Shakir Jan
01-134202-062

Adnan Musa
01-134202-113

Enhancing Agent's Performance in VizDoom Using Reinforcement Learning

Bachelor of Science in Computer Science

Supervisor: Mr. Adil Khan

Department of Computer Science
Bahria University Islamabad

May 2024

Certificate

We accept the work contained in the report titled “Enhancing Agents Performance in VizDoom using reinforcement learning” written by Shakir Jan and Adnan Musa as a confirmation of the required standard for the partial fulfillment of the degree of Bachelor of Science in Computer Science.

_____	_____	_____
Supervisor	**Internal Examiner**	**External Examiner**
Name: _____	Name: _____	Name: _____
Date: _____	Date: _____	Date: _____

_____	_____
Project Coordinator	**Head of Department**
Name: _____	Name: _____
Date: _____	Date: _____

Abstract

An agent interacting with its surroundings to learn how to make decisions is called a reinforcement learner. Through reinforcement learning, the agent gradually gains the ability to behave in a way that maximizes its overall reward. Instead of receiving instructions from an instructor, it learns by trial-and-error process. VizDoom is an easy-to-use reinforcement learning-based research platform for Doom games. The task is to train agents in different VizDoom scenarios, including basic scenario, defend the center, and deadly corridor making decisions based on visual information from the screen.

This project compares three common reinforcement learning algorithms. Double Dueling Deep Q Learning Network (DDDQN), Advantage Actor Critic (A2C), and Proximal Policy Optimization (PPO). We examine each algorithm's effectiveness in enhancing agent performance in the VizDoom environment through testing and analysis. Each of the previously stated algorithms is used to train the agents in our study, and their performance is assessed using a range of measures, including average score, number of kills, and reward.

To further optimize each algorithm's performance, we have tried different values for tuning hyperparameters during our training process. During the training process, we changed critical hyperparameter values that influence the overall agent behavior and lead to its improvement.

In the context of improving agent performance in VizDoom, the study's conclusions offer insightful information about the advantages and disadvantages of PPO, A2C, and DQ. Furthermore, our insights into hyperparameter optimization clarify the significance of parameter adjustment to optimize the performance of reinforcement learning algorithms in difficult real-world scenarios.

The present study explores how reinforcement learning (RL), a type of AI training, can be better applied to video games. By understanding how RL works in games, researchers can develop better ways to train AI agents, making them smarter and more fun to play against.

Acknowledgment

We thank our supervisor Mr. Adil Khan for his continuous support and encouragement throughout our project. Moreover, we also thank the Center of Excellence in Artificial Intelligence at Bahria University Islamabad for providing us with remote GPU access for our model training during our agent training.

List of Tables

Table 2.1 Representation of DRL Methods 9
Table 2.2 Results of competition Track 1 2016 [18]. 10
Table 2.3 Results of competition Track 2 2016 [18]. 11
Table 2.4 Results of competition Track 1 2017 [18] 12
Table 2.5 Results of competition Track 2 2017 [18] 13
Table 2.6 Results of Track 1 Doom AI competition 2018 [19]. 14
Table 2.7 Results of Track 2 Doom AI competition 2018 [20]. 14
Table 3.1 Considered hyperparameter settings in the experiment. 21
Table 4.1 Comparison of our work with existing similar literature 33

List of Figures

Figure 1.1. Shows a VizDoom scenario. 1
Figure 2.1 Two hidden layers of a basic Artificial Neural Network 4
Figure 2.2 shows the basic structure of CNN. 5
Figure 2.3 How Feedback Loop Works in Deep Reinforcement Learning 6
Figure 2.4 Scenarios used in track 2 11
Figure 3.1 Scenario of basic in VizDoom. 19
Figure 3.2 Starting position of the agent. 20
Figure 3.3 Agent killing the opponent. 20
Figure 3.4 Scenario of Defend the Center in VizDoom. 20
Figure 4.1 Kill Counts on Defend the Center using PPO. 26
Figure 4.2 Agent's reward Mean on Defend the Center using PPO. 26
Figure 4.3 Ammo left on Defend the Center using PPO. 26
Figure 4.4 Average episode length in PPO-trained Defend. 26
Figure 4.5 Reward graph trained using DDDQN in defend the center scenario. 27
Figure 4.6 Ammo vs episodes graph using DDDQN in defend the center scenario. 27
Figure 4.7 Reward graph trained using DDDQN in defend the center scenario. 27
Figure 4.8 Training of loss graph using DDDQN in defend-the-center scenario. 27
Figure 4.9. Training of agents in defend the center using the A2C algorithm. 28
Figure 4.10 Total reward obtained using A2C algorithm in defend the center. 28
Figure 4.11 Kills using PPO in deadly corridor scenario. 29
Figure 4.12 Average reward graph during training using PPO in the deadly corridor scenario. 29
Figure 4.13 Average length mean over timestamps. 29
Figure 4.14 Average Cumulative Reward over episodes in deadly corridor using A2C algorithm. 30
Figure 4.15 Total reward obtained by the agent using the A2C algorithm in the deadly corridor. 30
Figure 4.16 Ammo left on Basic Scenario using PPO. 31
Figure 4.17 Kills on basic Scenario using PPO. 31
Figure 4.18 Average reward graph during training using PPO in the basic Scenario 31
Figure 4.19 Average episode length during training using PPO in a basic scenario. 31
Figure 4.20 Kills vs episodes graph using DDDQN in a basic scenario. 32
Figure 4.21 Ammo vs episodes graph using DDDQN in a basic scenario. 32
Figure 4.22 Reward graph trained using DDDQN in a basic scenario. 32
Figure 4.23 Loss graph trained using DDDQN in a basic scenario. 32

List of Abbreviations

A2C	Advantage Actor-Critic
A3C	Asynchronous Advantage Actor-Critic
ANN	Artificial Neural Network
AI	Artificial Intelligence
CNN	Convolutional Neural Networks
DDDQN	Double Dueling Deep Q-Network
DQN	Deep Q Network
DRL	Deep Reinforcement Learning
DL	Deep Learning
FPS	First Person Shooter
MDP	Markov Decision Process
ML	Machine Learning
NN	Neural Network
NPC	Non-Player Character
PPO	Proximal Policy Optimization
PG	Policy Gradient
RL	Reinforcement Learning
TRPO	Trust Region Policy Optimization

Table of Contents

1. Introduction 1
   1.1 Project Overview 1
   1.2 Relevance to Recent Research 1
   1.3 Project Objective 1
   1.4 Problem Description 2
   1.5 Proposed Solution 2
   1.6 Project Scope 2
   1.7 Methodology 3
   1.8 Pros of Reinforcement Learning for 3D shooter Games 3
2. Literature Review 4
   2.1 Background 4
   2.2 Artificial Neural Network 4
   2.3 Convolutional Neural Network 4
   2.4 Markov Decision Process (MDP) 5
   2.5 Deep Reinforcement learning 5
   2.6 On-policy Against Off-policy Algorithms 6
   2.7 Proximal Policy Optimization 6
   2.8 Advantage Actor-Critic 7
   2.9 Deep Q Network 8
   2.10 VizDoom 8
   2.11 Associated Works 9
   2.12 Table Representation of DRL Methods 9
   2.13 Doom AI Competitions 2016 10
      2.13.1 Rules of Doom AI Competitions 2016 10
      2.13.2 Tracks of Doom AI Competitions 2016 10
   2.14 Doom AI Competitions 2017 12
      2.14.1 Rules of Doom AI Competitions 2017 12
      2.14.2 Tracks of Doom AI Competitions 2017 12
   2.15 Doom AI Competitions 2018 13
      2.15.1 Tracks of Doom AI Competition 2018 14
   2.16 Applications Beyond VizDoom 15
3. Methodology and Experiments 17
   3.1 Methodology Diagram 17

3.2 Tools and Technologies used for training using PPO, A2C, and DDDQN. 18
3.3 Hyperparameters 19
3.4 Environment and Scenarios in VizDoom 19
3.5 Experimental Design and Environment 21
3.6 Data Collection and Reward Shaping 21
3.7 Reward Shaping 22
3.8 Implementation of PPO Algorithm 22
3.8.1 Mathematical Form 22
3.9 Implementation of DDDQN (Double Dueling Deep Q-Network) Algorithm 23
3.9.1 Mathematical Form 24
3.10 Implementation of A2C (Advantage Actor Critic) Algorithm 24
3.10.1 Mathematical Form 24
3.11 Inputs for A2C algorithm 24
3.12 Hyper Parameters for A2C algorithm 25
4. Results and Comparison 26
4.1 Defend the Center 26
4.2 Deadly Corridor 29
4.3 Basic Scenario 30
4.4 Comparison with Prior Works 33
5. Conclusion 34
6. Future Work 35
7. References 36

1. Introduction

1.1 Project Overview

The ability of AI agents to make intelligent decisions in diverse and challenging environments is crucial for various applications. This project focuses on enhancing the performance of AI agents within VizDoom, a popular platform that integrates AI with Doom (a first-person shooter game) [1]. By training agents to navigate them in different VizDoom scenarios within the Doom game, we aim to demonstrate their adaptability and problem-solving capabilities in different VizDoom scenarios. Through extensive training and evaluation on various VizDoom maps, our agents achieved significant improvement in navigation, decision-making, shooting, and enemy combat. This project demonstrates the potential of reinforcement learning in different game scenarios for enhancing AI agent performance. Figure 1.1 below shows a VizDoom scenario known as Deadly Corridor with the agent aiming at the enemy.

Figure 1.1. Shows a VizDoom scenario.

1.2 Relevance to Recent Research

Research on deep reinforcement learning is increasing in modern times and has led to numerous applications in video games. The first achievement that pushed this wave was when Mnih et al. [2] showcased their DRL model that used raw video data and efficiently learned policies. Now research is proceeding in all areas including first-person shooters to real-time strategy games [3]. Take game testing where DRL has proven to increase test coverage [4]. Another is to create competitive opponents for human players [5].

1.3 Project Objective

The project aims to significantly improve the performance of AI agents on the VizDoom platform using reinforcement learning algorithms, which include PPO (Proximal Policy Optimization), A2C, and (Double Dueling Deep Q-Learning Network) DDDQN. The project will focus on the performance evaluation of agents, navigating them through various scenarios, avoiding obstacles, and resource management on different scenarios of VizDoom. This project has potential for real-world applications, such as intelligent autonomous systems and robotics capabilities, and contributes significantly to AI research by pushing the boundaries of reinforcement learning in complex game environments.

1.4 Problem Description

The project tackles a significant challenge in the realm of artificial intelligence (AI) and gaming. VizDoom is a platform that merges AI with the classic video game "Doom" offering a sophisticated environment for training AI agents. Within this framework, our project aims to optimize the performance of these AI agents, specifically focusing on their decision-making abilities regarding movement, ammo preservation, and shooting.

The optimization of agent performance is crucial for several reasons. Firstly, in dynamic and unpredictable environments, agents must make rapid and accurate decisions to navigate the game successfully. Movement decisions determine the agent's ability to explore the environment efficiently while avoiding obstacles and threats. Effective ammo preservation ensures that the agent can sustain itself in prolonged engagements, while strategic shooting is essential for eliminating adversaries and achieving game objectives.

RL serves as the backbone of the approach, enabling agents to learn and improve their decision-making skills through trial and error. RL algorithms allow agents to interact with the game environment, receive feedback in the form of rewards or penalties based on their actions, and adjust their strategies accordingly to maximize cumulative rewards over time.

By leveraging RL techniques, our project aims to train agents that not only perform well in specific scenarios but also generalize their learning to novel situations within the VizDoom environment. This generalization is crucial for ensuring that agents can adapt to various challenges and environments they may encounter during gameplay.

1.5 Proposed Solution

The reinforcement learning algorithm is used to train agents in different scenarios of VizDoom. Scenarios like basic, defend the center, and deadly corridor provide the agents with specific challenges to overcome and controlled environments for the agents to learn and adapt. Agents' performance is iteratively optimized by the decision-making process through playing each scenario repeatedly. We aim to significantly enhance the agent's performance making it more intelligent, efficient, and better at winning in VizDoom scenarios. We aim to train intelligent agents to navigate diverse scenarios, prioritize ammo conservation, maximize health, and employ strategic combat to master each VizDoom scenario efficiently. By employing reinforcement learning and designing suitable reward systems, agents are empowered to become resourceful and tactical masters, proving the effectiveness of reinforcement learning in adapting AI that excels in different VizDoom scenarios.

1.6 Project Scope

a. What is included in the project?

a) Implementation of reinforcement learning algorithms to train the agents on existing game scenarios.
b) Training of the agents in different VizDoom scenarios to adhere to specific challenges of the VizDoom environment.
c) Evaluation of the agent's performance based on its training.
d) Analysis of the agent's performance to identify areas for improvement.

b. Assumptions

a) The VizDoom environment is available and working properly.
b) The necessary hardware and software resources are available to train and evaluate the agents.
c) The agents are capable of learning from their experiences, and mistakes and improving agents' performance over time.

c. Limitations

a) The performance and training of the agents depend upon how efficiently the algorithms are used.
b) The agents may not be able to generalize their performance to new and unseen VizDoom scenarios.

1.7 Methodology

It employs an experimental approach (agent learning is based on trial and error rather than being explicitly programmed), where agents are trained using PPO, DDDQN, and A2C algorithms. We have implemented 3 different scenarios (basic, defend the center, and deadly corridor) to train the agents in a way that they learn to navigate, anticipate, and attack. It can move around and try different things, like moving left, moving right, attacking, moving forward, moving backward, turning left, and turning right. We have implemented reward shaping to get the desired behavior out of the agents. Data is collected by interacting with the environment. Reinforcement learning algorithms use this data to learn and improve the agent's policy, which maps states to actions. The goal is to maximize cumulative rewards over time, leading to better performance of the agents. Further, we have analyzed tensor board charts and have implemented matplotlib graphs to keep track of the important hyperparameters and values that affect the training process leading to a more balanced and effective approach.

1.8 Pros of Reinforcement Learning for 3D shooter Games

a. Being proficient in challenging tasks
Reinforcement learning is good at learning complex strategies plus dealing with changing environments, which in turn allows AI agents in First-person shooter games to be experts at skills like target prioritization, flanking, and cover-related tasks.

b. Generalization and adaptability
These algorithms can adapt to different scenarios, opponent actions, and even weapon configurations without any instructions and this makes it significant for diverse gameplays.

c. Emergent behavior
In FPS games, RL agents can astonish the developers with their ability to uncover unforeseen strategies and tactics, resulting in the emergence of more dynamic and inventive AI opponents.

d. Potential for competitive AI
Through continuous research and development, AI-driven reinforcement learning holds the potential to surpass human players in FPS games, introducing increasingly formidable and thrilling virtual adversaries.

