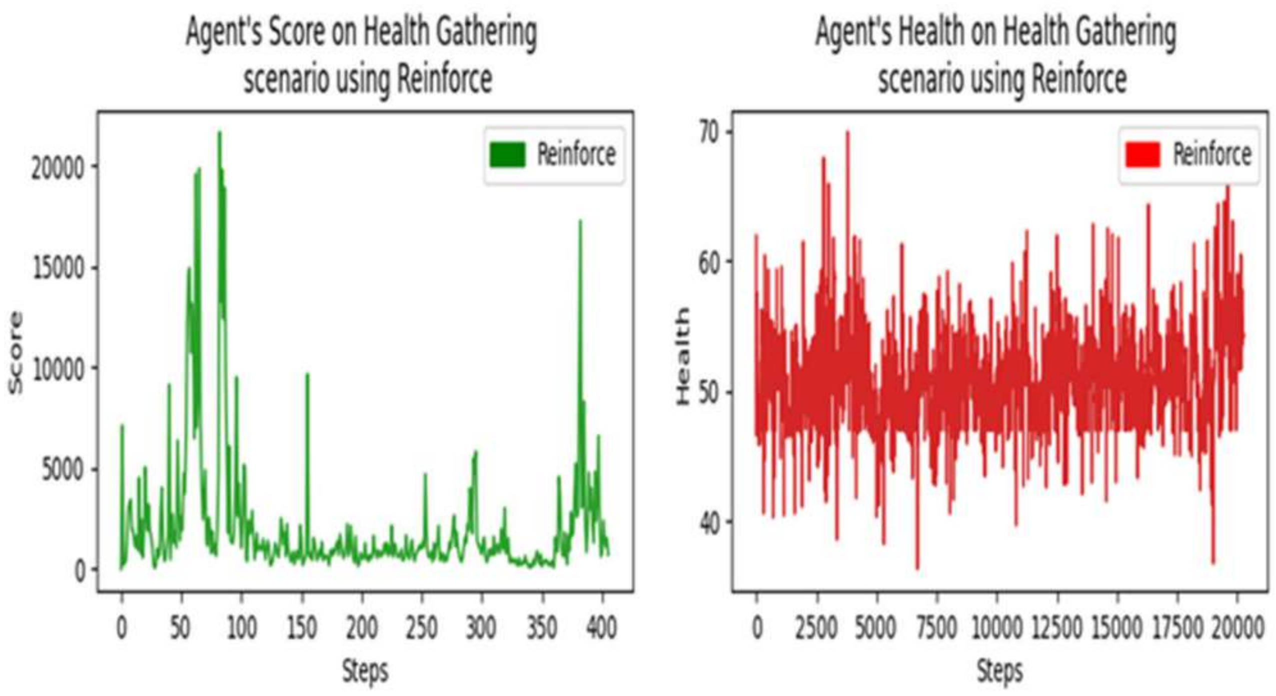


FIGURE 26. The agent performance trained with REINFORCE on the 'Health Gathering' scenario.

Bruno et al. reported single mean scores for each scenario without algorithm-specific comparisons.

A2C generally performed well across scenarios, while DRQN and REINFORCE showed mixed results. The deadly corridor scenario proved challenging for most algorithms, suggesting a need for further exploration of techniques for complex environments. Experience replay and RoE demonstrated potential for improving learning, but their impact varied across scenarios and algorithms. Tuning hyperparameters like 'γ' and'α' is essential for optimizing performance. The choice of image resolution and colour space can influence model performance.

V. COMPARATIVE ANALYSIS

In the challenging and demanding scenarios of VizDoom: 'Defend the Center,' 'Deadly Corridor,' and 'Health Gathering,' the evaluation of reinforcement learning algorithms—A2C, A2C_LSTM, DDQN, DRQN, C51-DDQN, DFP, and REINFORCE—shows appealing interplay of algorithmic attributes and adaptability to these challenging environments. A2C can be a viable option for VizDoom scenarios, but its performance depends on some specific characteristics of the scenario and the chosen hyperparameters. Its on-policy learning and actor-critic architecture offer advantages. However, limitations like high variance, sample bias, and adaptation difficulties require careful consideration and potential additional modifications for optimal performance in challenging VizDoom scenarios.

A2C-LSTM offers promising improvements over A2C for the challenging VizDoom scenarios, but its effectiveness somewhat depends on the nature of the scenario and careful implementation. Its strength in memory and handling delayed rewards are valuable, but challenges with complexity, vanishing gradients, and sample bias require consideration and potentially additional techniques for optimal performance.

