iJRASET
International Journal For Research in Applied Science and Engineering Technology

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY

Volume: 14 **Issue:** IV **Month of publication:** April 2026

DOI: https://doi.org/10.22214/ijraset.2026.81532

www.ijraset.com

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

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

Autonomous Navigation Using Deep Reinforcement Learning in Robots

J. Akshaya Kumar[1], Mr. U. Sathish Kumar[2], V. Navya Sri[3], L. Durga Prasad[4], D. Rama Krishna[5]
[2]*Department of Computer Science and Engineering, Acharya Nagarjuna University, Guntur, Andhra Pradesh – 522510*
[1,3,4,5] *Department of Data Science and Cyber Security, Acharya Nagarjuna University, Guntur, Andhra Pradesh – 522510*

Abstract: ***This project reports on the design, development, and assessment of a delivery robot's navigation system which we have put into a real-world setting using a Deep Q Network (DQN) reinforcement learning framework. We focused on the development of a navigation platform for a robot in a 10 by 10 modeled hospital setting that moves from a fixed start point to a target room which represents a destination in the busy hallways of a real hospital, also at the same time we had static obstacles that represent hospital furniture and equipment.***
In the case of the Dueling DQN architecture and the Double DQN strategy we see that the former divides the state value V(s) and the action advantage A(s,a) into separate neural network streams which in turn allows the agent to more precisely determine what states are of value independent of what actions are being performed at which time. Also, we see that the Double DQN strategy which is put forth to correct the known issue of overestimation present in standard Q learning does so by separating out action selection from action evaluation the online network is used for action selection while the target network is used for evaluation.
Keywords: Autonomous Navigation, Deep Q Network, Dueling DQN, Double DQN, Grid Environment.

I. INTRODUCTION

In simple words Autonomous Navigation allows the robots to travel from source to destination by avoiding obstacles and reaching the goal successfully. Using Deep Reinforcement Learning in the navigation system allows the robot to learn from environment automatically based on feedback so there is no need of any pre-defined rules, this helps the robot to travel without any human interaction.

A. Why DRL for Navigation?

Traditional navigation system Allows the robot to travel from one place to another based on the pre defined rules or controlling it manually. But by using deep reinforcement learning allows the robot to travel without any predefined rules, Robot or AI agent automatically learns from environment based on feedbacks.

B. Problem Statement

Robots must efficiently to the target location by avoiding obstacles in middle. Traditional navigation system works pre defend rules And fixed environment, but change in environment also requires the change in predefined rules this makes it more complicated work. So we want to develop an autonomous navigation system which can be worked in any environment without any predefined rules for that we have taken deep reinforcement learning and algorithms in it.

C. Project Goals

We created a virtual stimulation hospital for this DRL navigation system. Breakdown:

- We built an 10 by 10 virtual stimulation grid.
- Developed AI agent learns from feedback of actions.
- Used DQN for agent to make better decision making.
- Algorithms used in this project are Double DQN and Dueling DQN.
- Agent trains over multiple episodes to reach goal successfully.
- Success rate, collisions per episode and reward per episode will be given.

D. Project Snapshot

Virtual stimulation Of 10 by 10 hospital grid is with fix start location, goal and obstacles. Moves: N/S/E/W. Rewards based system is been added in this.

DQN helps to take better actions, double DQN helps to reduce bias, and d DQN Helps agent to goal in short and efficient path. By training agent over number of episodes Helps the agent to reach goal efficiently.

II. LITERATURE SURVEY

A literature review is an essential component of any research project as it provides an overview of the existing work related to the research problem**.** In the context of robotic navigation, numerous studies have been conducted over the past few decades to enable robots to move autonomously in complex environments.