2. Literature Review

2.1 Background

This section provides the foundational components utilized in our study. It encompasses the core principles of reinforcement learning alongside its interconnected neural networks. Furthermore, it provides a concise overview of the three distinct reinforcement learning algorithms employed within the experiment. Additionally, it delves into the concept of hyperparameter optimization and outlines the proposed training environment. Lastly, it explores relevant research studies associated with the subject matter under investigation.

2.2 Artificial Neural Network

Artificial Neural Networks (ANNs) are widely utilized structures, drawing inspiration from neurons in the brain. Recent advancements in deep learning encompass ANNs with numerous hidden layers, a characteristic relevant to reinforcement learning. Illustrated in Figure 2.1 is a fundamental feed-forward ANN, where signals propagate solely to neurons in the subsequent layer. Contrarily, a recurrent Neural Network may feature signals moving backward or within the same layer. Each connection within the network is associated with a real value, dictating the activation level of neurons. Activation is determined by a semi-linear combination of input values from the preceding layer [6].

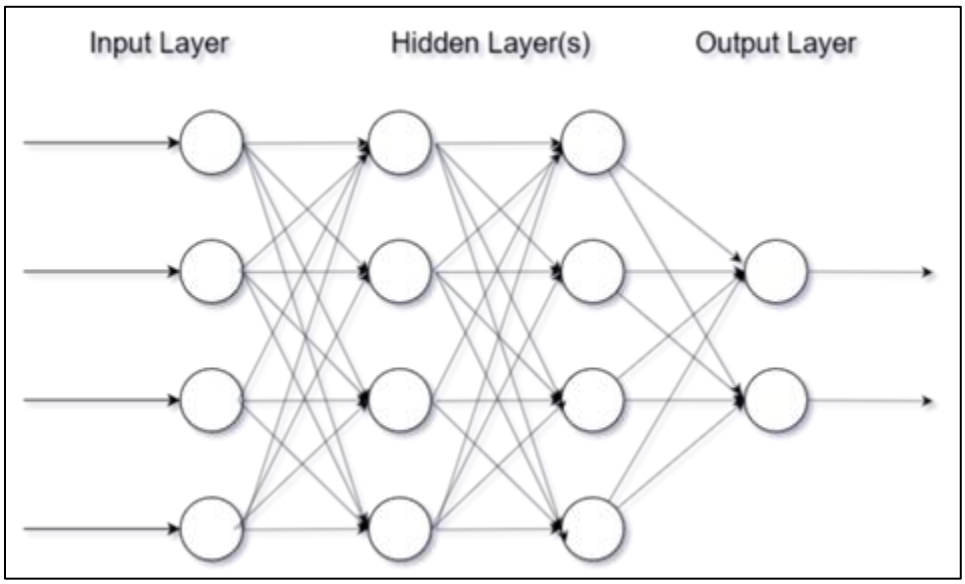


Figure 2.1 Two hidden layers of a basic Artificial Neural Network

2.3 Convolutional Neural Network

A convolutional neural network (CNN) is a category of AI networks comprising multiple layers in deep learning, primarily utilized for image visualization. It stands as one of the most effective and robust tools in handling extensive datasets, particularly in research domains like image recognition [7]. CNNs are widely favored in deep learning, deriving their name from the mathematical operation of convolution between matrices. Comprised of various layers including convolutional, non-linearity, pooling, and fully connected layers, CNNs exhibit remarkable capabilities due to their parameterized convolutional and fully connected layers. Unlike these, the pooling and non-linearity layers do not possess parameters. CNNs present significant advantages in machine learning, particularly for tasks such as image recognition and

natural language processing, among various other application domains. Figure 2.2 shows the basic working of a CNN where image processing is shown in action with the image going through convolutional layers and then at the end the possible actions (output). A basic CNN works by taking input in the form of an image or any other data form, passes it through several convolutional layers and then narrows down on one possible action that our agent takes in the end.

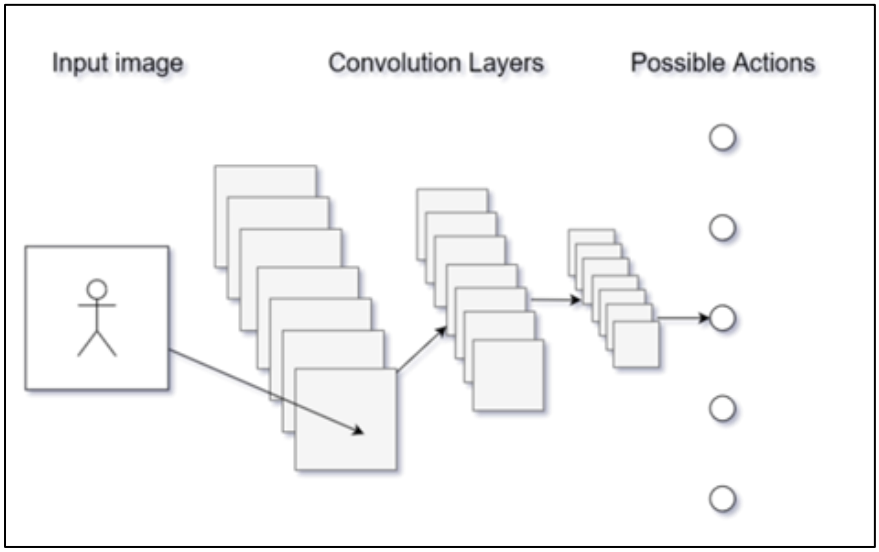


Figure 2.2 shows the basic structure of CNN.

2.4 Markov Decision Process (MDP)

As demonstrated by Markov Chains, Markov Property asserts that the future depends only on the current situation. This is expanded upon by Markov Decision Processes (MDPs), which include actions, rewards, and a discount factor γ. Future prizes are scaled back by this factor. MDPs use action-value pairs (q_π (s, a)) mapping. Maximizing q_* (s, a) where q_* is the largest expected reward for a particular state action pair is the goal of an optimum policy, $\pi_*(a|s)$. As a result, for each state, the best course of action is chosen from among those with the highest q_* values.

2.5 Deep Reinforcement learning

In machine learning, establishing a strategic approach is essential, with deep reinforcement learning being one prominent method [8], alongside supervised and unsupervised learning. Reinforcement learning revolves around enabling an agent to learn through interaction with its environment, where it discerns between actions deemed favorable or unfavorable, with encouragement towards favorable actions. Figure 2.3 below reflects on this as it shows how the basic feedback loop works in deep reinforcement learning. Through reinforcement learning, an agent accrues experience and leverages it to optimize designated objectives within its environment, facilitated by increasing reward magnitudes. This approach has gained traction in addressing intricate sequential decision-making challenges, evident in environments like chess or modern games such as StarCraft 2.

Machine learning furnishes automated methods for pattern detection, subsequently facilitating task execution based on the identified patterns. Deep learning employs neural networks comprising successive layers of nonlinear transformations, each contributing to varying levels of abstraction. The diverse range of neural network layer types in modern applications each possesses distinct advantages, limitations, and applications. Selection of the appropriate type depends on the desired objectives.

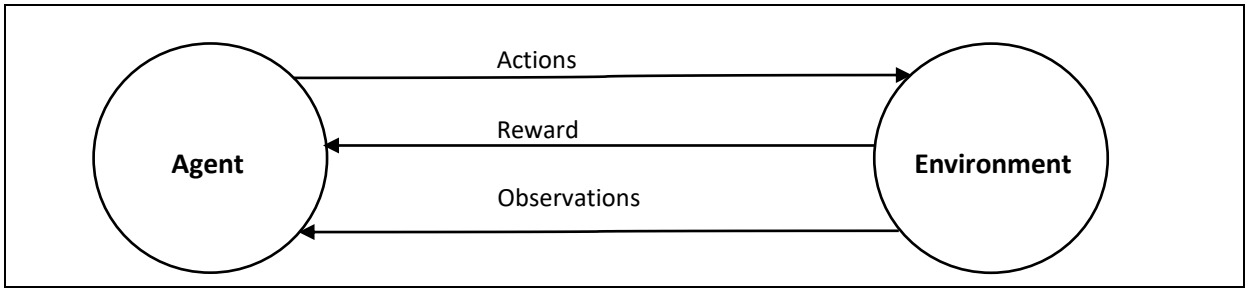


Figure 2.3 How Feedback Loop Works in Deep Reinforcement Learning

Figure 2.3 illustrates the agent's actions based on observations of its surroundings, the environment changes because of these actions, the agent gets rewards and observes the altered state, which helps it decide what to do next. The agent eventually learns to make decisions that maximize its cumulative rewards through this iterative process of action selection and feedback, which enhances its performance in the environment.

2.6 On-policy Against Off-policy Algorithms

Two on-policy algorithms, A2C and PPO, as well as one off-policy algorithm, DQN, are the subjects of this study's investigation. In contrast to off-policy algorithms, on-policy reinforcement algorithms have more sampling complexity, whereas off-policy techniques have more optimization difficulties [9]. Agents can learn optimal policies based on performance feedback from their interactions with the environment by using reinforcement learning techniques.

There are two categories of policies: off-policy and on-policy. The agent uses a batch of data corresponding to its current policy in on-policy algorithms. On the other hand, off-policy algorithms use information from past policies to make updates to the present policy.

2.7 Proximal Policy Optimization

In the dynamic field of ML, continual advancements introduce novel algorithms to address challenges in reinforcement learning. One such relatively recent algorithm is Proximal Policy Optimization (PPO), initially introduced by researchers at OpenAI in 2017 [10]. PPO operates as a policy gradient algorithm, oscillating between gathering information from the surroundings and maximizing a goal function via stochastic gradient ascent. Notably, PPO offers a simpler implementation compared to other policy gradient algorithms like trust region policy optimization.

PPO boasts superior sample complexity when compared to algorithms such as A2C (Advantage Actor-Critic), demonstrating enhanced performance in benchmark environments such as Atari games, striking a balance between sample complexity and implementation ease. Training in an on-policy manner, PPO employs a stochastic policy, exploring via sampled actions determined by its most recent policy iteration. Although initially characterized by randomness, PPO gradually mitigates randomness with each update, encouraging the utilization of previously discovered rewards.

The variant of PPO employed in this study utilizes a technique known as clipping. This technique ensures that policy updates do not deviate significantly from prior policies, constraining the objective interval to a narrow range. Consequently, the probability ratio is clipped, negating the incentive for the algorithm to stray beyond the objective goal in cases where the objective deteriorates while disregarding the clipping when improvements are observed.

The pseudo below is taken from [11].

Algorithm 1 PPO-Clip - pseudo code

Input: initial policy parameters θ_0, initial value function parameters ϕ_0
for $k = 0, 1, 2, \ldots$ **do**
  Collect set of trajectories $\mathcal{D}_K = \{\tau_i\}$ by running policy $\pi_k = \pi(\theta_k)$ in the environment.
  Compute rewards-to-go $\hat{R}_t$.
  Compute advantage estimates, $\hat{A}_t$ (using any method of advantage estimation) based on the current value of function $V_{\phi k}$.
  Update the policy by maximizing the PPO-Clip Objective:

$$\theta_{k+1} = arg\ \max_{\theta} \frac{1}{|\mathcal{D}_K|T} \sum_{\tau \in \mathcal{D}_K} \sum_{t=0}^{T} min\left(\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta k}(a_t|s_t} A^{\pi_{\theta k}}(s_t, a_t), g\left(\epsilon, A^{\pi_{\theta k}}(s_t, a_t)\right) \right)$$

  typically via stochastic gradient ascent with Adam.
  Fit value function by regression on mean-squared error:

$$\phi_{k+1} = arg\ \min_{\phi} \frac{1}{|\mathcal{D}_K|T} \sum_{\tau \in \mathcal{D}_K} \sum_{t=0}^{T} \left(V_\phi(s_t) - \hat{R}_t \right)^2$$

  typically via some gradient descent algorithm.
end for

2.8 Advantage Actor-Critic

OpenAI introduced Advantage Actor-Critic (A2C), an asynchronous version of the Asynchronous Advantage Actor-Critic (A3C) algorithm. Policy-based and value-based reinforcement learning algorithms are combined in this method. A2C uses two neural networks, one for the actor and one for the critic, just like other actor-critic techniques. The actor modifies the policy distribution on the advice given by the critic, and the critic oversees estimating the value function. The pseudo-code, as depicted in Algorithm 2, is derived from, and further elaborated upon in the work by [12].

Algorithm 2 Advantage actor-critic - pseudo code

// Assume parameter vectors θ
Initialize step counter t ← 1
Initialize episode counter E ← 1
repeat
  Reset gradients: dθ ← 0 and $d\theta_v$ ← 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}
  until terminal s_t or t - t_{start} == t_{max}
  $R = \begin{cases} 0 \text{ for terminal } s_t \\ V(s_t\theta_v) \text{ for non-terminal } s_t \end{cases}$

  for i ϵ { t -1,...,$t_s tart$ } **do**
    R ← $r_i + \gamma$R
    Accumulate gradients wrt θ: dθ ← dθ + ∇_θ log$\pi(a_i|s_i;\theta)$(R - V $(s_i;\theta_v)$) + $\beta_e \varphi$H($\pi(a_i|s_i;\theta)$ / $\varphi\theta$
    Accumulate gradients wrt θ_v: dθ_v + β_v(R - V $(s_i;\theta_v)$)(φV($s_i\theta_v$) / $\varphi\theta_v$)
  end for
  Perform update of θ using dθ and of θ_v using dθ_v.
  E ← E + 1
until E > E_{max}

2.9 Deep Q Network

Deep Q-Learning, the algorithm behind the Deep Q Network (DQN), stands as a pioneering example in reinforcement learning, capable of tackling complex problems that would typically challenge human capabilities. Originally implemented for Atari 2600 emulator games, the technique functions in a setting where the state is represented by a 210x160 128 RGB pixel frame. This raw input is preprocessed into grayscale, reduced to 110x84, and cropped to eliminate irrelevant image portions, leading to a final 84x84 frame. DQN operates by utilizing a replay memory to store state-action pairs for approximating the expected return and employing an ϵ-greedy policy for action selection. The algorithm begins by initializing the replay buffer, a step demanding considerable system RAM, along with initializing the action-value function with random weights. Subsequently, it selects actions randomly or based on the best current knowledge, initially exhibiting randomness. After executing an action, the resulting reward and state are stored in the replay memory. A batch is then sampled from this memory, and gradient descent is performed on this batch to adjust the weights. The experimental scenario, such as that seen in the Doom environment, mirrors this setup, where the agent's state information consists solely of a frame buffer.
The pseudocode as depicted in Algorithm 3 is taken from [13].