DDQN, an evolution of the classic Q-learning algorithm, addresses overestimation bias in Q-value estimates by incorporating a target network. While DDQN's value-based approach equips it to handle the complexities of the scenarios by estimating expected future rewards for actions, it deals with exploration challenges, employing epsilon-greedy exploration strategies. DDQN falls short of efficiently navigating the vast action spaces inherent in VizDoom. Furthermore, DDQN's efficacy often relies on meticulous hyperparameter tuning, and achieving convergence necessitates substantial effort.

DRQN's RNN architecture captures temporal dependencies and learns from past experiences, mitigating sample bias and helping agents adapt to changing aspects of a scenario. However, recurrent connections amplify minor errors over time, leading to unstable training and unpredictable behaviour if not appropriately controlled.

C51-DDQN builds upon DDQN by introducing Q-value distributions, affording a more shading representation of uncertainty in estimates. This feature positions C51-DDQN to acquire the variability in outcomes more effectively, a particularly advantageous characteristic when facing stochastic environments like VizDoom. In the scenarios characterized by unpredictable enemy behaviour and weapon accuracy, C51-DDQN's ability to model uncertainty could contribute

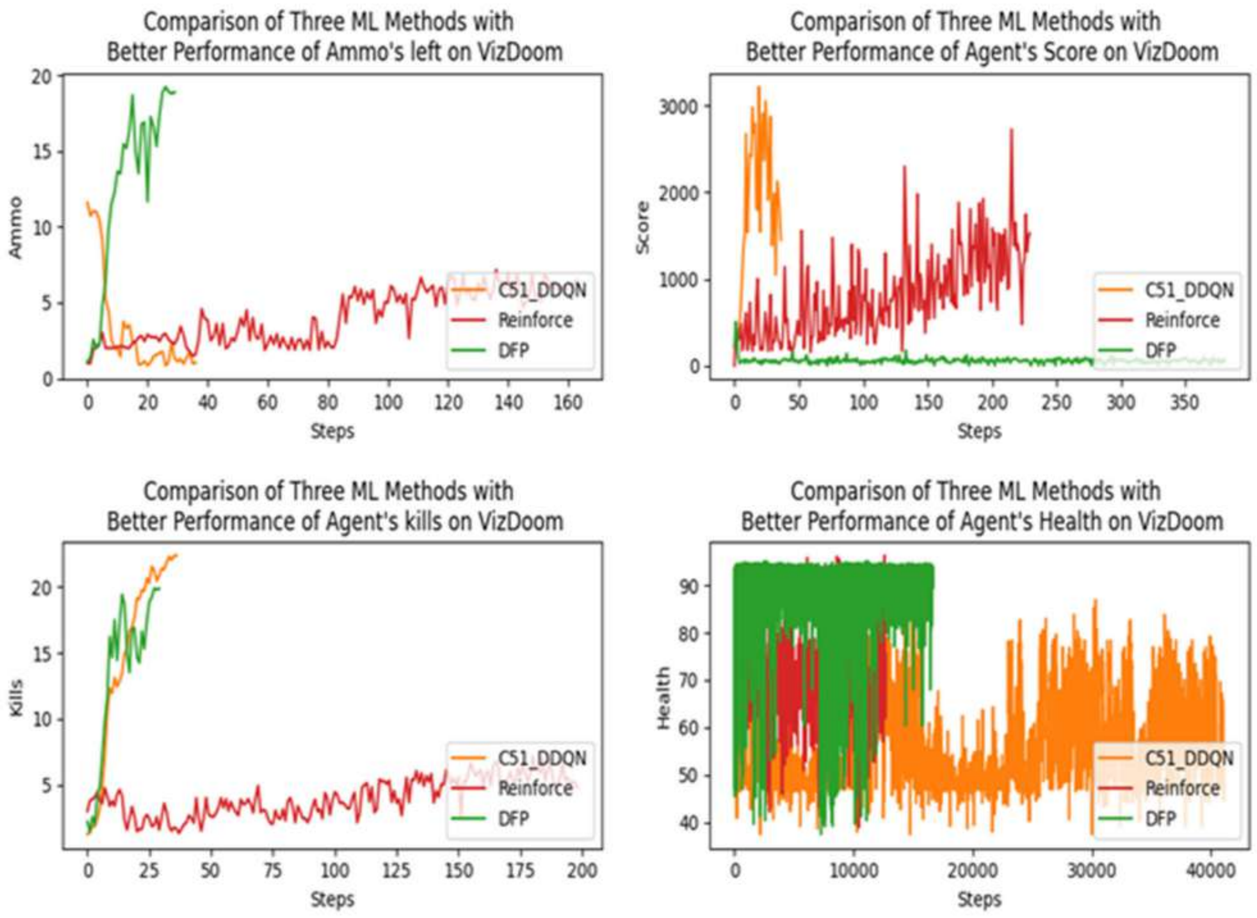


FIGURE 27. **Comparison of the three algorithms that performed well on the VizDoom Scenarios**

to robust learning. However, this approach introduces computational complexity as the algorithm must learn probability distributions over Q-values. While it may need more demanding training, the potential benefits of enhanced uncertainty modelling make C51-DDQN a compelling candidate, particularly in scenarios with sparse rewards and exploration hurdles.

DFP, a hybrid model that blends model-free and model-based learning, emerges as a promising approach in VizDoom scenarios. The model-based aspect of DFP exploits learned environment models to simulate possible future outcomes, mitigating the limitations of pure model-free methods in sparse-reward setups. This capability holds significant promise for addressing the challenge of delayed rewards in the scenarios, potentially accelerating learning and promoting more efficient exploration. However, DFP's success hinges on the quality of the learned model, necessitating extensive training data and hyperparameter tuning. The combination of model-free and model-based components makes DFP a valuable means for navigating the complexities of VizDoom.

In contrast, REINFORCE diverges from the value-based methods by adopting a policy-centric approach. Rather than estimating Q-values, REINFORCE directly learns policies mapping states to actions, rendering it less dependent on precise Q-value approximations. These characteristics position REINFORCE as a potential solution to sparse reward scenarios, as it tries to optimize policies without relying on value estimation. However, policy-based methods necessitate careful exploration strategy and learning rate adjustments to strike an effective balance between exploration and exploitation, especially in the high-dimensional state spaces characteristic of VizDoom. Additionally, the policy gradients characteristic of REINFORCE may introduce challenges in managing complex state spaces, demanding experimentation with different architectures and training techniques.