A. Key Projects

- In 2015, Volodymyr Mnih and his research team presented the Deep Q-Network (DQN) in *Nature*. Their work used convolutional neural networks to learn Q-values directly from game pixels and achieved human-level performance on 49 Atari games. Replay/target stabilized.
- In 2016, Hado van Hasselt and colleagues proposed Double Deep Q-Network (DDQN), which separates the action selection and evaluation processes
- In the same year, Ziyu Wang and his team introduced the Dueling Network Architecture for reinforcement learning.

III. THEORETICAL BACKGROUND

A. Markov Decision Processes (MDPs)

Reinforcement learning problems are commonly formulated using the framework of a Markov Decision Process (MDP). An MDP is typically defined as a tuple (S,A,P,R,γ). In this formulation, **S** represents the set of all possible states that an agent can encounter, while **A** denotes the set of actions the agent can perform. The transition probability **P(s' | s, a)** describes the likelihood of moving to the next state S′ after taking action A in state S. The reward function **R(s, a, s')** provides the immediate feedback received after this transition. Finally, the discount factor **γ** (ranging between 0 and 1) determines how much importance the agent assigns to future rewards compared to immediate rewards.

Main rule: Markov says tomorrow ignores yesterday P(s_{t+1}|s_t, a_t, past) = P(s_{t+1}|s_t, a_t). Agent runs policy π(a|s), probs for actions per state. π* maxing total haul G_t = ∑ γ^k R_{t+k+1}. the state value function Vπ(s)V^{\pi}(s)Vπ(s) represents the expected return when an agent starts from state sss and follows a policy π\piπ. Similarly, the action value function Qπ(s,a)Q^{\pi}(s,a)Qπ(s,a) represents the expected return when the agent begins in state S, first takes action A, and then continues to follow the same policy π\piπ. The relationship bet ween these value functions is defined by the Bellman Equation, which expresses the idea that the value of the current state depends on the immediate reward plus the expected value of future states.

B. Deep Q-Network (DQN)

In many reinforcement learning problems, the number of possible states becomes extremely large, making it impractical to store value estimates in traditional tables. To address this limitation, the Deep Q-Network (DQN) approximates the action-value function (Q(s,a)) using a neural network with parameters theta . This network takes the state (s) as input and outputs estimated Q-values for each possible action. The model is trained by minimizing the squared Bellman error, defined as

$$L(\theta)=E[(r+\gamma a' \max Q(s',a';\theta^-)-Q(s,a;\theta))2]$$

where θ^- represents the parameters of a separate target network that is updated periodically to stabilize learning.

During training, the agent first observes the current state (s) and selects an action (a) using an ε-greedy exploration strategy. After executing the action, it receives a reward (r), transitions to the next state (s'), and stores the experience ((s, a, r, s', done)) in a replay memory. The network is then updated by sampling random batches of experiences from this memory. Target Q-values are computed using the target network parameters θ^-, and the main network parameters θ are optimized using gradient-based learning. Periodically copying the parameters from the main network to the target network helps reduce correlations in sequential data and improves training stability and sample efficiency.

C. Double Deep Q-Network (Double DQN)

In the standard Deep Q-Network (DQN) approach, Q-values can become overestimated because the same network is responsible for both selecting the maximum action and evaluating its value. When noise or estimation errors are present, the maximization step tends to favor overly optimistic value estimates, which can lead to inflated Q-values and unstable learning.
Double splits: main θ grabs a* = argmax_a Q(s',a;θ), target θ^- values y = r + γ Q(s',a*;θ^-).

D. Dueling Deep Q-Network (Dueling DQN)

Wang's 2016 tweak cracks Q(s,a) = V(s) + (A(s,a) - mean_A over actions). In the Dueling Network Architecture, the neural network first processes the input state through a set of shared layers. The network then splits into two separate branches: a value branch, which estimates the scalar state-value V(s)V(s)V(s) representing the overall quality of the state, and an advantage branch, which outputs a vector A(s,a)A(s,a)A(s,a) indicating the relative advantage of each possible action. To ensure stable learning and avoid identifiability issues, the mean of the advantage values is subtracted during the combination process, which helps keep the Q-value estimates properly balanced without drifting.