Algorithm 3 DQN

Initialize replay memory buffer
Initialize action-value function with random weights
for $episode = 1, M$ **do**
    Initialize sequence s.
    for $timestep = 1, T$ **do**
        With probability ϵ select a random action
        otherwise select $max_a Q*(s,a)$
        Perform action a, receive reward and state
        Store transition $(s_t, a_t, r_t, r_{t+1}$ in replay memory
        $Sety_j = \begin{cases} \mathrm{r}_j \text{ for terminal state} \\ \mathrm{r}_j + \gamma max_{a'} Q(s,a) \text{ for non terminal state} \end{cases}$
        Perform gradient descent step
    end for
end for

2.10 VizDoom

Introduced in 2016 by Kempka et al., VizDoom emerged as a research platform for reinforcement learning, drawing inspiration from the success achieved in the Atari 2600 domain. Unlike its predecessors, VizDoom exclusively relies on visual input, making it the first first-person shooter environment to adopt such an approach. This design choice offers the advantage of obviating the necessity for researchers and developers to engineer particulars that the agent needs to learn.
Additionally, Doom distinguishes itself with its lightweight and adaptable nature, consuming only 40MB of disk space. Being software-rendered, it can be executed without a desktop environment, facilitating remote operation within a terminal window.
In the years 2016 and 2017, a multiplayer death-match competition served as a platform for further evaluation and testing of visual-based reinforcement learning in intricate environments necessitating concurrent tasks, such as navigation, resource collection, aiming, and opponent evasion. The competition's outcomes revealed that while the bots demonstrated competence in tactical aspects like aiming and shooting, notable vulnerabilities were apparent, and easily exploitable by human players. For instance, the bots' reliance on hard-coded crouching revealed a susceptibility to evade vertical aiming. Furthermore, shortcomings were evident in strategic decision-making areas such as positioning and navigation.

2.11 Associated Works

Various platforms are available to assess the efficacy of reinforcement learning algorithms, with OpenAI Gym being a prominent choice [14]. Notably, it includes the Atari environment, which was pivotal in the success of early visual learning reinforcement agents. A more contemporary platform, VizDoom [15] derives from the iconic 1993 video game Doom, presenting intricate 3D environments wherein agents navigate and confront adversaries across diverse scenarios.

Since its inception, VizDoom has served as a key component in numerous research endeavors, primarily utilized for benchmarking different iterations of RL algorithms. Table 2.1 below shows the representation of various DRL Algorithms and their working. Notably, in 2016 and 2017, competitions pitted various reinforcement learning agents against each other, with the IntelAct bot, employing a Direct Future Prediction algorithm, emerging as the clear victor, while a DQN-based bot secured second place.

DQN, alongside A2C and A3C, remains a staple in comparative studies of algorithmic variants. For instance, both were employed in the work of [16], where A3C exhibited markedly superior performance. This study employed a VizDoom scenario dubbed "Defend the Center," with an agent in the middle of a room having fixed ammunition and health. The objective in this scenario is to prolong survival by neutralizing spawning enemies aiming to eliminate the agent. The researchers suggested future investigations should explore the "Deadly Corridor" scenario.

2.12 Table Representation of DRL Methods

Table 2.1 Representation of DRL Methods

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

The early stages of reinforcement learning involved developing a robot's football-playing skills [17]. Subsequently, with the introduction of the VizDoom platform, numerous studies emerged focusing on training artificial intelligence agents to play Doom through reinforcement learning. VizDoom competitions were held in 2016, 2017, and 2018. The agents in these competitions aimed to train agents efficiently for first-person shooter (FPS) deathmatch games. The agents had to make efficient decisions based on the visual information of the VizDoom scenarios. To play well, agents need to navigate, visualize the scenarios, explore, and handle the opponent at the same time. The competition's outcomes indicate that while reinforcement learning can create competent agents for playing Doom but are not competent to compete against humans.

2.13 Doom AI Competitions 2016

The competition's goal was to create an agent that plays a multi-player deathmatch game and tries to score as many frags as possible. According to Doom, a frag is defined as the total number of opponents killed minus the total number of suicides (an agent that ends its life due to damage caused by itself). All the environment's features, including the depth buffer, unique game variables, and Doom's built-in agents, as well as any external resources like custom scenarios, could be used by the competitors to prepare their agents offline. Agents were limited to resources such as remaining health points, ammunition supply, and screen buffer information during the competition.

2.13.1 Rules of Doom AI Competitions 2016

In the tournament, there were two separate game tracks, and each track consisted of twelve matches, each lasting ten minutes. There was a total of two hours of gaming per track. In every match, an agent started the game, and whenever it got eliminated, it instantly respawned at a randomly chosen respawn site. To make things fair, the respawn site was selected as randomly as possible from among the other players. Also, when a bot respawned, it had special protection for the first two seconds, during which it couldn't be attacked. This setup added an extra layer of strategy and fairness to the competition.

2.13.2 Tracks of Doom AI Competitions 2016

a) Limited deathmatch on a known map (Track 1)

Each agent had to compete on a single, pre-defined map. The only available weapon for Track 1 was a rocket launcher. Most VizDoom scenarios were somewhat small, allowing players to eliminate opponents by blasting a rocket into a nearby wall. It was also comparatively simple to commit suicide. Resources like armor, medikits, and ammunition were shown on the map. A fair match-making method was created since the number of participants 9 exceeded the maximum of 8 players per game allowed by VizDoom. Only one agent was eliminated out of all agents. 2 agents who had not performed well in the first 9 matches were not allowed to play in the last three. Table 2.2 below shows the results of the competition done on Track 1 2016, how many kills each agent does and how many times it died in the process.

Table 2.2 Results of competition Track 1 2016 [18].

Place	Agents	Frags (F)	Suicides (S)	F/S ratio	Kills	Deaths
1	F1	559	38	1.35	597	413
2	Arnold	413	119	1.90	532	217
3	Clyde	393	83	0.77	476	509
4	TUHO	312	112	0.67	424	465
5	5vision	142	64	0.28	206	497
6	Colby Mules	131	91	0.25	222	516
7	AbyssII	118	99	0.21	217	542
8	WallDestroyerXxx	-130	143	-0.41	13	315
9	Ivomi	-578	727	-0.68	149	838

b) Full death match on unseen maps (Track 2)

Each agent in track 2 had to compete on an unseen map and was initially equipped with a pistol. Compared to Track 1, the scenarios were roomier and included open areas, which increased the importance of precise aim. Table 2.3 below shows the results associated with Track 2, how each bot performed and its metrics. The scenarios featured a variety of weapons and equipment, including armor, medikits, and ammunition. Every map was created by the authors of the competition.

Table 2.3 Results of competition Track 2 2016 [18].

Place	Bot	Frags (F)	F/S ratio	Kills				Suicides (S)				Deaths			
				M1	M2	M3	T	M1	M2	M3	T	M1	M2	M3	T
1	IntelAct	256	3.08	113	49	135	297	19	7	5	41	47	24	12	83
2	Arnold	164	32.8	76	37	53	167	2	1	0	3	3	1	1	5
3	TUHU	51	0.66	51	9	13	73	7	15	0	22	31	29	17	77
4	Colby Mules	18	0.13	8	5	13	26	1	7	0	8	60	27	44	129
5	5vision	12	0.09	12	10	4	26	3	8	3	4	45	37	47	131
6	Ivomi	-2	-0.01	6	5	2	13	2	13	0	15	69	33	35	137
7	WallDestroyerXxx	-9	-0.06	2	0	0	2	0	5	6	11	48	30	78	156

Figures 2.4, and 2.5 below show the scenarios pictorially as to how those used in track 2 differed from track 1.

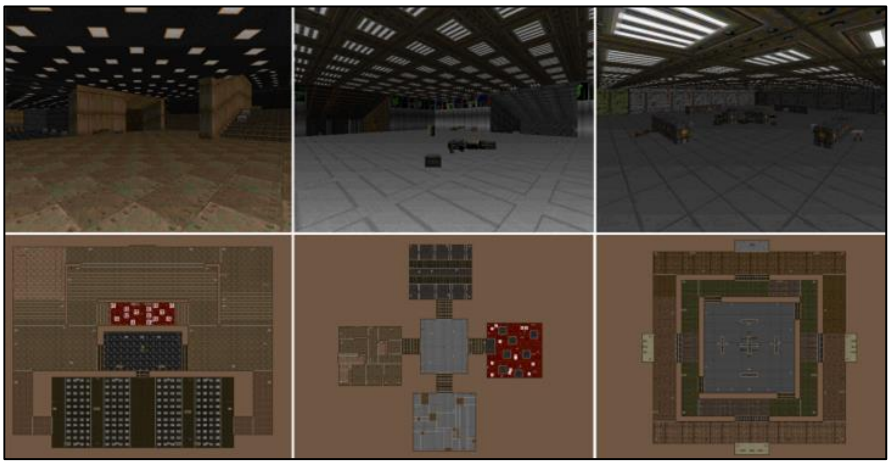


Figure 2.4 Scenarios used in track 2.

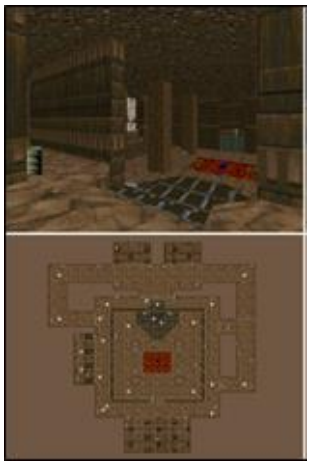


Figure 2.5 Scenario used in track 1.

2.14 Doom AI Competitions 2017

2.14.1 Rules of Doom AI Competitions 2017

Not much of the rules changed compared to the previous competition. However, a new version of VizDoom (1.1.2) was introduced. The increase in the number of players from 8 to 16 allowed all players to submit to their agents in a single game. Each track consists of 10 matches lasting 10 minutes long. Five new and unseen maps were introduced in Track 2, each of which was utilized for two matches. The scenarios were selected at random from a larger list of scenario packs that the Doom community had provided. There were four highly rated Doom multiplayer scenarios, each with several scenarios ranging from 6 to 32 (for a total of 88 maps). As a result, the chosen scenarios were distinguished by their high caliber and careful design. These scenarios were smaller than the ones utilized in the 2016 competitions.

2.14.2 Tracks of Doom AI Competitions 2017

a) Track 1

In comparison to 2016, the quality of bots that were submitted in 2017 was noticeably higher. Track 1 consists of strong agents in terms of their performance. The difference in the number of fragments was not very large, the top agent scored 248 while the lowest agent scored 109. Marvin won the track with 248 frags, which was just 3 more than Arnold2, the runner-up, and 33 more than Axon. The results are shown in a tabular form below in Table 2.4. The interesting thing is to note that Marvin concentrated on obtaining supplies, such as medikits and armor, which significantly improved his odds of survival, rather than being particularly skilled at hitting targets. Though it was killed the least frequently, Marvin was struck the greatest number of times. However, Arnold2 was more focused on targeting (accurate shooting and detection).

Table 2.4 Results of competition Track 1 2017 [18]

Place	Agents	Frags (F)	Suicides (S)	F / S ratio	Kills	Deaths
1	Marvin	248	67	1.16	315	213
2	Arnold2	245	69	0.84	314	69
3	Axon	215	37	0.77	252	37
4	TBoy	198	31	0.60	229	330
5	F1	164	15	0.57	179	290
6	YanShi	158	88	0.58	246	273
7	DoomNet	139	40	0.50	179	280
8	Turmio	132	77	0.87	209	280
9	Alpha Doom	109	30	0.39	139	281

b) Track 2

Since every agent received more than 50 frags, it was able to move, aim, and fire at its opponents, like Track 1. There wasn't much of a difference between the first two bots in track 2. Arnold4 took first place in the competition with 275 frags, closely followed by YanShi with 273. Throughout the entire competition, Arnold4 was the most accurate bot and the only one without suicide. This proved to be essential in defeating YanShi, who had the same total number of kills 275 but two suicides. However, YanShi was the greatest agent at avoiding death and had the most detection precision, which allowed him to attain the highest frags/death ratio. The results are stated in a tabular form below in Table 2.5. IntelAct, the previous competition's champion of Track 2, came in second place with significantly fewer points 221. It's possible that fewer things on the VizDoom scenario in Track 2 caused Marvin to finish lower in fourth place with 193 frags.

Table 2.5 Results of competition Track 2 2017 [18]

Place	Agents	Frags (F)	F/S ratio	Kills				Suicides (S)						Deaths					
				M1	M2	M3	T	M1	M2	M3	M4	M5	T	M1	M2	M3	M4	M5	T
1	Arnold4	275	1.25	35	49	50	84	0	0	0	0	0	0	36	48	42	44	50	220
2	Yanshi	273	1.47	56	67	55	56	1	1	0	0	0	2	28	33	39	40	56	186
3	IntelAct	221	0.89	53	46	51	51	10	1	6	0	3	20	36	48	53	48	62	247
4	Marvin	193	0.99	36	38	52	46	4	3	4	0	12	23	34	44	39	41	47	195
5	Turmio	164	0.82	12	58	26	45	5	2	7	0	3	17	46	40	39	41	44	200
6	TBoy	139	0.58	26	16	33	13	7	0	0	0	0	7	50	48	48	49	47	240
7	DoomNet	62	0.28	7	20	9	19	4	8	6	0	4	22	36	38	42	51	54	221

Figure 2.6 below shows the scenarios of Track 2 of 2017 and helps us visualize how it differs from that of 2016.