Stochasticity is a dominant feature in VizDoom scenarios, introducing unpredictability in enemy behaviour, weapon accuracy, and environmental dynamics. DDQN and C51-DDQN may demand supplementary techniques such as exploration Strategies, quantile regression, risk-averse Q-learning, and prioritized experience replay to accommodate this stochasticity effectively, as their reliance on value estimation can lead to suboptimal policies in the presence of uncertainty. The effectiveness of these techniques depends on the specific scenario characteristics and hyperparameter

TABLE 3. Comparison of our work with other related works in the literature.

Attribute/Factor/Feature	**Our Work**	**Niels Justesen *et al.* Work [55]**	**Paulo Bruno *et al.* Work [56]**
Algorithms	A2C, A2C_LSTM, DDQN, DRQN, C51-DDQN, DFP, REINFORCE	A2C, A2C+RoE	Deep Q-Learning
Scenarios	Defend the Center, Deadly Corridor, Health Gathering	Deathmatch, Health Gathering, Deadly Corridor, My Way Home	Basic, Defend the Line, Medikits, and Poison
Metric/Variables	Ammo, Kills, Score, Health	Score, kills, medkit & armor pickups	Score
Technique/Approach	Experience replay	Rarity of Events (RoE)	Experience Replay, Dropout
Available Buttons/Actions:	**Defend the Center**: turn left, turn right, attack. **Deadly Corridor**: move left, move right, attack, move forward, move backward, turn left, turn right. **Health Gathering**: turn left, turn right, and move forward.	**Deathmatch**: 20 different actions. **Health Gathering**: turn left, turn right, and move forward. **Deadly Corridor**: move left, move right, attack, move forward, move backward, turn left, turn right. **My Way Home**: turn left, turn right, move forward, move left, move right.	**Basic**: move left, move right, and shoot **Defend the Line**: turn left, turn right, and shoot. **Medikits and Poisons**: turn left, turn right, and move forward.
Hyperparameters	γ=0.99, α=0.0001 to 0.01, Batch size=32	γ=0.99, α=0.0007, Batch size=64	γ=0.99, α=0.0001, Batch size=64, ε-greedy policy= 1.0 to 0.1
Frames per action	4	4	8
Colour to Gray Scale	108x60	80×80	64x48
Mean Score on each scenario	**DTC**: A2C=~2500, A2C_LSTM=~600, DDQN=~1000, DRQN=~210 C51_DDQN=~1630, DFP=~95, REINFORCE=~545 **DC**: A2C=~220, A2C_LSTM=~42, DDQN=~270, DRQN=~654, C51_DDQN=~1120, DFP=~92, REINFORCE=~234 **HG**: A2C=~441, A2C_LSTM=~690, DDQN=~1285, DRQN=~915 C51_DDQN=~225, DFP=~240, REINFORCE=~440	**Deathmatch:** A2C=4611±2595, A2C+RoE= 4062±2442. **Health Gathering**: A2C= 399±107, A2C+RoE= 1261±533. **Deadly Corridor**: A2C= 0.00±0.0, A2C+RoE= 40±49. **My Way Home**: A2C= 96.69±0.12, A2C+RoE= 97.89±0.01.	**Basic**: 78. **Defend the Line**: 12. **Medikits and Poisons**: 518.47.