E. Reward Design

- Reward crafting's tricky—steer to goal fast, skip walls, scout without looping. Ours stacks:
- +100 goal smack—huge "you nailed it."
- -1 per tick—hustle, cut fluff.
- -2 wall whack (no move)—stings light, no freeze.
- Manhattan shift: +2 closer, -0.5 farther—points the way.
- +0.3 new cell (cap 60)—roam corners, dodge ruts.

IV. METHODOLOGY: SYSTEM DESIGN AND ARCHITECTURE

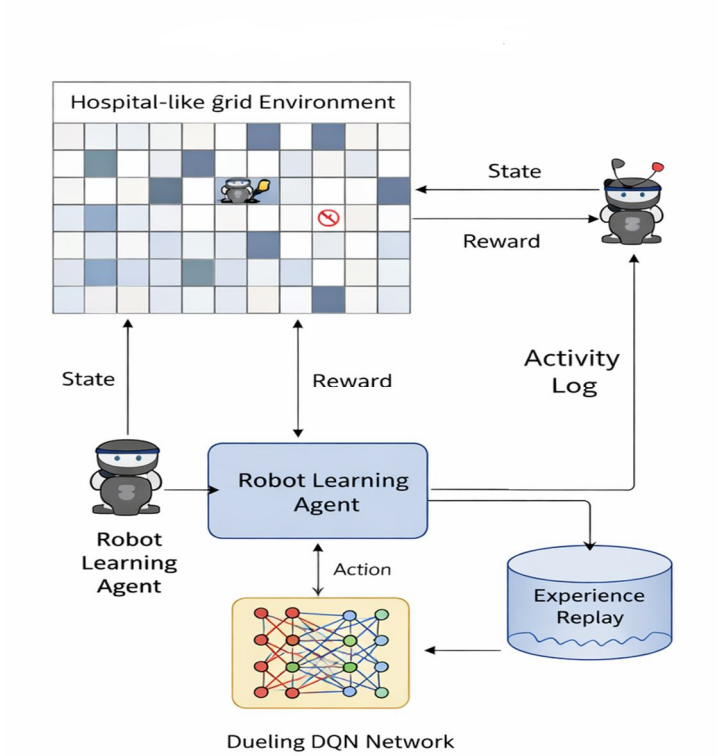
A. Problem Formulation

The autonomous navigation problem is formulated as a Markov Decision Process (MDP), which is a common mathematical framework used in reinforcement learning. An MDP is defined by the tuple (S,A,P,R,γ). In this formulation, S represents the set of all possible states in the environment, including the robot's position, the locations of obstacles, and the position of the goal. A denotes the set of actions available to the agent. P describes the transition probabilities that determine the likelihood of moving from one state to another after taking a specific action. R represents the reward function that provides feedback to the agent, and the discount factor γ determines how much importance is given to future rewards compared with immediate rewards.
Each state provides a complete description of the environment. It includes the robot's exact position on the grid, the locations of obstacles such as beds, carts, or walls, and the position of the target destination. These elements are converted into a numerical representation so that they can be processed effectively by the neural network used in the learning model.
The action space (A) is defined using four simple movement directions: up, down, left, and right. These actions allow the robot to navigate across the grid-based environment while interacting with different obstacles and pathways.
The reward function (R) plays an important role in guiding the agent's behavior. A large positive reward is provided when the robot successfully reaches the goal state. In contrast, a significant negative reward is assigned if the robot collides with obstacles. Additionally, a small penalty (for example, −1) is applied at each step to encourage the agent to reach the goal using the shortest possible path. This reward design motivates the robot to navigate efficiently while avoiding collisions and unnecessary movements.