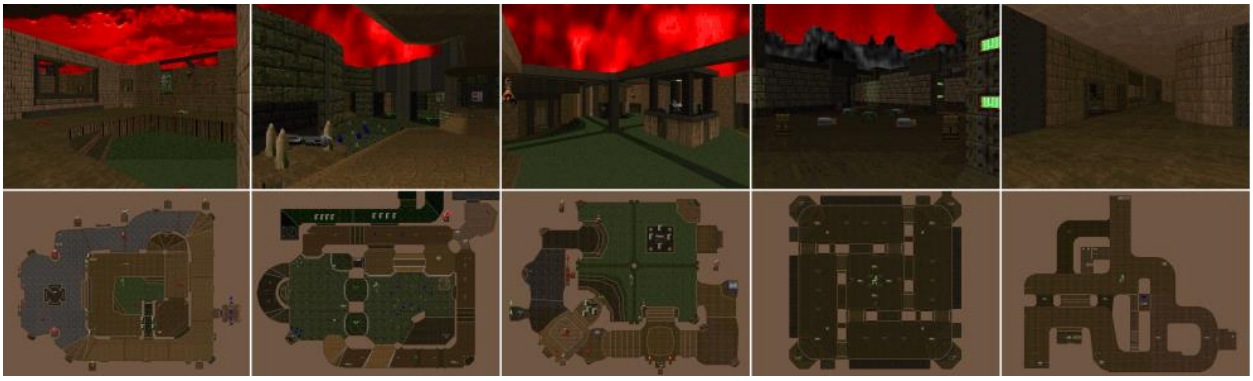


Figure 2.6 Scenarios of Track 2 of 2017 competition [18].

2.15 Doom AI Competitions 2018

The goal was to develop an agent that can complete Doom game levels without the need for auxiliary information by using data that is typically available to human players, primarily visual data. The VizDoom framework, which provides real-time screen access, must be used by the agent. Agents were given the task of beating single-player levels as quickly as they could. The entry threshold is minimal because levels change with complexity over time.

2.15.1 Tracks of Doom AI Competition 2018

a) Track 1

51 teams participated in the 2018 competition and submitted 204 agents out of which the 4 best were selected based on their performance. TSAIL is a reinforcement learning agent that participated in the Visual Doom competition on computational intelligence and Games 2018. TSAIL outperformed all other agents in single-player mode, where they had to navigate levels and survive for as long as possible with an average completion time of 25.34 minutes. TSAIL consistently completed the single-player levels faster than most other agents. Similarly, Doom Net was another AI agent that played in the competition. Its average competition time was 29.86 minutes across 10 unknown maps. It performed well on maps 2, 5, and 8, achieving an average completion time of 5 minutes. The results of the Agents in Track 1 of this 2018 competition can be seen in Table 2.6. This suggests it excels in efficient map exploration and resource acquisition. Doom Net's overall performance indicates efficient enemy navigation and resource management. Its performance across varied maps implies an ability to adjust its behavior to different environments and challenges.

Table 2.6 Results of Track 1 Doom AI competition 2018 [19].

Team	Agent	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	Total time (m)
TSAIL	**TSAIL**	3.90	5.00	0.72	0.34	0.82	0.32	4.06	5.00	5.00	0.20	25.34
DoomNet	**DoomNet**	0.97	5.00	0.95	1.97	5.00	0.33	5.00	5.00	5.00	0.65	29.86
VIPLAB	**agent_viplab**	2.45	5.00	5.00	0.87	2.42	0.48	5.00	5.00	5.00	0.33	31.54
Ddangelo	**DoomGai**	5.00	5.00	5.00	5.00	0.65	1.00	5.00	5.00	5.00	0.68	37.33

b) Multiplayer Track (Track 2)

The objective was to develop an agent that can participate in Doom deathmatches against other agents based solely on player-accessible data and without the need for any auxiliary information. For the agent to connect to the game, the VizDoom framework must be used. Track 2 is an entire Doom deathmatch on unseen scenarios, much as in past years. Agents will engage in multiplayer competitions, with the winner being the agent who gathers the most fragments. The Results of this Track 2 competition can be seen in Table 2.7. Where the number of frags for this competition is defined as frags = number of killed opponents' number of suicides. 33 teams participated in the 2018 track 2 competition and submitted 152 agents out of which the 3 best were selected from 10 unknown VizDoom scenarios based on maximum frag number.

Table 2.7 Results of Track 2 Doom AI competition 2018 [20].

Team	Agent	M1	M2	M3	M4	M5	M6	M7	M8	M9	M10	Total frags
Bwbell	**Marv2in**	19	15	53	31	21	51	33	34	32	53	342
TSAILAB	**AWM**	19	21	30	33	39	27	22	12	19	24	246
michaelkrax	**CVFighter**	26	18	21	21	30	40	16	24	9	29	234

Geisler was the first to implement AI in first-person shooter games [21] where a player's behavior was modeled in Soldier of Fortune 2. Bots were tuned using genetic algorithms in Counterstrike. An algorithm was devised by Smith to optimize team tactics in an unreal tournament. SARSA was another reinforcement learning algorithm that was part of research in FPS games [22]. Reinforcement learning methods were applied to learn the behavior of tanks in the game BZFlag [23]. Reinforcement plays a crucial role in the contribution of efficient algorithms in video games [24]. A lot of work is being done to train agents to interact with the environment to outperform human intelligence and achieve superintelligence [25]. Superintelligence is being achieved and human intelligence is being outperformed using reinforcement learning. These features are helpful in self-organized systems (like cellular networks) to improve their performance [26]. Reinforcement learning has revolutionized every category of games whether it is classical arcade games, real-time strategic games, and single agent 3D games TORCS to multi-agent 3D games like Quake 3. The process is only possible due to the publicly available research platforms [27] and competitions that are making a huge contribution to the development of artificial intelligence in games.
VizDoom is a software platform proposed for research in machine learning that takes advantage of unprocessed visual information for learning and minimizes the need for high-level information. It provides full control of the environment for training agents in different scenarios.

2.16 Applications Beyond VizDoom

While the VizDoom platform has served as a prominent testing ground for deep reinforcement learning (DRL) algorithms, the applications of these algorithms extend far beyond the realm of first-person shooter games. DRL algorithms, including Proximal Policy Optimization (PPO), Double Deep Q-Network (DDDQN), and Advantage Actor-Critic (A2C), have found diverse applications across various industries and domains, showcasing their versatility and effectiveness in solving complex problems.

a. Robotics

In robotics, DRL algorithms are used for tasks like autonomous navigation, object manipulation, and robotic control. For example, researchers have used DRL to train robots to do things like move robotic arms, navigate in changing environments, and even walk on two legs. The work in [28] combines machine learning, computer vision, and robotic systems to create a vision-based learning framework for intelligent robot control. The idea is to use deep reinforcement learning as a general-purpose framework for AI, meaning it can be applied to different tasks and platforms.

b. Finance

DRL algorithms are being used in finance for tasks like algorithmic trading, portfolio optimization, and risk management. These algorithms use historical financial data and real-time market information to learn the best trading strategies and adjust to changing market conditions to maximize returns and minimize risks. Deep reinforcement learning (DRL) has been recognized as an effective approach in quantitative finance, and many beginners are interested in gaining hands-on experience. However, building a practical DRL trading agent that can decide where to trade, at what price, and what quantity involves challenging development and debugging. The work in [29] introduces a DRL library called FinRL, which helps beginners get started in quantitative finance and develop their own stock trading strategies.

c. Healthcare

In healthcare, DRL algorithms have been applied for medical image analysis, drug discovery, and personalized treatment planning. These algorithms can assist in diagnosing medical conditions from imaging data, optimizing treatment protocols, and even discovering new therapeutic compounds through virtual screening and molecular design. The rapid increase in the percentage of chronic disease patients

along with the recent pandemic pose immediate threats on healthcare expenditure and elevate causes of death. This calls for transforming healthcare systems away from one-on-one patient treatment into intelligent health systems, to improve services, access, and scalability, while reducing costs. Reinforcement Learning (RL) has witnessed an intrinsic breakthrough in solving a variety of complex problems for diverse applications and services. The work in [30] surveys recent models and techniques of RL that have been developed for supporting intelligent-healthcare systems.

d. Autonomous Vehicles

The development of autonomous vehicles relies heavily on DRL techniques for decision-making and control. These algorithms enable vehicles to perceive their surroundings, navigate complex traffic scenarios, and make real-time driving decisions to ensure safe and efficient transportation. Autonomous driving is a promising technology to reduce traffic accidents and improve driving efficiency. The work in [31] presents a deep reinforcement learning (DRL)-enabled decision-making policy, constructed for autonomous vehicles to address the overtaking behaviors on the highway.

e. Natural Language Processing (NLP)

DRL algorithms have been used in NLP for tasks like language translation, sentiment analysis, and text generation. These algorithms can learn human-like language from large text datasets, enabling applications like chatbots and virtual assistants. The paper in [32] discusses statistical models and the limitations due to machine intelligence, as well as the importance of CNN in neural networks. It also talks about TensorFlow, an open-source software for deep learning, and how it has an edge over conventional models. The paper also recommends using reinforcement learning as an extension to neural networking, which is widely used in gaming for NLP.

f. Manufacturing and Industry

DRL algorithms are like smart tools that can help factories and other industrial places run better. They use data from sensors and machines to find problems, predict when things might break, and make the whole production process work smoother. Some experts through their work in [33] think these algorithms are important for dealing with the ups and downs of modern industry. They're also looking at new ideas, like digital twins, to make these algorithms even better.

g. Environmental Conservation

Researchers are using computer algorithms to help understand and protect the environment. These algorithms analyze data and satellite images to predict changes in habitats and come up with plans to protect animals and plants. The United Nations has declared the next decade as the time to focus on fixing damaged ecosystems caused by human activity. One of the big problems is that human actions have broken up habitats and made them too small for animals or too far apart for them to move around. Even though we need to figure out how to fix this, we're not sure how to do it effectively. A research paper in [34] shows that a type of computer algorithm called deep reinforcement learning could help with planning how to reconnect habitats.

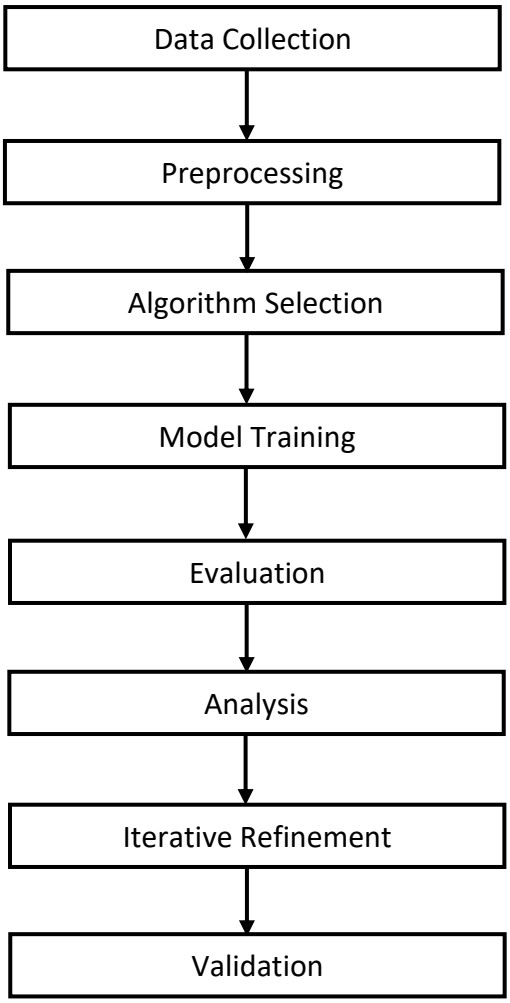
h. Gaming and Entertainment

Beyond VizDoom, DRL algorithms are changing the gaming and entertainment industry by improving game mechanics, creating realistic virtual environments, and making characters more lifelike. An article in [35] talks about the latest developments in reinforcement learning (RL) and how it's being used in games. It gives an overview of RL and explains how it's being used now. The article also looks at how much scientific research on deep learning (DL) and reinforcement learning is focused on games like ATARI, Chess, and Go.

3. Methodology and Experiments

This section explains how the research was done and gives details about the experiments. The study focuses on comparing different algorithms using the game Doom. In Doom, players see the game from their character's perspective and can move around freely in a 3D environment. They can pick up weapons and health packs and fight against computer-controlled enemies that can attack them. One interesting thing about Doom is that it allows users to create their custom game scenarios, like designing maps, programming how things work in the environment, and setting specific goals and conditions for winning. First, we will look at the implementation of different algorithms used in our project and then the following parts will talk more about the specific scenarios used in the experiments.

3.1 Methodology Diagram



Explanation

a) **Data Collection:** Arrows flow from here, indicating data is collected from the VizDoom environment.
b) **Preprocessing:** The collected data is sent for preprocessing (normalization, etc.). The resulting preprocessed data becomes the input for the next step.
c) **Algorithm Selection:** A decision is made to choose an appropriate reinforcement learning algorithm based on the problem and data.
d) **Model Training:** The preprocessed data is used to train the chosen algorithm, updating the agent's policy or value function.
e) **Evaluation:** The performance of the trained agent is assessed using various metrics.
f) **Analysis:** The results from evaluation are analyzed to understand the agent's behavior and performance.
g) **Iterative Refinement:** Based on the analysis, the methodology is refined by adjusting algorithms, hyperparameters, or preprocessing techniques.
h) **Validation:** The effectiveness and generalization of the trained agent are validated through different methods.

3.2 Tools and Technologies used for training using PPO, A2C, and DDDQN.

The following tools and technologies were used for the implementation of the algorithms.

a. PyTorch:
PyTorch is an effective deep-learning framework for creating and refining neural network models. It provides a flexible and intuitive framework for building and training deep learning models.

b. Tensor board:
Tensor board is a package that allows us to log and visualize training metrics so that Tensor board can analyze them.

c. VizDoom:
A platform for training agents in different scenarios with different complexity levels.

d. Open Gym:
The VizDoom environment used to train reinforcement learning agents can be interacted with via OpenAI Gym. It makes it simpler to test out various reinforcement learning algorithms by offering a consistent method for defining environments, acting in them, and monitoring their states and rewards. Agents interact with the VizDoom environment by initializing the environment, taking actions, and receiving rewards.

e. Matplotlib:
We have used Matplotlib to analyze the performance of the reinforcement learning agents and have visualized the current state of the game. To see how the agent is adapting to the VizDoom environment and learning new skills, we have used Matplotlib to plot a variety of graphs, including training progress, awards the agent has received over time, kills, and any other pertinent data.

f. Git:
Git is utilized for version control, mostly for cloning the VizDoom repository and handling codebase changes. Git version control ensures that the codebase of the project is manageable, well-organized, and simple for us to share and replicate. Our project has incorporated these technologies to produce a thorough framework for testing and training reinforcement learning agents within the VizDoom setting. They give us the tools we need to define environments, put algorithms into practice, train models, see the outcomes visually, and manage code effectively.