tuning. Epsilon-greedy is simple but might need further exploration strategies like upper confidence bound (UCB) in dynamic environments. Distributional learning offers better robustness against stochasticity, while risk-averse Q-learning can sacrifice potential rewards for stability. Prioritized experience replay can accelerate learning in stochastic environments, and multi-agent approaches can enhance adaptation and robustness. DFP's model-based facet naturally adapts to stochastic environments by considering multiple trajectories and outcomes. REINFORCE, built upon policy optimization, is intrinsically suited to handle stochasticity, offering an inherent advantage in such dynamic settings. Training stability emerges as a shared challenge, with DDQN and C51-DDQN known for their susceptibility to training instabilities such as divergence or slow convergence. While target networks somewhat mitigate these issues, they do not eliminate them. DFP, thanks to its model-based component, may adopt more stable training by providing a structured learning signal.

VI. CONCLUSION AND FUTURE WORK

Training agents capable of generalizing across different scenarios and environments remains a formidable challenge in reinforcement learning. This study introduced and compared multiple agents for excelling in the 3D first-person shooter game Doom. These agents demonstrated skills across various VizDoom scenarios, mimicking human-like behaviour. Moreover, the VizDoom Game AI platform proved the effectiveness of different reinforcement learning algorithms in navigating complex 3D multi-agent scenarios without predefined instructions or scripts, highlighting their potential for creating competent in-game adversaries.

Furthermore, the research found that transforming the reinforcement learning problem into a supervised learning paradigm, such as REINFORCE, yields improved performance and faster training, mainly when the scenario (maps) offers rich and temporally dense measurement signals. The experiments indicated that more extensive measurements result in better outcomes, allowing models to make more comprehensive predictions, akin to how supporting tasks enhance deep learning vision classifiers, as demonstrated by DFP.

In conclusion, this study suggests that C51-DDQN, DFP, and REINFORCE, reinforcement learning algorithms, can effectively serve as the primary training algorithms for agents, surpassing others in performance. These experiments signify a new era in artificial intelligence, showcasing the potential of learning from visual information, particularly in flexible and dynamic scenarios like VizDoom.

Looking ahead, we plan to use these agents as foundational architectures for tackling more challenging tasks. Future research endeavors will also explore various extensions to diversify learning behaviours in multiple ways. Besides, the future of applying reinforcement learning (RL) algorithms to first-person shooter (FPS) 3D games holds immense potential for pushing the boundaries of AI and gameplay. Here are some exciting potential avenues for future research:

A. INCREASING SAMPLE EFFICIENCY

1) Hierarchical RL: Decomposing complex tasks into smaller, manageable sub-tasks can significantly reduce the data needed for training.
2) Meta-learning and transfer learning: Transferring knowledge from previous experiences or training tasks to new scenarios can improve learning speed and generalizability.
3) Curriculum learning: Gradually increasing the difficulty of training environments can guide the agent toward efficient exploration and exploitation.

B. HANDLING UNCERTAINTY AND DYNAMIC ENVIRONMENTS

1) Stochastic policies: Learning policies that adapt to unpredictable situations and enemy behaviour can lead to more dynamic and realistic AI opponents.
2) Model-based RL: Building internal models of the game world can enable the agent to reason about future scenarios and plan more effectively.
3) Multi-agent learning: Training agents to cooperate and strategize in teams can create even more challenging and complex gameplay experiences.