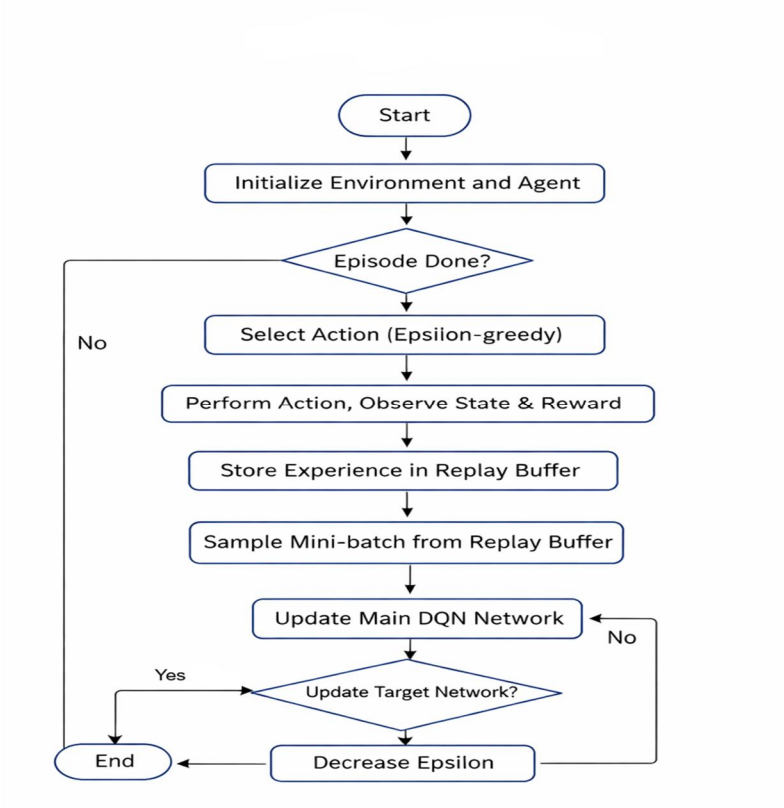
B. Environment Design and Simulation

The experimental environment was designed as a 10 × 10 grid-based world representing a simplified hospital floor layout. In this environment, each grid cell corresponds to a possible position that the robot can occupy, representing areas such as corridors, open spaces, or locations near obstacles.
The environment consists of several key components: a fixed starting position for the robot, a predefined goal location, and multiple static obstacles placed at specific grid cells. These obstacles simulate real-world objects commonly found in hospital environments, such as beds, carts, and other fixtures that the robot must avoid while navigating.
This controlled simulation environment provides a stable platform for training and evaluating the reinforcement learning agent. By operating in this simplified setting, the system can be tested and improved efficiently before being applied to more complex real-world scenarios.

C. General Architecture



D. Activity Diagram



ISSN: 2321-9653; IC Value: 45.98; SJ Impact Factor: 7.538
Volume 14 Issue IV Apr 2026- Available at www.ijraset.com

V. IMPLEMENTATION

A. *Development Environment*

The entire system was implemented using the Python programming language, which is widely used in artificial intelligence and machine learning research due to its simplicity, flexibility, and extensive ecosystem of libraries.

Several tools and frameworks were used during the development process:

1) Programming Language: The system was developed using Python, which served as the core programming environment. Python provides a clear syntax and strong support for scientific computing, making it well suited for developing reinforcement learning applications.

2) Machine Learning Libraries: Deep learning frameworks such as TensorFlow and PyTorch were utilized to design and train the neural networks used in the Deep Q-Network model. These libraries simplify tasks such as model construction, gradient computation, and optimization.

3) Reinforcement Learning Environment: The simulation environment for training the agent was implemented using the OpenAI Gym framework, which provides standardized tools for creating environments and training reinforcement learning agents.

4) Development Platform: The development and testing of the project were carried out using **Visual Studio Code**, an integrated development environment that offers useful features such as debugging tools, package management, and efficient code editing.