3.3 Hyperparameters

a. **Learning Rate (LR):**
The pace at which the optimizer is used to change the model's parameters during training.

b. **Discount Factor (gamma):**
In the Q-learning update equation, the discount factor is used for future rewards. The parameters that control the agent's exploration strategy begin with a high level of exploration and gradually decrease are called exploration parameters (explore start, explore stop, decay rate).

c. **Memory Size:**
The capacity of the experience replay buffer, which is used to hold historical data for training purposes. The quantity of experiences taken from the replay buffer for every training iteration is known as the batch size.

d. **Total Episodes:**
The number of training episodes is the total number of episodes that were used to train the agent.

e. **Pretraining Episodes:**
The total number of firsthand encounters that are added to the replay buffer before training. The number of frames to repeat an action on is called the frame skip. The number of episodes after which the model weights are saved is known as the frequency of model saving.

Using a DDDQN model, the offered code teaches an agent how to play Doom. It communicates with the gaming environment, preprocesses state, chooses actions according to the model's projected Q-values, and double Q-learning and experience replay are used to update the model's parameters. The tensor board is used to visualize the training progress as a variety of hyperparameters are adjusted to optimize the training process.

3.4 Environment and Scenarios in VizDoom

This section talks about the different situations used to test the agents. It gives some basic information about each scenario, like how rewards are given, what the agents are supposed to do, and how they can reach their goals. It also includes details like the size of the game screen, how often frames are skipped, and screenshots for each scenario.

a) Basic Scenario

In a basic scenario, an agent has the properties of attacking, moving right and left. Ammo is the available game variable in the scenario. Agent resides in a room with a single enemy. To visualize below is Figure 3.1 showing the basic scenario in VizDoom.

Figure 3.1 Scenario of basic in VizDoom.

b) Deadly Corridor Scenario

In the scenario of a deadly corridor, an agent has the properties of moving left, moving right, attacking, moving forward, backward, turning left, and right. Health and ammo are the available game variables of the scenario. The main aim of this scenario is to reach the goal (the vest). The vest is placed at the opposite end of the room. The agent appears in a room with a total of 6 enemies at left and right [36]. Two different screenshots of our agent in starting position and then killing the opponent is shown in Figures 3.2, and 3.3 below.

Figure 3.2 Starting position of the agent.

Figure 3.3 Agent killing the opponent.

c) Defend the center Scenario.

In this scenario, shown in Figure 3.4, there's a central agent armed with a gun and limited ammunition. The agent can only turn and shoot left or right. Around the central agent, five enemies are coming from different directions. When the agent kills one enemy, another one immediately appears. The goal for the agent is to kill as many enemies as possible. Each enemy killed gives the agent a reward of +1, but if the agent gets killed, it receives a penalty of -1. However, because the agent doesn't have much ammo, the agent can get killed. Figure 3.4 shows the Defend the Center VizDoom scenario below.

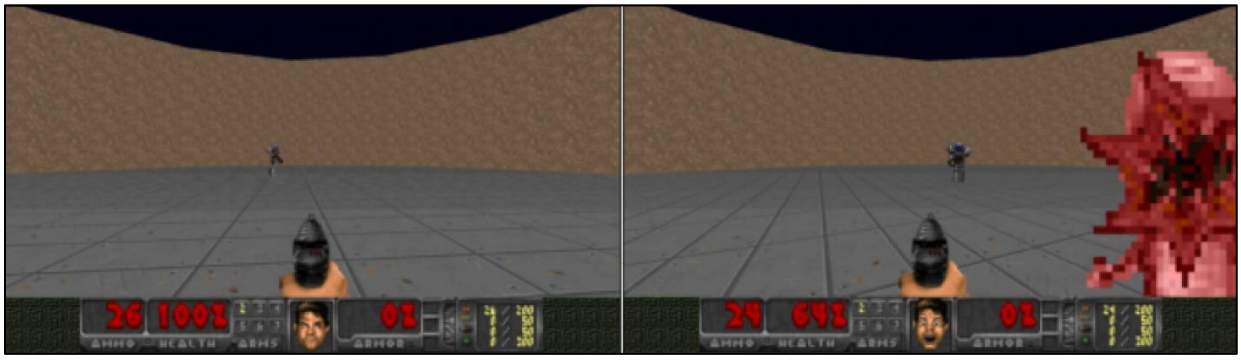


Figure 3.4 Scenario of Defend the Center in VizDoom.

3.5 Experimental Design and Environment

This section describes the experimental setup and hyperparameters (Table 2.8) that were used to train and evaluate the agents. There is a designated batch size of 32 and a sequence length of 4. Weight modifications are made to the target networks before the start of training iterations. The parameter Epsilon starts at 1.0 at the beginning of the training phase and gradually decreases with each training step until it eventually converges to 0.0001. The agents undergo testing with Epsilon set to 0.01 after a predetermined number of steps, and a graphic representation of the average episode reward across 100 episodes is produced. The considered hyperparameters settings in the experiment are shown in Table 3.1. A 60,000-bit memory buffer is used, and only successive observation sequences are read from it, with the last observation acting as a terminal point. This modification is especially important in situations like health-gathering, as it mitigates major negative incentives if the agent expires during the last observation. Regularly, modifications are implemented to enhance training, including increases to the batch size and sequence length, which are 64 and 8, respectively. The first four events are used to refresh the agent's memory, while the previous four are utilized for training. We increase the number of gradient steps to 8000 and the interval before memory sampling to 15 steps. To further improve the training procedure, the learning rate has been lowered to 0.0001.
This entire implementation is carried out using PyCharm 2021 Professional version, VizDoom, TensorFlow, Keras, OpenCV, CMake, GCC, and Python 3.6 (64-bit) on a Windows 10 64-bit operating system that is furnished with an NVIDIA GTX 1070 GPU with 8.00 GB DDR RAM and 256 GB SSD as well as an Intel(R) Core (TM) i3-6100U CPU @ 2.30GHz 2.30 GHz. The algorithms execute actions, optimize, and modify the networks by thousands to millions of iterations. Table 3.1 outlines the hyperparameter settings for three different algorithms. A2C, PPO, and DDDQN. Each hyperparameter is an essential component that can significantly influence the performance and training of the agent.

Table 3.1 Considered hyperparameter settings in the experiment.

Parameters	A2C	PPO	DDDQN
Discount Factor	0.99	0.99	0.99
Learning Rates	0.0002	0.0001	0.0002
Experience	60k	60k	60k
Screen Buffer	640, 480	640, 480	640, 480
Batch Size	32	32	32
Initial Decay	1.0	1.0	1.0
Final Decay	0.0001	0.0001	0.0001
History Length	4	4	4
Frames per Action	4	4	4
Time Elapsed (Hours)	50	72	70

3.6 Data Collection and Reward Shaping

Data collection forms an important component of the agent training process as it lays the foundation for the agent to get acquainted with its environment and subsequently learn and improve its performance over time. The major steps in data collection include Environment interaction, State representation, Action execution, Feedback signal, and Experience replay buffer [37].

a. Environment Interaction:

It refers to the interplay between agent action and its observation as the agent takes any action in the environment. The agent just receives different observations in return letting it know the outcome of its doings.

b. State Representation:

State refers to the compiled form of observations that is not limited to visual information, but also includes sensory information and any relevant features that the agent can receive at any given moment.

c. Action Execution:

Based on the present state of the environment, the agent selects the most suitable action that is then executed in the game leading to the shift of the environment to a new state.

d. Feedback (Reward Signal):

Depending on the action taken by the agent, the agent receives feedback in the form of a reward that fluctuates in value, motivating or demotivating the agent to take the action in the future. For example, successfully hitting a target may yield a positive reward, while taking damage could result in a negative reward.

e. Experience Replay Buffer:

The experiences (state, action, reward) are stored in a buffer for the agent to learn from a diverse track record rather than just close on the result of the immediate action. This approach improves the efficiency of learning in the long run.

3.7 Reward Shaping

a. Definition of Rewards

It is the indicator or value that is used as a form of feedback by the agent to know the consequence of its action.

b. Encouraging Desired Actions and Discouraging Undesired Actions

By molding the reward function equation in a way that motivates the agent to take certain actions, we manipulate those ways. For example, in a reward function reward = kill count * 1200 + delta ammo* = -0.5, we are motivating the agent to eliminate more enemies by giving the kill count a larger positive value while demotivating to lose delta ammo by giving ammo a negative value.

3.8 Implementation of PPO Algorithm

PPO is an RL algorithm that is designed to optimize the policies in reinforcement learning. PPO improves the policies by avoiding large policy updates that lead to catastrophic instability or forgetting. PPO is the derivation of trust region policy optimization (TRPO) algorithm [38]. TRPO decides the size of the optimal step for the upgradation of a gradient in an RL algorithm. Quadratic approximation was used to solve the TRPO algorithm but in recent times PPO algorithm has been used to solve problems involving first-order approximation.

3.8.1 Mathematical Form

$$L^{CLIP}(\theta) = E_t\,[min\,(\pi_\theta(a_t|\,s_t)\;/\;\pi_{\theta\,old}(a_t|\,s_t)\,A_t^{GAE},\; clip(1-\epsilon,1+\epsilon)\,A_t^{GAE})] \qquad \text{eq 3.1}$$

Explanation

a. Policy π_θ:

In $\pi_\theta(a_t|s_t)$ a_t is the probability of taking action in the state s_t under the parameterized policy parameters θ.

b. Old policy:

In $\pi_{\theta, old}(a_t|s_t)$ a_t is the probability of taking action in the state s_t under the old policy. It is used to calculate the ratio between probabilities in the objective function.

c. Advantage A_t^{GAE}:

A_t^{GAE} is an estimate of the advantage function that can be found in the generalized advantage estimation. It calculates the benefit of taking action in a state s_t about the average benefit in that state. It aids in lowering the policy gradient estimate's variance.

d. Objective Function:

$L^{CLIP}(\theta)$ is the objective function that is used to maximize the reward and limit the policy updates. It is used to prevent significant policy changes by encouraging new policies to be like the old ones.

e. Clip Function:

The policy update's size is constrained by the clip function. It is employed to make sure that there are no significant changes to the policy in between updates, avoiding significant oscillations that can cause instability.

f. Expectation E_t:

An expected sequence of events, actions, and rewards is assumed. The expectation is estimated by the algorithm using samples from several trajectories.

The probability ratio $r_t(\theta) = (\pi_\theta(a_t|s_t) / \pi_{\theta\ old}(a_t|s_t)$ can be represented as follows.

$$L^{CPI}(\theta) = E_t [(\pi_\theta(a_t| s_t) / \pi_{\theta\ old}(a_t| s_t) A_t^{GAE}] \quad \text{eq 3.2}$$

$$L^{\mathbf{CPI}}(\theta) = E_t\left[r_t(\theta) A_t \right] \quad \text{eq 3.3}$$

The conservative policy iteration is denoted by the superscript CPI. Maximizing L^{CPI} without a restriction would result in a large policy update, therefore, we are now considering modifying the objective. The following is the primary goal we suggest.

$$L^{CLIP}(\theta) = E_t [min \left(r_t(\theta) A_t, clip \left(r_t(\theta), \left(1-\epsilon,1+\epsilon\right) A_t \right)\right] \quad \text{eq 3.4}$$

Here epsilon (ϵ) is the hyperparameter that is tuned based on the scenario and controls the size of the policy update. L^{CPI} is the initial term inside min. By clipping the probability ratio, second term clip ($r_t(\theta)$, (1−ϵ,1+ϵ) A_t adjusts the surrogate aim and eliminates the motivation for shifting r_t out of the interval [1 - ϵ,1 + ϵ]. The goal is the lower bound or a pessimistic bound, on the unclipped objective since take the minimum of the clipped and unclipped objective. Under this technique, we only incorporate the change in probability ratio when it makes the objective worse and we ignore it otherwise [39].

3.9 Implementation of DDDQN (Double Dueling Deep Q-Network) Algorithm

DDDQN is a variant of the DQN algorithm that encompasses both DQN and double Q-learning. Dueling deep Q-learning introduces a novel network architecture consisting of two streams. One estimates the state-value function V(s) and outputs a scalar, while the other estimates the action advantage function {A (s, a); a ∈ A} and outputs a vector of size representing the number of possible actions. The outputs from the two streams are then aggregated (in a specific way) to produce the Q-values {Q (s, a); a ∈ A}. While the advantage stream calculates the benefit of acting at a particular state, the value stream allows us to estimate state values because some states are indifferent to all possible actions (no action leads to an improved predicted cumulative reward). Thus, the advantage is Q (s, a) = A (s, a) + V (s) [40].
There are two distinct estimations in the dueling DQN, and they are as follows:

i. An agent's perceived usefulness in a particular state is estimated using this measure of state value.
ii. Calculate the benefit of each action in each state [41].

3.9.1 Mathematical Form

Two Q-value functions and the Q-learning update algorithm form the foundation of the mathematical description of DDDQN.

$$Q\left(s,\, a;\, \alpha,\, \beta\right) \,=\, V\,\left(s;\, \theta,\, \beta\right) \,+\, \left[\, A\,\left(a,\, s;\, \theta,\, \alpha\right) \,-\, 1/\left|A\right|\, \sum\, \bar{a}\, A\,\left(\bar{a},\, \theta,\, \alpha\right)\right] \qquad \text{eq 3.5}$$

The two components of the Q-network include a value stream (V (s; θ, β)) and an advantage system A (a, s; θ, α)). Here

- Q (s, a; θ, α, β) is the estimated Q-value for state s and action a.
- V (s; θ, β) is the value of state 's'.
- A (a, s; θ, α) is the advantage of taking action 'a' in state 's'.
- θ represents the parameters of the shared layers.
- α represents the parameters of the advantage stream layers.
- β represents the parameters of the value stream layers.
- |A| is the number of possible actions.

3.10 Implementation of A2C (Advantage Actor Critic) Algorithm

A2C is a reinforcement learning algorithm that combines the properties of both policy gradient methods and value function methods [42]. A2C is the synchronous version of the A3C policy gradient method. A2C stabilizes learning by substituting the advantage function for the Action value function as the Critic. The Advantage function is supposed to determine how much better action is taken at a state in comparison to the average value of the state.