C. IMPROVING HUMAN-AI INTERACTION

1) Explainable AI: Understanding the reasoning behind an agent's decisions can foster trust and engagement with the player.
2) Personalized player modeling: Adapting the AI's behaviour to the player's skill level and preferences can create a more enjoyable and balanced experience.
3) Natural language communication: Enabling AI agents to communicate and collaborate with players using natural language can create more profound and more immersive interactions.

D. EXPLORING NOVEL APPLICATIONS

1) Procedural content generation: RL agents can learn to generate new maps, weapons, and game modes, adding endless replayability and variety.
2) AI coaching and feedback: RL agents can be used to provide personalized coaching and feedback to players, helping them improve their skills and strategies.
3) AI development platforms: Creating user-friendly platforms for developers to utilize and customize RL algorithms for their games quickly can democratize AI in game development.

These are just a few examples, and the possibilities constantly evolve as research in RL and game AI progresses. By addressing the existing challenges and exploring these exciting avenues, RL can revolutionize how we design, play, and interact with FPS games in the future. It's important to note that ethical considerations should be integrated into future research. The impact of AI-powered violence and the potential for bias in learning algorithms need careful attention to ensure RL's responsible game development and application. By combining cutting-edge research with thoughtful design and ethical principles, we can ensure that the future of RL in FPS games is both technologically impressive and enjoyable, fair, and enriching for all players.

REFERENCES

[1] M. J. Mišic, Đ. M. Đurdevic, and M. V. Tomaševic, "Evolution and trends in GPU computing," in *Proc. 35th Int. Conv. MIPRO*, May 2012, pp. 289–294.

[2] N. Jouppi, C. Young, N. Patil, and D. Patterson, "Motivation for and evaluation of the first tensor processing unit," *IEEE Micro*, vol. 38, no. 3, pp. 10–19, May 2018.

[3] M. T. Hagan, H. B. Demuth, and M. Beale, *Neural Network Design*. Worcester, U.K.: PWS, 1997.

[4] K. Adil, F. Jiang, S. Liu, W. Jifara, Z. Tian, and Y. Fu, "State-of-the-art and open challenges in RTS game-AI and StarCraft," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 12, pp. 16–24, 2017, doi: 10.14569/ijacsa.2017.081203.

[5] 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*.

[6] L. Kaiser, M. Babaeizadeh, P. Milos, B. Osinski, R. H. Campbell, K. Czechowski, D. Erhan, C. Finn, P. Kozakowski, S. Levine, A. Mohiuddin, R. Sepassi, G. Tucker, and H. Michalewski, "Model-based reinforcement learning for Atari," 2019, *arXiv:1903.00374*.

[7] Y. Zhao et al., "Winning is not everything: Enhancing game development with intelligent agents," in *IEEE Trans. Games*, vol. 12, no. 2, pp. 199–212, Jun. 2020, doi: 10.1109/TG.2020.2990865.

[8] K. Adil, F. Jiang, S. Liu, A. Grigorev, B. B. Gupta, and S. Rho, "Training an agent for FPS doom game using visual reinforcement learning and VizDoom," *Int. J. Adv. Comput. Sci. Appl.*, vol. 8, no. 12, pp. 32–41, 2017, doi: 10.14569/ijacsa.2017.081205.

[9] A. Khan, M. Naeem, M. Z. Asghar, A. Ud Din, and A. Khan, "Playing first-person shooter games with machine learning techniques and methods using the VizDoom game-AI research platform," *Entertainment Comput.*, vol. 34, May 2020, Art. no. 100357, doi: 10.1016/j.entcom.2020.100357.

[10] 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)*, Sep. 2016, pp. 1–8.

[11] Z.-J. Pang, R.-Z. Liu, Z.-Y. Meng, Y. Zhang, Y. Yu, and T. Lu, "On reinforcement learning for a full-length game of StarCraft," in *Proc. AAAI Conf. Artif. Intell.*, 2019, pp. 4691–4698.

[12] K. Y. K. Adil, Y. Fu, F. Lou, W. Jifara, F. Jiang, and L. Shaohui, "A competitive combat strategy and tactics in RTS games AI and StarCraft," in *Proc. 18th Adv. Multimedia Inf. Process. (PCM)*, in Lecture Notes in Computer Science, vol. 10736. Cham, Switzerland: Springer, 2017, pp. 3–12, doi: 10.1007/978-3-319-77383-4_1.

[13] *Grand Theft Auto V Xbox One*, R. Games, New York, NY, USA, 2014.