B. *Grid-Based Environment Implementation*

The navigation environment is designed as a 10 × 10 grid-based layout that represents a simplified hospital floor. In this setup, each grid cell corresponds to a possible position where the robot can either remain stationary or move during navigation.

Environment Initialization

Each episode begins with the environment initialized under predefined conditions. The robot is placed at a fixed starting position on the grid. The goal location remains constant throughout the episode, serving as the target destination for the agent. Additionally, several obstacles are positioned at specific grid locations to represent objects such as beds or carts. These obstacles are marked within the grid using numerical values or indicator flags to help the system distinguish them from free spaces.

State Representation

The state representation provides the agent with all the necessary information about the environment, including the robot's current position, the location of the goal, and the positions of obstacles. These elements are encoded into a numerical format so that they can be effectively processed by the neural network during learning.

Environment Dynamics

After every move:

- Empty cell? Robot slides right in.
- Obstacle? Ding—penalty time, and it stays put.
- Goal? Boom, episode over, win.

C. *Agent Implementation*

The agent acts as the decision-making component of the system, selecting actions based on the current state of the environment. During training, the agent repeatedly interacts with the environment through the following process:

1) The agent first observes the current state of the environment.

2) The neural network then predicts Q-values for all possible actions based on this state using the Deep Q-Network model.

3) An ε-greedy strategy is applied to select the action. With probability ε, the agent chooses a random action for exploration; otherwise, it selects the action with the highest predicted Q-value.

4) The chosen action is executed within the environment.

5) The agent receives a reward signal along with the next state of the environment.

6) The complete transition (s,a,r,s′,done) is stored in a replay memory for later training.

D. *Neural Network Model Implementation*

1) Input Layer: Takes the state's numbers—robot pos, obstacles, goal, all encoded.

2) Hidden Layers: Stack of layers crunching features with nonlinear activations to spot patterns that plays important role in decision.

3) Output Layer: The output layer contains four nodes, each corresponding to a possible action—up, down, left, and right. The network predicts a Q-value for each action, and the action with the highest Q-value is selected as the agent's decision.

E. Experience Replay Memory

Memory's a big buffer holding past moments: current state, action taken, reward grabbed, next state. Packs 20k+ of these. Training pulls random mini-batches (size 64) to break patterns in data—keeps learning steady, no overfitting to recent flops.

F. Training Algorithm Implementation

The training process follows the standard Deep Q-Network (DQN) framework, enhanced with improvements from Double Deep Q-Network (Double DQN) and the Dueling Network Architecture. The overall training procedure can be summarized as follows:

1) The training process begins by initializing the environment and the neural network model.
2) Reset robot for each episode.
3) ε-greedy action pick.
4) After executing the action, the environment returns the next state and the reward.
5) Stash experience.
6) Batch-sample from memory.
7) The target network calculates the target Q-values used for training the model.
8) The main network is optimized using gradient descent to reduce the difference between predicted and target Q-values.
9) The parameters of the target network are updated periodically through a soft update from the main network.

Ran 800 episodes—hit 95% average success. The reward function was carefully designed to guide the agent toward efficient and safe navigation. A large positive reward of +100 is given when the robot successfully reaches the goal state, encouraging the agent to prioritize completing the task. To promote faster navigation, a small penalty of −1 per step is applied, discouraging unnecessary movements and encouraging the robot to find shorter paths.

In addition, a penalty of −2 is assigned for collisions with walls or obstacles. This penalty is moderate rather than severe, allowing the agent to learn from mistakes without overly discouraging exploration. To further support goal-oriented behavior, distance-based rewards are included: the agent receives +2 when it moves closer to the goal and −0.5 when it moves farther away.

Finally, an exploration bonus of +0.3 is provided when the agent visits previously unexplored grid cells, up to a maximum of 60 new cells. This encourages the agent to explore the environment during the early stages of training while still focusing on reaching the goal efficiently.