3.10.1 Mathematical Form

$$L\left(\theta\right) \,=\, \sum_{t=0}^{T}(log\pi_{\theta}(a_t|\ s_t)\cdot\delta_t+\ \beta\cdot H\ (\pi_{\theta}(\ a_t|\ s_t))\ -\ \alpha\cdot V^{\pi\theta}\left(s_t\right)) \qquad \text{eq 3.6}$$

- s_t: State at time t.
- a_t: Action taken at time t.
- r_t: Reward received at time t.
- $\pi_\theta(a_t|s_t)$: Policy, i.e., the probability of taking action at a given state s_t and parameterized by θ.
- $V^{\pi\theta}(s_t)$: Value function, estimating the expected return starting from state s_t following policy π_θ.
- θ_t: Advantage function, representing the advantage of taking action a_t in-state s_t over the average action value in that state.
- γ: Discount factor for future rewards.

3.11 Inputs for A2C algorithm

a. **Input shape:** The size of every state in the environment (C, H, and W).
b. **Action size**: TURN_LEFT, TURN_RIGHT, ATTACK (3 for Defend the Center), MOVE_LEFT, MOVE_RIGHT, ATTACK (3 for Basic), MOVE_LEFT, MOVE_RIGHT, ATTACK, MOVE_FORWARD, MOVE_BACKWARD, TURN_LEFT, TURN_RIGHT (7 for Deadly Corridor).
c. **Seed:** For reproducibility, use a random seed.

d. **Device:** Select between using the CPU and the GPU.
e. **Gamma:** Future reward discount factor.
f. **Alpha:** The actor (policy) network's learning rate.
g. **Beta:** The critical (value) network's learning rate.

3.12 Hyper Parameters for A2C algorithm

a. **Gamma:** Future reward discount factor.
b. **Alpha:** Actor learning rate.
c. **Beta:** Critical learning rate.
d. **Action size**: TURN_LEFT, TURN_RIGHT, ATTACK (3 for Defend the Center), MOVE_LEFT, MOVE_RIGHT, ATTACK (3 for Basic), MOVE_LEFT, MOVE_RIGHT, ATTACK, MOVE_FORWARD, MOVE_BACKWARD, TURN_LEFT, TURN_RIGHT (7 for Deadly Corridor).

4. Results and Comparison

In each scenario, reinforcement learning algorithms evaluate how well the agent is performing using metrics like remaining ammunition, enemy kills, overall score, and average health. Monitoring these variables throughout training sessions, they provides us with insights into the agent's progress. After each training step, the objective is to attain the desired outcomes. To accurately demonstrate the performance of every agent, their results are looked at individually. This approach enables a clear comprehension of variations among agents and allows overlaps between different models.

4.1 Defend the Center

The agent's displays trained on the defend-the-center scenario are shown in this section, along with their learning curves.

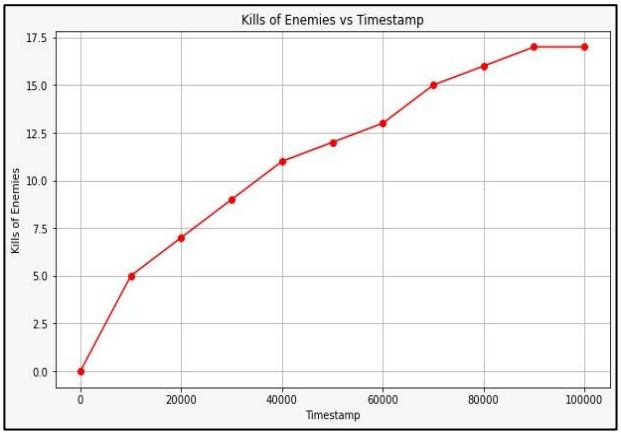


Figure 4.1 Kill Counts on Defend the Center using PPO.

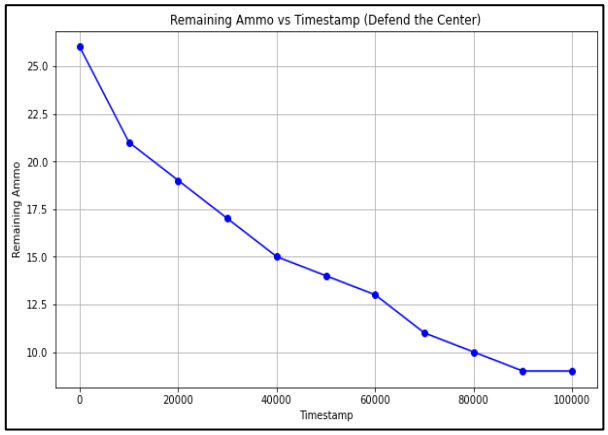


Figure 4.3 Ammo left on Defend the Center using PPO.

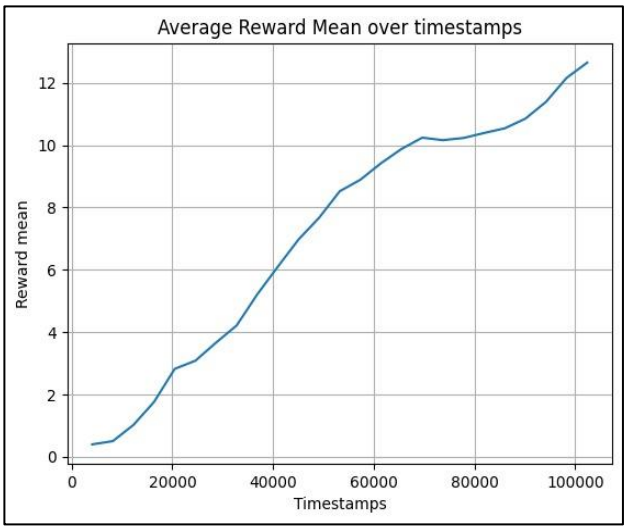


Figure 4.2 Agent's reward Mean on Defend the Center using PPO.

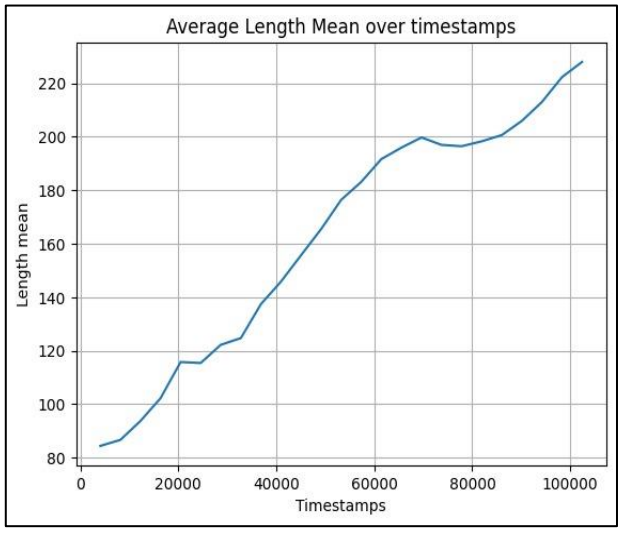


Figure 4.4 Average episode length in PPO-trained Defend. the Center agents.

The above graphs in Figures 4.1, 4.2, 4.3 and 4.4 show the performance of the Agent trained using PPO on Defend the Center scenario. Fig 4.1 shows the number of enemies killed as the agent is trained over increasing timestamps (periods of time) which gets higher as a positive sign. Fig 4.2 shows the Agent's average reward over each episode over extensive training which increases positively. Fig 4.3 shows the Ammo left as the agent is trained over extensive timestamps which decreases as it kills more enemies. Fig 4.4 shows the average episode length over timestamps which increases as the Agent learns to survive and kill its enemies gaining higher rewards.

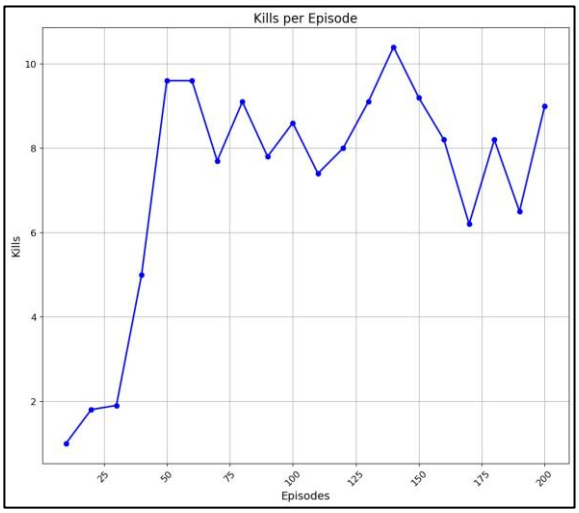


Figure 4.5 Reward graph trained using DDDQN in defend the center scenario.

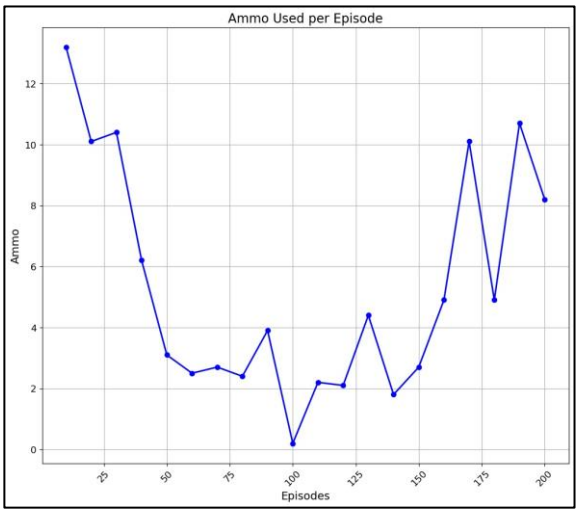


Figure 4.6 Ammo vs episodes graph using DDDQN in defend the center scenario.

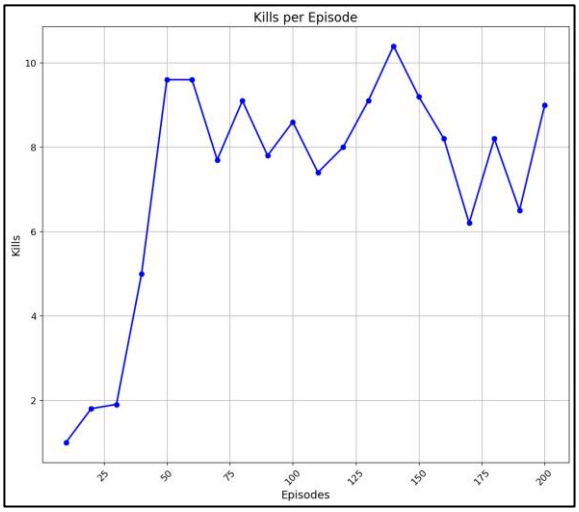


Figure 4.7 Reward graph trained using DDDQN in defend the center scenario.

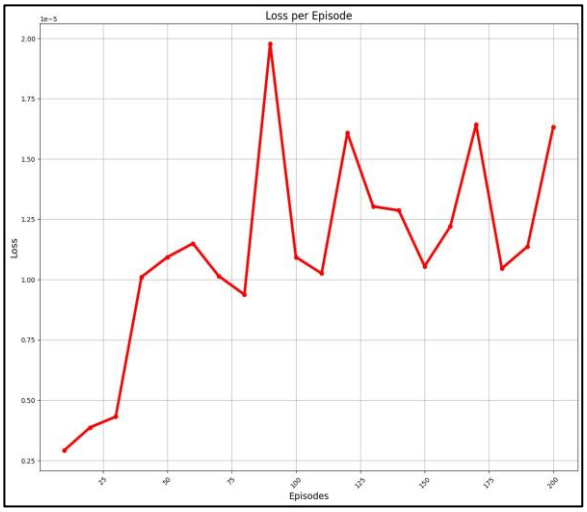


Figure 4.8 Training of loss graph using DDDQN in defend-the-center scenario.

The agent trained with DDDQN demonstrated above-average performance, achieving satisfactory scores in the given scenario. Notably, the agent excelled in terms of both overall kills plus ammo preservation as shown in Fig. 4.5, Fig.4.6, Fig.4.7, and Fig.4.8.

The agent trained with the A2C algorithm exhibited a gradual improvement in performance during training. The graph shows that the agent's performance initially remains flat at around -0.2 until 500 episodes. There is a gradual increase in the average score over episodes from 500 to 2000 episodes. Following this initial phase, the agent demonstrably improved its ability to gather health packages, as evidenced by the stepwise increase in the curve. The agent's performance increases significantly after 2000 episodes, reaching a peak average score of about 0.8 around 2500 episodes. This trend of improvement was also observed in other performance metrics such as score, ammo remaining, and kill count, with some occasional fluctuations as presented in Fig. 4.9, and Fig. 4.10.

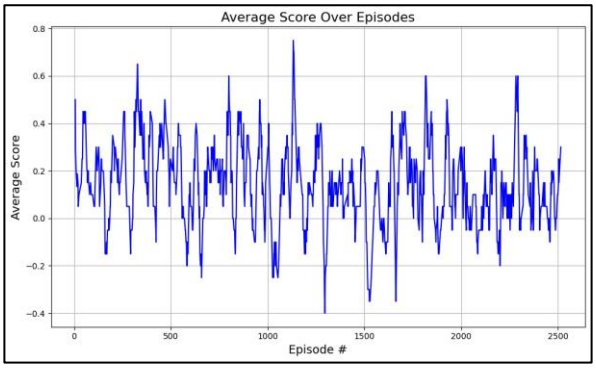


Figure 4.9. Training of agents in defend the center using the A2C algorithm.

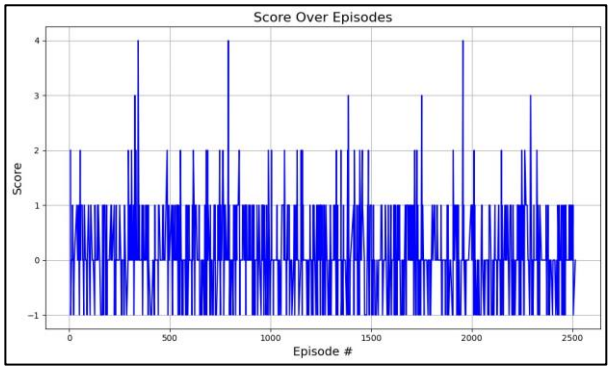


Figure 4.10 Total reward obtained using A2C algorithm in defend the center.

4.2 Deadly Corridor

The given section presents the displays including learning charts of the agents trained on the Deadly-Corridor scenario.