[14] G. Ekaputra, C. Lim, and I. E. Kho, "Minecraft: A game as an education and scientific learning tool," in *Proc. Inf. Syst. Int. Conf. (ISICO)*, 2013, p. 1.

[15] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," 2016, *arXiv:1602.01783*.

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.

[17] I. Grondman, L. Busoniu, G. A. D. Lopes, and R. Babuska, "A survey of actor-critic reinforcement learning: Standard and natural policy gradients," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 6, pp. 1291–1307, Nov. 2012.

[18] C. Schulze and M. Schulze, "ViZDoom: DRQN with prioritized experience replay, double-Q learning and snapshot ensembling," in *Intelligent Systems and Applications*. Cham, Switzerland: Springer, 2019, pp. 1–17.

[19] M. Hausknecht and P. Stone, "Deep recurrent Q-learning for partially observable MDPs," 2017, *arXiv:1507.06527*.

[20] R. K. Brejl, H. Purwins, and H. Schoenau-Fog, "Exploring deep recurrent Q-learning for navigation in a 3D environment," *EAI Endorsed Trans. Creative Technol.*, vol. 5, no. 14, Jan. 2018, Art. no. 153641.

[21] W. Dabney, M. Rowland, M. Bellemare, and R. Munos, ''Distributional reinforcement learning with quantile regression,'' in *Proc. AAAI Conf. Artif. Intell.*, 2018, pp. 2892–2901, doi: 10.1609/aaai.v32i1.11791.

[22] A. Dosovitskiy and V. Koltun, ''Learning to act by predicting the future,'' 2016, *arXiv:1611.01779*.

[23] R. J. Williams, ''Simple statistical gradient-following algorithms for connectionist reinforcement learning,'' *Mach. Learn.*, vol. 8, nos. 3–4, pp. 229–256, May 1992, doi: 10.1007/bf00992696.

[24] A. Khan, F. Jiang, S. Liu, and I. Omara, ''Playing a FPS doom video game with deep visual reinforcement learning,'' *Autom. Control Comput. Sci.*, vol. 53, no. 3, pp. 214–222, May 2019, doi: 10.3103/s0146411619030052.

[25] G. Lample and D. S. Chaplot, ''Playing FPS games with deep reinforcement learning,'' in *Proc. AAAI Conf. Artif. Intell.*, Feb. 2017, vol. 31, no. 1, pp. 2140–2146, doi: 10.1609/aaai.v31i1.10827.

[26] Y. Sun, A. Khan, K. Yang, J. Feng, and S. Liu, ''Playing first-person-shooter games with A3C-anticipator network based agents using reinforcement learning,'' in *Proc. 5th Int. Conf. Artif. Intell. Secur. (ICAIS)*, New York, NY, USA, 2019, pp. 463–475, doi: 10.1007/978-3-030-24268-8_43.

[27] A. M. K. A. Khan, M. Z. Asghar, M. Naeem, and A. U. Din, ''Playing first-person perspective games with deep reinforcement learning using the state-of-the-art game-AI research platforms,'' in *Deep Learning for Unmanned Systems*, vol. 984, A. T. A. A. Koubaa, Ed. Cham, Switzerland: Springer, 2021, pp. 635–667.

[28] M. N. A. Khan, A. M. Khattak, M. Z. Asghar, and A. H. Malik, ''Playing doom with anticipator-A3C based agents using deep reinforcement learning and the ViZDoom game-AI research platform,'' in *Deep Learning for Unmanned Systems*, vol. 984, A. T. A. A. Koubaa, Ed. Cham, Switzerland: Springer, 2021, pp. 503–562.

[29] L. P. Kaelbling, M. L. Littman, and A. W. Moore, ''Reinforcement learning: A survey,'' *J. Artif. Intell. Res.*, vol. 4, pp. 237–285, May 1996.

[30] Y. Li, ''Deep reinforcement learning: An overview,'' 2017, *arXiv:1701.07274*.

[31] J. Z. Chen, ''Reinforcement learning generalization with surprise minimization,'' 2020, *arXiv:2004.12399*.

[32] Y.-H. Wang, T.-H.-S. Li, and C.-J. Lin, ''Backward Q-learning: The combination of Sarsa algorithm and Q-learning,'' *Eng. Appl. Artif. Intell.*, vol. 26, no. 9, pp. 2184–2193, Oct. 2013, doi: 10.1016/j.engappai.2013.06.016.