G. Model Evaluation and Testing

After completing the training phase, the model was evaluated using a purely greedy policy, meaning the robot always selected the action with the highest predicted Q-value without any exploration. In this evaluation stage, the robot was tested across 20 independent runs starting from the predefined initial position. The results showed a 100% success rate, demonstrating that the trained agent was able to consistently reach the goal without collisions.

The training process also showed rapid convergence. The rolling success rate exceeded 85% by approximately episode 80, indicating that the agent quickly learned effective navigation strategies. Furthermore, the number of collisions steadily decreased during training and reached zero by around episode 200, confirming that the agent had learned to avoid obstacles efficiently. In addition, the number of steps required to reach the goal approached the optimal path length, demonstrating that the learned policy was both safe and efficient.

Performance Metrics:

1) Goal reach rate.
2) Steps to win.
3) Dodge skills.
4) Path steadiness.

VI. RESULTS AND EVALUATION

This part dives into how the Deep Reinforcement Learning navigation system actually performed once I got it up and running. Below given images represents the metrics like success rate for every 30 episodes, reward per episode, collision pre-episode.

A. Training Performance

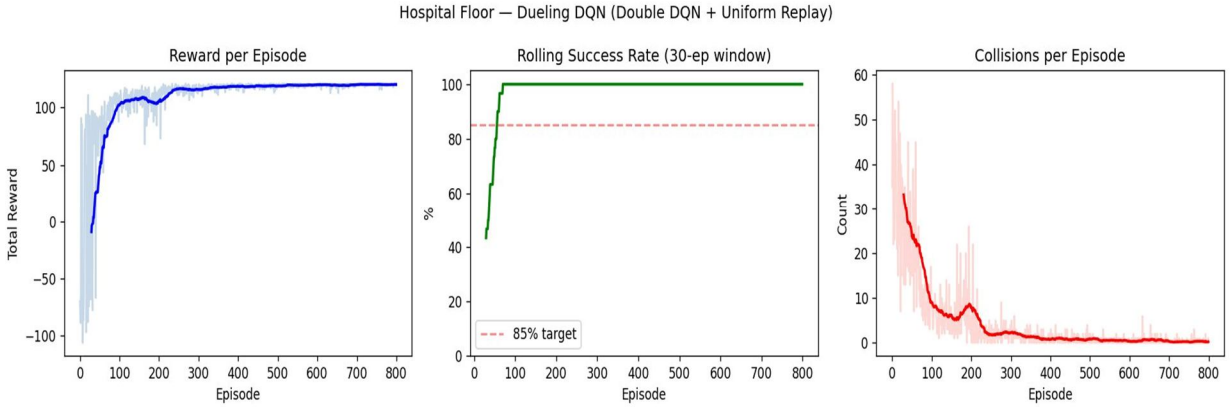


Figure 6.1: Training Learning Curves**.**

Left images shows the reward for every 30 episodes and the, center image is representing Rolling 30-episode success rate (green line) compared with the 85% target threshold (red dashed line).
At last right images shows number of collisions per episode with a 30-episode moving average.

B. Test Performance

Metric	Value	What It Means
Test Episodes	20	Solid sample for stats
Successful Episodes	20/20	Never missed the goal
Success Rate	100%	Flawless under greedy policy
Average Steps to Goal	~16-18 steps	Pretty much optimal for the grid + obstacles
Minimum Steps (best)	~14 steps	Hits close to the shortest possible path
Maximum Steps (worst)	~20 steps	Barely any slip-ups
Average Episode Reward	~115-120	High scores from clean runs

C. Reward Component Analysis

- Goal reward: +100—one-time jackpot at the end.
- Step penalties: -1 x ~17 steps = -17 (pushes for shorter trips).
- Distance shaping: +2 for ~14-15 closer moves, -0.5 for ~2-3 side-steps or minor backtracks ≈ +29 - 1.25.
- Obstacle penalties: 0 (no hits in good runs).
- Exploration bonus: +0.3 x ~15-17 new cells ≈ +4.8.