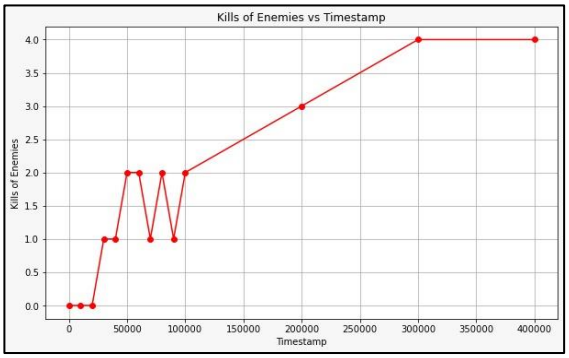


Figure 4.11 Kills using PPO in deadly corridor scenario.

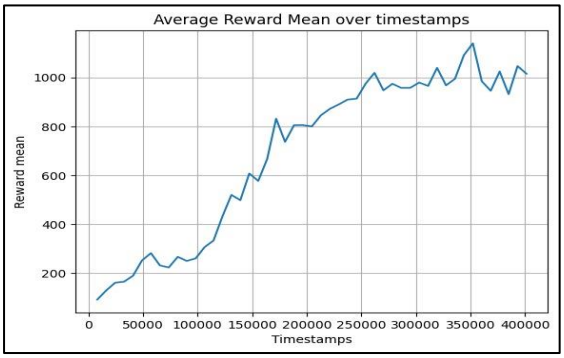


Figure 4.12 Average reward graph during training using PPO in the deadly corridor scenario.

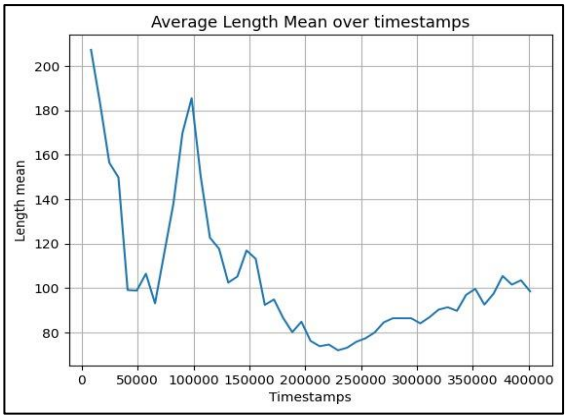


Figure 4.13 Average length mean over timestamps.

Figures in 4.11, 4.12, and 4.13 show the graphs for kills, average reward and average episode length as the Agent is trained over increasing number of timestamps from 0 to 400000 using PPO algorithm. AI agents using the A2C algorithm for training performed much better. The graphs present the average score trends of such agents in the deadly corridor VizDoom scenario, highlighting fluctuations and important patterns in their performance. The average score ranges from 0 to 1000 on the y-axis, with episode numbers from 0 to 5000 on the x-axis. Statistically, the average score demonstrates an overall upward trend as episodes progress, indicating continuous learning and improvement by the agents. The optimum reward achieved was 800, showing the peak performance level reached by the agents. However, there are notable fluctuations in the scores, with a significant change at episode 3000, suggesting a challenging phase in the agents' training. The varying distances between the blue lines on the graph emphasize performance diversity among training runs, with some consistently outperforming others. These fluctuations and patterns provide valuable insights into the effectiveness of the A2C algorithm in training agents for the deadly corridor VizDoom scenario, showcasing the learning dynamics and performance variability among the agents.

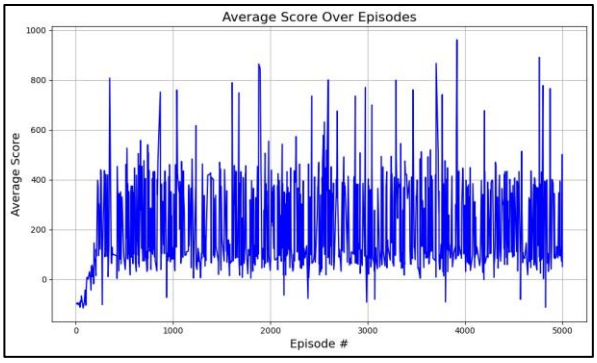


Figure 4.14 Average Cumulative Reward over episodes in deadly corridor using A2C algorithm.

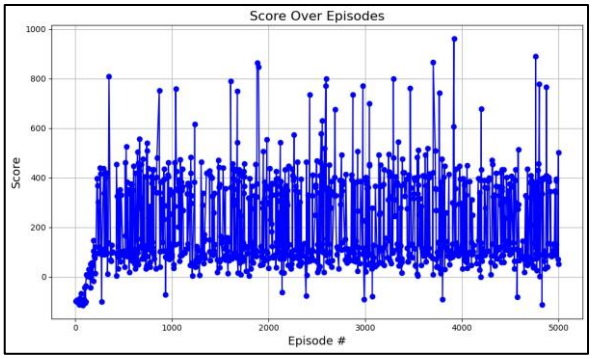


Figure 4.15 Total reward obtained by the agent using the A2C algorithm in the deadly corridor.

Overall, the agent improved on health-gathering, doing it steadily and stepwise; at ~5000 steps, the performance arc moved towards ~71%. Similarly, the agent's metrics such as score, ammo-left, and kills improved gradually stepwise, which is presented in Fig. 4.14, and Fig.4.15.

4.3 Basic Scenario

This section presents the displays and learning charts of the agents trained in the 'Basic' scenario. PPO proves to be effective and reasonable for training agents in the basic scenario of VizDoom for optimal performance. Fig.4.16 shows a gradual decrease in the amount of ammo left as the steps progress from 0 to 600. This downward trend suggests that the ammo is being consumed or utilized over time in the scenario being analyzed. Fig.4.17 shows a clear upward trajectory in the average reward mean, starting from a negative value (around -100) and reaching a positive value (around 25) by the 75,000th timestamp. This suggests positive reinforcement learning. As the agent takes actions that navigate well and defeat the monster, it receives positive rewards, motivating it to learn these desirable behaviors. The increase isn't linear, it appears the agent is gradually improving its performance. This is a typical learning curve observed in reinforcement learning tasks. There might be initial exploration and mistakes, leading to negative rewards initially. As the agent learns the environment and refines its strategy, the rewards become progressively more positive.

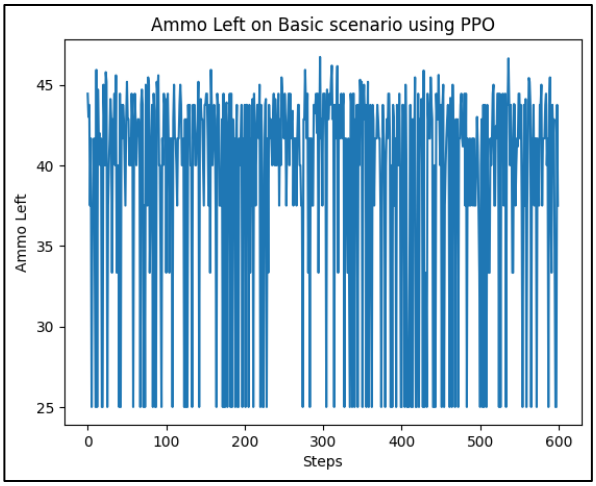


Figure 4.16 Ammo left on Basic Scenario using PPO.

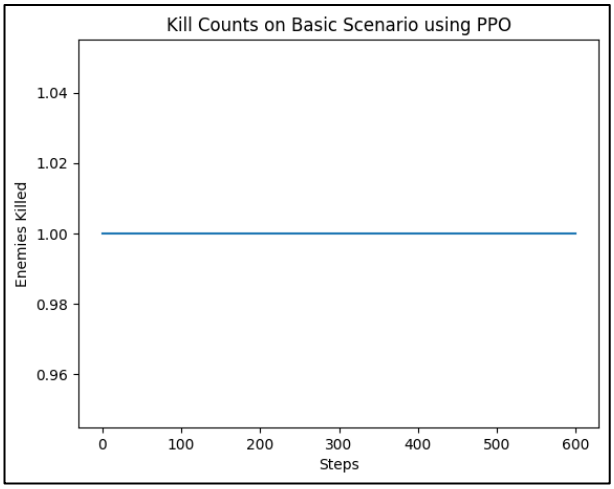


Figure 4.17 Kills on basic Scenario using PPO.

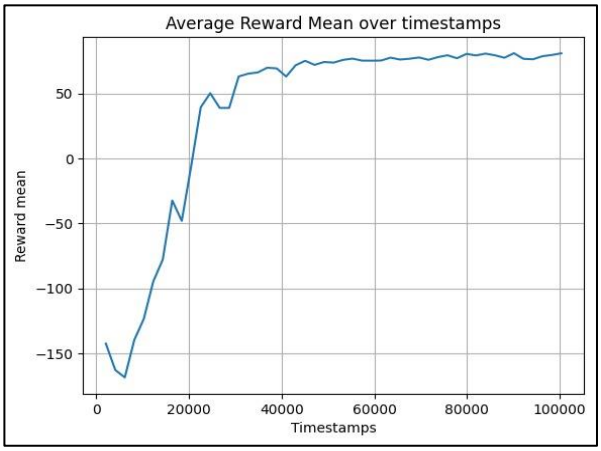


Figure 4.18 Average reward graph during training using PPO in the basic Scenario.

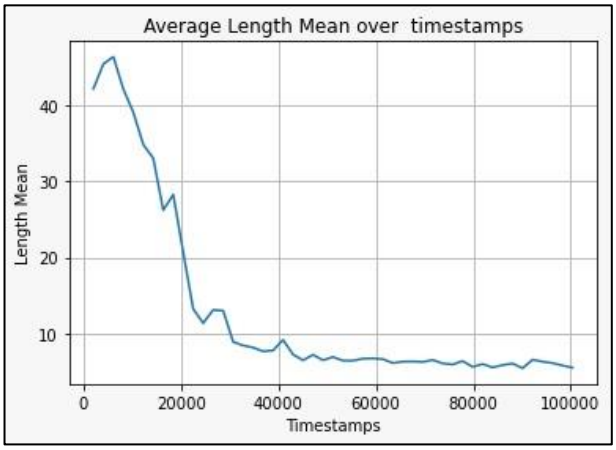


Figure 4.19 Average episode length during training using PPO in a basic scenario.

Figure 4.16 reflects the information of Ammo as the Agent is trained on the basic scenario using PPO, with time it just uses the only bullet to fire hence showing plenty of Ammo left. Figure 4.17 is the number of enemies killed on basic scenario using PPO as the agent is trained which is constant as there in only one enemy in this map that is killed. Figure 4.18 shows the Average reward that the agent gains through hitting the reward shaping equation in the basic scenario using PPO that through different phases increases with time and experience. Figure 4.19 is about the length of the episode as the agent is trained in the basic scenario using PPO which gets shorter as the agent gets smarter and quicker.

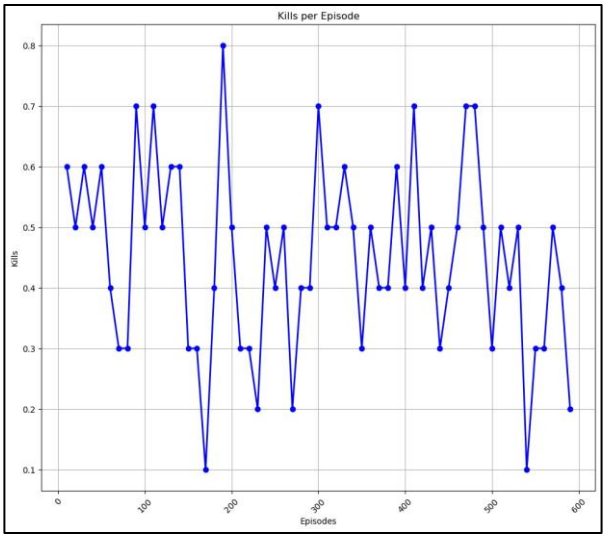


Figure 4.20 Kills vs episodes graph using DDDQN in a basic scenario.

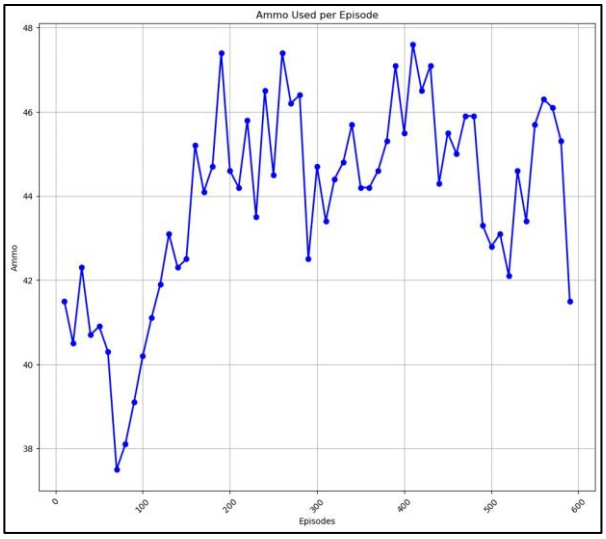


Figure 4.21 Ammo vs episodes graph using DDDQN in a basic scenario.

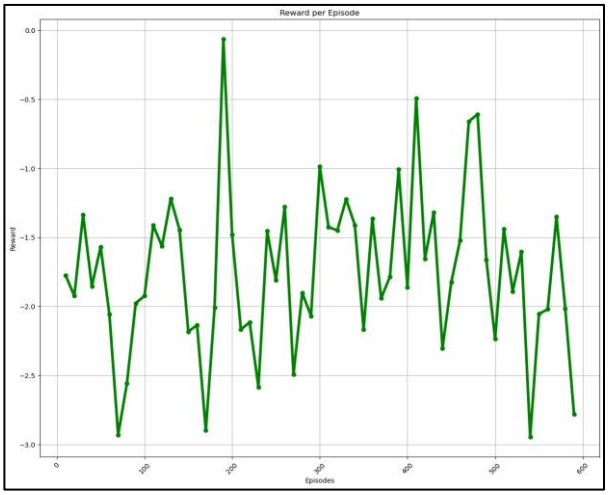


Figure 4.22 Reward graph trained using DDDQN in a basic scenario.

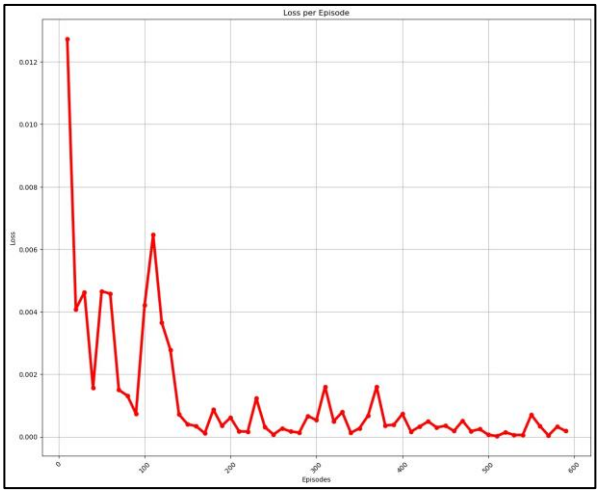


Figure 4.23 Loss graph trained using DDDQN in a basic scenario.