[33] F. Guihery, G. Bossert, D. Crémilleux, O. Tétard, B. Gigodeaux, and É. Klein, ''Reinforced autonomous agents with attack-defense exercises in realistic environments,'' in *Proc. 28th C&ESAR*, 2021, p. 191.

[34] A. Kumar, ''Playing pong using Q-learning,'' M.S. thesis, Dept. Comput. Sci., West Chester Univ. West Chester, Pennsylvania, PA, USA, 2021.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, ''Human-level control through deep reinforcement learning,'' *Nature*, vol. 518, no. 7540, pp. 529–533, Feb. 2015, doi: 10.1038/nature14236.

[36] M. Roderick, J. MacGlashan, and S. Tellex, ''Implementing the deep Q-network,'' 2017, *arXiv:1711.07478*.

[37] M. Sewak, ''Deep Q-network (DQN), double DQN, and dueling DQN,'' in *Deep Reinforcement Learning*. Singapore: Springer, 2019, pp. 95–108.

[38] G. Liu, W. Deng, X. Xie, L. Huang, and H. Tang, ''Human-level control through directly-trained deep spiking Q-networks,'' 2021, *arXiv:2201.07211*.

[39] A. Khan, J. Feng, S. Liu, and M. Z. Asghar, ''Optimal skipping rates: Training agents with fine-grained control using deep reinforcement learning,'' *J. Robot.*, vol. 2019, pp. 1–10, Mar. 2019, doi: 10.1155/2019/2970408.

[40] M. Abadi et al., ''TensorFlow: Large-scale machine learning on heterogeneous distributed systems,'' 2016, *arXiv:1603.04467*.

[41] N. Ketkar, ''Introduction to Keras,'' in *Deep Learning With Python*. Berkeley, CA, USA: Apress, 2017, pp. 95–109.

[42] G. a. A. K. Bradski, ''OpenCV,'' *Dr. Dobb's J. Softw. Tools*, vol. 120, pp. 122–125, Jan. 2000.

[43] S. Huang, A. Kanervisto, A. Raffin, W. Wang, S. Ontañón, and R. F. J. Dossa, ''A2C is a special case of PPO,'' 2022, *arXiv:2205.09123*.

[44] R. Li, C. Wang, Z. Zhao, R. Guo, and H. Zhang, ''The LSTM-based advantage actor-critic learning for resource management in network slicing with user mobility,'' *IEEE Commun. Lett.*, vol. 24, no. 9, pp. 2005–2009, Sep. 2020.

[45] M. Hessel, J. Modayil, H. Van Hasselt, T. Schaul, G. Ostrovski, W. Dabney, D. Horgan, B. Piot, M. Azar, and D. Silver, ''Rainbow: Combining improvements in deep reinforcement learning,'' in *Proc. AAAI Conf. Artif. Intell.*, Lousiana, LA, USA, 2018, pp. 3215–3222, doi: 10.1609/aaai.v32i1.11796.

[46] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, ''Prioritized experience replay,'' presented at the ICLR, Carolina, PR, USA, 2016, doi: 10.48550/arXiv.1511.05952.

[47] H. Van Hasselt, A. Guez, and D. Silver, ''Deep reinforcement learning with double Q-learning,'' in *Proc. 30th AAAI Conf. Artif. Intell.*, Phoenix Arizona, AZ, USA, 2016, pp. 2094–2100, doi: 10.1609/aaai.v30i1.10295.

[48] J. Ou, X. Guo, M. Zhu, and W. Lou, ''Autonomous quadrotor obstacle avoidance based on dueling double deep recurrent Q-learning with monocular vision,'' *Neurocomputing*, vol. 441, pp. 300–310, Jun. 2021.

[49] F. Moreno-Vera, ''Performing deep recurrent double Q-learning for Atari games,'' in *Proc. IEEE Latin Amer. Conf. Comput. Intell. (LA-CCI)*, Nov. 2019, pp. 1–4.

[50] N. K. Manaswi, ''RNN and LSTM,'' in *Deep Learning With Applications Using Python*. Berkeley, CA, USA: Springer, 2018, pp. 115–126.

[51] C. Schulze and M. Schulze, ''ViZDoom: DRQN with prioritized experience replay, double-Q learning, and snapshot ensembling,'' 2018, *arXiv:1801.01000*.

[52] M. G. Bellemare, W. Dabney, and R. Munos, ''A distributional perspective on reinforcement learning,'' in *Proc. Int. Conf. Mach. Learn.*, Sydney, NSW, Australia, 2017, pp. 449–458, doi: 10.48550/arXiv.1707.06887.