VII. CONCLUSION

This project focused on developing an autonomous navigation system for robots using Deep Reinforcement Learning. The main objective was to enable a robot to move from a starting location to a target destination within a simulated hospital environment while safely avoiding obstacles such as beds, medical equipment, and other fixtures.

To simulate this scenario, a 10 × 10 grid environment was created to represent a simplified hospital floor layout. Various obstacles were placed at different grid positions to mimic real-world constraints. During training, the robot explored the environment by performing different actions and gradually learned optimal navigation strategies through rewards for successful movements and penalties for collisions or inefficient behaviour. At the core of the system was a Deep Q-Network (DQN), a neural network that estimates the expected value of each possible action based on the current state of the environment. To improve the learning performance and stability of the model, two advanced extensions were incorporated: Double Deep Q-Network (Double DQN) and the Dueling Network Architecture. Double DQN helps reduce the problem of overestimated Q-values that can occur in standard DQN models, while the Dueling architecture separates the estimation of state values and action advantages, enabling more efficient learning in complex environments.

Experimental results showed that deep reinforcement learning was highly effective for autonomous navigation. The trained agent successfully learned efficient paths that avoided obstacles and minimized unnecessary movements. The integration of Double DQN reduced overoptimistic value estimates, while the Dueling architecture improved learning efficiency when multiple actions produced similar outcomes. As a result, the training process became more stable, and the final model achieved a 100 % success rate during testing within the simulated environment.

REFERENCES

[1] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. Nature, 518(7540), 529-533. The groundbreaking Nature paper that introduced DQN—showed neural nets could master Atari games from raw pixels alone, beating human pros.

[2] van Hasselt, H., Guez, A., & Silver, D. (2016). Deep reinforcement learning with double Q-learning. In Proceedings of the AAAI Conference on Artificial Intelligence (Vol. 30, No. 1). Fixed DQN's bad habit of overestimating action values by splitting selection (online net) from evaluation (target net)—much more stable training.

[3] Wang, Z., Schaul, T., Hessel, M., Hasselt, H., Lanctot, M., & de Freitas, N. (2016). Dueling network architectures for deep reinforcement learning. In International Conference on Machine Learning (pp. 1995-2003). PMLR. Smart architecture splitting Q-values into state value V(s) and action advantages A(s,a)—learns way faster when actions don't differ much.