DDDQN does not prove to be very effective for training the agent in the basic scenario. Fig.4.21 shows an upward trend in the average ammo used per episode. It starts at around 38 ammos used per episode and increases to around 46 ammos used per episode by the end of the graph. This upward trend suggests that the agent might be becoming less efficient in its ammo usage as the training progresses. The agent might be exploring more and taking less optimal actions that require more ammo to complete. Although Fig. 4.22 shows a clear upward trend in the average reward per episode. It starts at around -5 and increases to around 75 by the 1000th episode. There appears to be some variability in the reward throughout training. This is common in reinforcement learning, as the agent explores different actions and learns from their experiences. As for figure 4.20 it shows how the agent is performing the kills as he is trained over a number of episodes using DDDQN in basic map and figure 4.23 shows the loss graph which is decreasing over time, meaning an improved agent performance as it is trained over larger number of episodes.

4.4 Comparison with Prior Works

We have compared our approach with Justesen et al. and Paulo Bruno et al., who focused on A2C and DQN, respectively. The comparison Table 4.1 is shown below to see the details. To assess performance, we evaluated scenarios consisting of Basic, Defend the Center, and Deadly-Corridor, and considered metrics such as ammo, kills, and score. Our approach in DDDQN and A2C utilized experience replay, while Justesen et al. introduced RoE and Paulo Bruno et al. employed experience replay and dropout. Tuning hyperparameters, such as γ and α, played a crucial role in achieving optimal results. Overall, A2C demonstrated strong performance, while PPO and DDDQN yielded mixed outcomes. The Deadly Corridor scenario presented significant challenges, indicating the need for further investigation. Both experience replay and RoE exhibited potential with varying effects.

Table 4.1 Comparison of our work with existing similar literature

Factors	**Our Work**	**Niels Justesen al. Work [43]**	**Paulo Bruno et al. Work [44]**
Algorithms	PPO, DDDQN, A2C	A2C, A2C+RoE	Deep Q-Learning
Scenarios	Basic, Defend the center, Deadly Corridor	Defend the center, Deadly Corridor, Health Gathering	Deathmatch, Deadly Corridor, Health Gathering, My Way Home
Matrices and Variables	Kills, Ammo, Score	Score, Kills, medikits, and armor pickups	Score
Technique (approach)	Experience Replay	Rarity of Event (ROE)	Experience Replay, Dropout
Hyperparameters Values	ϒ=0.99, α=0.0001 to 0.01 Batch size=32	ϒ=0.99, α=0.0007 Batch size=64	ϒ=0.99, α=0.0001 Batch size = 64, greedy policy 1.0 to 0.1
Frames/Action	4	4	8
Color to Greyscale	108x60	80x80	64x48
Mean Score	DTC: PPO = ~1400 DDDQN = ~1300 A2C = ~3100 Deadly Corridor: PPO = ~1000 DDDQN = ~800 Basic: PPO = ~85 DDDQN = ~20	Death Match: 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: 518.87

5. Conclusion

In this project, we have worked on different VizDoom scenarios to efficiently enhance agents' performance. We have utilized three different reinforcement learning algorithms that include PPO, DDDQN, and A2C. The scenarios include a variety of complexity levels to enhance agents' performance, The scenarios include basic, defend the center, and deadly corridor.

Our results demonstrate how well these algorithms for reinforcement learning can handle a variety of dynamic and changing settings. Every algorithm showcased its advantages in particular scenarios, emphasizing how crucial it is to choose an algorithm depending on the requirements of the given task. PPO demonstrated exceptional stability and resilience in a range of scenarios, and it was especially good at navigating through different VizDoom scenarios. Its steady performance was aided by its capacity to mitigate significant policy revisions while optimizing policies.

With the addition of target networks and a dual Q-network, DDDQN performed exceptionally well in scenarios requiring strategic decision-making, like defending the center. Interestingly, in our study, we found that PPO works equally well for the training of our agents in the deadly corridor scenario along with A2C where we have made quite a progress in achieving a higher average score and reward. Navigating complex circumstances required the ability to optimize the Q-value estimation and efficiently handle temporal dependencies. In addition to offering a thorough assessment of each algorithm's unique capabilities, our study's integration of various algorithms highlighted the value of algorithmic diversity in addressing a range of problems in the VizDoom context.

6. Future Work

Certainly! In the realm of RL, there's a constant quest to improve agent performance in complex environments. The avenues for further research outlined in the study suggest several strategies to enhance this performance.

Firstly, algorithm hybridization proposes combining different algorithms to create new, potentially more effective approaches. This could involve taking the strengths of one algorithm and pairing it with the strengths of another to address weaknesses or capitalize on synergies.

Secondly, hyperparameter tuning involves fine-tuning the parameters of each algorithm to maximize its performance across various scenarios. This process aims to uncover subtle adjustments that can significantly boost efficiency and effectiveness.

Thirdly, scenario-specific customization suggests tailoring algorithms to the specific characteristics of different environments. By understanding how algorithms interact with environmental factors, researchers can optimize them for better adaptability and performance.

Fourthly, real-time adaptation mechanisms would enable agents to adjust their strategies dynamically during training in response to changing conditions. This adaptability could be crucial for navigating dynamic environments where conditions can shift unpredictably.

Lastly, transfer learning involves using knowledge gained from one scenario to help learn faster in another scenario. This approach aims to reduce the time and computational resources needed to train agents in new environments by capitalizing on previously acquired knowledge. This aims to advance the capabilities of RL agents in handling dynamic and complex scenarios, laying the foundation for more robust and adaptable artificial intelligence systems.

7. References

[1] Khan, A., Shah, A. A., Khan, L., Faheem, M. R., Naeem, M., & Chang, H. T. (2024). Using VizDoom Research Platform Scenarios for Benchmarking Reinforcement Learning Algorithms in First-Person Shooter Games. IEEE Access.

[2] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D. & Riedmiller, M. (2013), Playing Atari with deep reinforcement learning".

[3] Justesen, N., Bontrager, P., Togelius, J., & Risi, S. (2019). Deep learning for video game playing. IEEE Transactions on Games, 12(1), 1-20.

[4] Bergdahl, J., Gordillo, C., Toll mar, K., & Gisslén, L. (2020, August). Augmenting automated game testing with deep reinforcement learning. In 2020 IEEE Conference on Games (CoG) (pp. 600-603). IEEE.

[5] Khan, A., Naeem, M., Asghar, M. Z., Din, A. U., & Khan, A. (2020). Playing first-person shooter games with machine learning techniques and methods using the VizDoom Game-AI research platform. Entertainment Computing, 34, 100357.

[6] Sutton, R. S. & Barto, A. G. (2018), Reinforcement Learning: An Introduction, second edn, The MIT Press.

[7] S. Albawi, T. A. M. & Al-Zawi, S. (2017), 'Understanding of a convolutional neural network'.

[8] François-Lavet, V., Henderson, P., Islam, R., Bellemare, M. G., & Pineau, J. (2018). An introduction to deep reinforcement learning. Foundations and Trends® in Machine Learning, 11(3-4), 219-354.

[9] Fakoor, R., Chaudhari, P., Soatto, S., & Smola, A. J. (2019). Meta-q-learning. arXiv preprint arXiv:1910.00125.

[10] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

[11] Achiam, J. (2020). Spinning up documentation, release.

[12] Li, R., Wang, C., Zhao, Z., Guo, R., & Zhang, H. (2020). The LSTM-based advantage of actor-critic learning for resource management in network slicing with user mobility. IEEE Communications Letters, 24(9), 2005-2009.

[13] Wang, J. X., Kurth-Nelson, Z., Kumaran, D., Tirumala, D., Soyer, H., Leibo, J. Z., ... & Botvinick, M. (2018). Prefrontal cortex as a meta-reinforcement learning system. Nature Neuroscience, 21(6), 860-868.

[14] Brockman, G., Cheung, V., Pettersson, L., Schneider, J., Schulman, J., Tang, J. & Zaremba, W. (2016), 'OpenAI gym'.

[15] Kempka, M., Wydmuch, M., Runc, G., Toczek, J., & Jaśkowski, W. (2016, September). VizDoom: A doom-based AI research platform for visual reinforcement learning. In the 2016 IEEE conference on computational intelligence and games (CIG) (pp. 1-8). IEEE.

[16] Bakhanova, M., & Makarov, I. (2021). Deep reinforcement learning in VizDoom via dqn and actor-critic agents. In Advances in Computational Intelligence: 16th International Work-Conference on Artificial Neural Networks, IWANN 2021, Virtual Event, June 16–18, 2021, Proceedings, Part I 16 (pp. 138-150). Springer International Publishing.

[17] Asada, M., Uchibe, E., Noda, S., Tawara Sumida, S., & Hosoda, K. (1994). A vision-based reinforcement learning for coordination of soccer-playing behaviors. In Proceedings of AAAI-94 Workshop on AI and A-life and Entertainment (pp. 16-21).

[18] Wydmuch, M., Kempka, M., & Jankowski, W. (2018). VizDoom competitions: Playing doom from pixels. IEEE Transactions on Games, 11(3), 248-259.

[19] Poznan University (02/02/2024). Visual Doom AI competition 2018 – Single Track (1)

[20] Poznan University (02/02/2024). Visual Doom AI competition 2018 – Multi Track (2)

[21] Geisler, B. J. (2002). An Empirical Study of Machine Learning Algorithms Applied to Modeling Player Behavior in a" First Person Shooter" Video Game (Doctoral dissertation, University of Wisconsin-Madison).

[22] McPartland, M., & Gallagher, M. (2010). Reinforcement learning in first person shooter games. IEEE Transactions on Computational Intelligence and AI in Games, 3(1), 43-56.

[23] Smith, T. C., & Miles, J. (2014). Continuous and Reinforcement Learning Methods for First-Person Shooter Games. GSTF Journal on Computing (Joc), 1(1).

[24] Jeering, A., Bein, D., & Verma, A. (2019, January). Comparison of deep reinforcement learning approaches for intelligent game playing. In 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC) (pp. 0366-0371). IEEE.

[25] Amato, C., & Shani, G. (2010, May). High-level reinforcement learning in strategy games. In AAMAS (Vol. 10, pp. 75-82).

[26] Moysen, J., & Giupponi, L. (2018). From 4G to 5G: Self-organized network management meets machine learning. Computer Communications, 129, 248-268.

[27] Bellemare, M. G., Naddaf, Y., Veness, J., & Bowling, M. (2013). The arcade learning environment: An evaluation platform for general agents. Journal of Artificial Intelligence Research, 47, 253-279.

[28'] Dargazany, A. (2021). DRL: Deep Reinforcement Learning for Intelligent Robot Control--Concept, Literature, and Future. arXiv preprint arXiv:2105.13806.

[29'] Liu, X. Y., Yang, H., Chen, Q., Zhang, R., Yang, L., Xiao, B., & Wang, C. D. (2020). FinRL: A deep reinforcement learning library for automated stock trading in quantitative finance. arXiv preprint arXiv:2011.09607.

[30'] Abdellatif, A. A., Mhaisen, N., Chkirbene, Z., Mohamed, A., Erbad, A., & Guizani, M. (2021). Reinforcement learning for intelligent healthcare systems: A comprehensive survey. arXiv preprint arXiv:2108.04087.

[31'] Liao, J., Liu, T., Tang, X., Mu, X., Huang, B., & Cao, D. (2020). Decision-making strategy on highway for autonomous vehicles using deep reinforcement learning. IEEE Access, 8, 177804-177814.

[32'] Sharma, A. R., & Kaushik, P. (2017, May). Literature survey of statistical, deep and reinforcement learning in natural language processing. In 2017 International conference on computing, communication and automation (ICCCA) (pp. 350-354). IEEE.

[33'] del Real Torres, A., Andreiana, D. S., Ojeda Roldán, Á., Hernández Bustos, A., & Acevedo Galicia, L. E. (2022). A review of deep reinforcement learning approaches for smart manufacturing in industry 4.0 and 5.0 framework. Applied Sciences, 12(23), 12377.

[34'] Equihua, J., Beckmann, M., & Seppelt, R. (2024). Connectivity conservation planning through deep reinforcement learning. Methods in Ecology and Evolution.

[35'] Souchleris, K., Sidiropoulos, G. K., & Papakostas, G. A. (2023). Reinforcement learning in game industry—Review, prospects and challenges. Applied Sciences, 13(4), 2443.

[36] Khan, A., Shah, A. A., Khan, L., Faheem, M. R., Naeem, M., & Chang, H. T. (2024). Using VizDoom Research Platform Scenarios for Benchmarking Reinforcement Learning Algorithms in First-Person Shooter Games. IEEE Access.

[37] Vitalii Sopov, ilya Markarov Reward Shaping for Deep Reinforcement Learning in VizDoom.

[38] Schulman, J., Levine, S., Abbeel, P., Jordan, M., & Moritz, P. (2015, June). Trust region policy optimization. In International conference on machine learning (pp. 1889-1897). PMLR.

[39] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., & Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347

[40] Mehdi Boubnan and Ayman Chauki 2018. Deep reinforcement learning applied to doom.

[41] Sanghi, N., & Sanghi, N. (2021). Deep Q-Learning. Deep Reinforcement Learning with Python: With PyTorch, TensorFlow and OpenAI Gym, 155-206.

[42] Alibabaei, K., Gaspar, P. D., Assunção, E., Alirezazadeh, S., Lima, T. M., Soares, V. N., & Caldeira, J. M. (2022). Comparison of on-policy deep reinforcement learning A2C with off-policy DQN in irrigation optimization: A case study at a site in Portugal. Computers, 11(7), 104.

[43] Justesen, N., & Risi, S. (2018, August). Automated curriculum learning by rewarding temporally rare events. In 2018 IEEE Conference on Computational Intelligence and Games (CIG) (pp. 1-8). IEEE.

[44] Serafim, P. B. S., Nogueira, Y. L. B., Vidal, C., & Cavalcante-Neto, J. (2017, November). On the development of an autonomous agent for a 3d first-person shooter game using deep reinforcement learning. In 2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames) (pp. 155-163). IEEE.