[53] S. Thoresen, ''Solving long-term planning problems with direct future prediction,'' M.S. thesis, Fac. Math. Natural Sci., Univ. Oslo, Oslo, Norway, 2021.

[54] N. Justesen and S. Risi, ''Automated curriculum learning by rewarding temporally rare events,'' in *Proc. IEEE Conf. Comput. Intell. Games (CIG)*, Maastricht, The Netherlands, Aug. 2018, pp. 1–8, doi: 10.1109/CIG.2018.8490448.

[55] P. B. S. Serafim, Y. L. B. Nogueira, C. Vidal, and J. Cavalcante-Neto, ''On the development of an autonomous agent for a 3D first-person shooter game using deep reinforcement learning,'' in *Proc. 16th Brazilian Symp. Comput. Games Digit. Entertainment (SBGames)*, Curitiba, Brazil, Nov. 2017, pp. 155–163, doi: 10.1109/SBGames.2017.00025.

ADIL KHAN received the M.S. degree from City University Peshawar, Khyber Pakhtunkhwa, Pakistan. He has over 12 years of university-level teaching, research, and laboratory experience. He has published many publications in top-tier academic journals and conferences. His research interests include machine learning, deep learning, game agents, and bioinformatics. He is a Reviewer of many reputed journals, such as IEEE Access and IEEE Transactions on Games.

ASGHAR ALI SHAH received the M.S. degree in information technology from IM|Sciences, Peshawar, Pakistan, in 2007, and the Ph.D. degree in computer science from the University of Management and Technology (UMT), Lahore, Pakistan, in July 2023. He has been an Assistant Professor with the Department of Computer Sciences, Bahria University, since February 2017. He has over 18 years of teaching experience at various prestigious universities and eight years of research experience. He has more than 35 research articles published in esteemed journals. His research interests include machine learning, deep learning, computer vision, network security, and bioinformatics.

LAL KHAN received the M.S. degree in computer science from the Federal Urdu University of Arts, Science, and Technology, Islamabad, in 2017, and the Ph.D. degree from the Department of Computer Science and Information Engineering, Chang Gung University, Taiwan. He is currently working as an Assistant Professor with the Department of Software Engineering, Foundation University Islamabad, Pakistan. His research interests include machine learning, deep learning, natural language processing (NLP), and speech recognition. He is working on an NLP task for resource-deprived languages and health informatics.

MUHAMMAD REHAN FAHEEM was born in Pakistan. He received the B.S.I.T. degree from the Islamia University of Bahawalpur, Pakistan, in 2012, the M.Phil. degree in computer science from NCBAE, Lahore, Pakistan, in 2016, and the Ph.D. degree from the University of Management and Technology, Lahore, Pakistan. He is with the Department of Computer Science, The Islamia University of Bahawalpur, Bahawalpur, Pakistan. He has strong analytical and problem-solving abilities. His research interests include the Internet of Things, the Web of Things, image processing, information retrieval, computer graphics, computer vision, graphics modelling, and machine learning.

MUHAMMAD NAEEM received the Ph.D. degree in computer science from Brock University, Canada. He taught for 34 years with the Department of Computer Science, University of Peshawar, Peshawar, Pakistan. He is currently a Professor in Canada. His research interests include pattern recognition, medical image analysis, machine learning, and deep learning. His current research interest includes artificial intelligence in games.

HSIEN-TSUNG CHANG received the M.S. and Ph.D. degrees from the Department of Computer Science and Information Engineering (CSIE), National Chung Cheng University, in July 2000 and July 2007, respectively. Following his academic pursuits, he joined as a Faculty Member of the Computer Science and the Information Engineering Department, Chang Gung University, where he has been a Full Professor. He is currently an esteemed Artificial Intelligence Research Center Member with Chang Gung University and holds a significant position with the Department of Physical Medicine and Rehabilitation, Chang Gung Memorial Hospital. He dedicates his services to the Bachelor Program in Artificial Intelligence, enriching the academic environment and advancing student knowledge and research. His extensive research interests include artificial intelligence in games (game AI), natural language processing, information retrieval, big data, web services, and search engines. He continues to explore and innovate in these realms, aiming to propel the fields forward. He is also the Director of the Web Information and Data Engineering Laboratory (WIDE Laboratory), overseeing various groundbreaking projects and research endeavors.