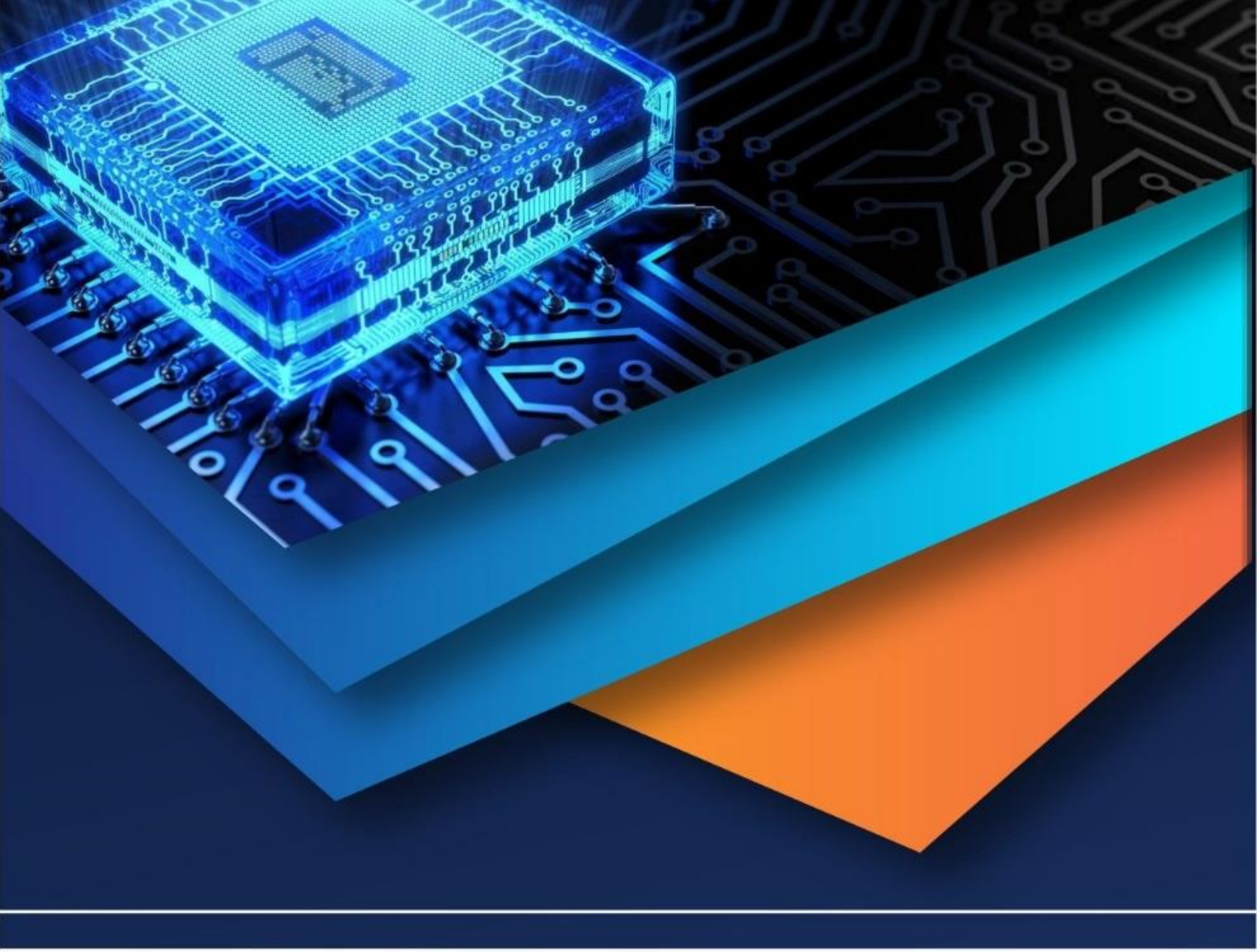
[4] Schaul, T., Quan, J., Antonoglou, I., & Silver, D. (2016). Prioritized experience replay. In International Conference on Learning Representations (ICLR). Instead of random replay, prioritize surprising experiences—makes learning from important moments instead of boring repeats.

[5] Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction (2nd ed.). MIT Press. The definitive RL textbook—covers MDPs, Q-learning, policy gradients, everything from basics to advanced.

[6] Watkins, C. J. C. H., & Dayan, P. (1992). Q-learning. Machine Learning, 8(3-4), 279-292. The original paper that invented Q-learning—made off-policy learning practical for the first time.

[7] Ng, A. Y., Harada, D., & Russell, S. (1999). Policy invariance under reward transformations: Theory and application to reward shaping. In Proceedings of the 16th International Conference on Machine Learning (ICML), 278-287. Proves you can add shaping rewards (like distance bonuses) without messing up the optimal policy—key for navigation tasks.

[8] Mirowski, P., Pascanu, R., Viola, F., Soyer, H., Ballard, A., Banino, A., ... & Hadsell, R. (2016). Learning to navigate in complex environments. arXiv preprint arXiv:1611.03673. DeepMind's work getting RL agents to navigate realistic 3D maze environments from visual input.



10.22214/IJRASET

45.98

IMPACT FACTOR: 7.129

IMPACT FACTOR: 7.429

INTERNATIONAL JOURNAL FOR RESEARCH

IN APPLIED SCIENCE & ENGINEERING TECHNOLOGY