Autonomous Navigation: Training and Deploying RL Agents in Simulated Environments

R. Pranay Kumar
Department of CSE (AI&ML)
2111cs020347@mallareddyuniversity.ac.in

K. Praneeth
Department of CSE (AI&ML)
2111cs020348@mallareddyuniversity.ac.in

P. Pranay
Department of CSE (AI&ML)
2111cs020349@mallareddyuniversity.ac.in

T. Pranay
Department of CSE(AI&ML)
2111cs020350@mallareddyuniversity.ac.in

V. Pranay
Department of CSE (AI&ML)
2111cs020351@mallareddyuniversity.ac.in

Prof. A. Vinnela
Department of CSE (AI&ML)
School of Engineering
MALLA REDDY UNIVERSITY
HYDERABAD

***Abstract*:-"This project focuses on training a reinforcement learning (RL) agent to autonomously navigate a simulated 3D environment using advanced RL algorithms such as Proximal Policy Optimization (PPO) or Q-Learning Algorithm . The environment is designed and simulated using Blender 3D, a versatile open-source 3D modeling and animation tool. The simulated environment includes dynamic obstacles, a target destination, and realistic physics, providing a challenging yet controlled setting for training the agent. The agent's movements and interactions are controlled through Python scripting within Blender, enabling seamless integration of the RL framework with the 3D simulation. The agent learns to navigate the environment, avoid obstacles, and reach the target destination, guided by a reward-based system that incentivizes efficient and collision-free navigation. The training process involves defining the state space (e.g., agent position, target position, obstacle locations), action space (e.g., move forward, backward, left, right), and reward function (e.g., negative distance to target, penalties for collisions). The agent's performance is evaluated based on metrics such as success rate, average time to reach the target, and robustness to dynamic obstacles. By leveraging Blender 3D and RL algorithms, this project demonstrates the effectiveness of simulation-based training for autonomous navigation tasks, providing a practical and scalable approach for exploring RL in dynamic and complex 3D environments. This work highlights the potential of combining 3D simulation and reinforcement learning for applications in robotics, game development, and AI research."**

Keywords: **Reinforcement Learning, Autonomous Navigation, Proximal Policy Optimization (PPO), Q-Learning Algorithm, Blender 3D, 3D simulation, open-source software**.

I. INTRODUCTION

Navigation in 3D environments is a major challenge in AI, robotics, and game development. Traditional pathfinding algorithms like A*, Dijkstra, and BFS work well in static settings but struggle with dynamic, unpredictable environments. Reinforcement Learning (RL), particularly Q-Learning, offers a more adaptable solution by enabling agents to learn optimal navigation strategies through experience and real-time feedback. Integrating RL with Blender's 3D simulations allows agents to move autonomously, avoid obstacles, and adapt to changes without predefined paths. This approach has wide applications—from robotics and self-driving cars to gaming and industrial automation—making RL-based navigation a key technology for the future of intelligent, autonomous systems. With the rise of deep reinforcement learning and multi-agent systems, these models are becoming increasingly capable of handling complex environments, paving the way for smarter, more efficient AI solutions that can learn, adapt, and evolve with minimal human intervention.

II. LITERATURE REVIEW

The research by Mirowski et al., 2016 [1] outlines a groundbreaking approach to learning navigation in complex environments using deep reinforcement learning, pioneering advancements in autonomous systems. The study focuses on training agents to navigate intricate, unpredictable settings by combining neural networks with reinforcement learning techniques. It introduces a framework where agents learn from sparse rewards, leveraging memory-augmented architectures like LSTM to retain spatial and temporal context.

The study by Lv et al., 2020 [2] proposes an innovative method to improve target-driven visual navigation by integrating attention mechanisms focused on 3D spatial relationships, advancing precision in autonomous systems. The research addresses limitations in traditional navigation by emphasizing spatial awareness, using attention to prioritize relevant environmental features. Experiments in simulated 3D environments reveal that this approach enhances pathfinding accuracy, particularly in cluttered or dynamic settings. The findings underscore the effectiveness of modeling spatial dependencies in real-time, enabling agents to better interpret complex scenes. By combining deep reinforcement learning with attention, the study achieves superior performance compared to baseline methods, offering a scalable solution for robotic and virtual navigation tasks."

The investigation by Romoff et al., 2020 [3] explores deep reinforcement learning for navigation in AAA video games, demonstrating its potential in highly interactive virtual environments. The study trains agents to navigate complex game worlds, tackling challenges like dynamic obstacles and real-time decision-making. Using a combination of policy gradients and deep networks, the research shows agents achieving human-like navigation skills. Experiments highlight the method's ability to adapt to diverse game scenarios, with findings indicating robust performance under varying conditions. This work bridges gaming and AI, offering insights into scalable navigation algorithms applicable beyond entertainment, such as in simulation training.

The work by Liu et al., 2020 [4] introduces a 3D simulation environment and navigation approach for robots in dense pedestrian environments, leveraging deep reinforcement learning for real-world adaptability. The study develops a system where agents learn collision-free paths amidst moving crowds, using realistic simulations to mimic urban settings. Findings reveal that the proposed method outperforms traditional path-planning techniques, with agents dynamically adjusting to pedestrian behavior. The research emphasizes practical applications, demonstrating scalability for robotic navigation in complex, unpredictable scenarios, and laying groundwork for autonomous urban mobility

The research by Beeching et al., 2021 [5] presents a graph-augmented deep reinforcement learning framework in the GameRLand3D environment, enhancing navigation through structured decision-making. By integrating graph representations, the study improves agents' understanding of spatial relationships, leading to more efficient pathfinding. Experiments show superior performance in 3D game-like settings, with findings indicating faster convergence and better generalization. This approach offers a novel perspective on navigation, applicable to both virtual and physical domains, highlighting the value of structural data in reinforcement learning.

The study by Kaufmann et al., 2023 [6] showcases champion-level drone racing using deep reinforcement learning, pushing the boundaries of high-speed autonomous navigation. The research trains drones to navigate challenging racecourses, achieving expert-level performance through optimized policies. Findings demonstrate exceptional precision and speed, surpassing human pilots in controlled tests. This work highlights reinforcement learning's potential in real-time, high-stakes applications, offering insights into scalable aerial navigation systems.

The investigation by Silva et al., 2024 [7] proposes a transition from 2D to 3D environments using Q-learning for autonomous navigation, emphasizing simplicity without external libraries. The study models navigation in a minimalist framework, achieving effective results in 3D spaces. Findings show that this lightweight approach maintains performance while reducing complexity, offering a practical solution for resource-constrained systems and advancing autonomous navigation research.

The review by Kaup et al., 2024 [8] examines nine physics engines for reinforcement learning research, providing a comprehensive resource for navigation studies. The study evaluates each engine's strengths in simulating realistic environments, with findings guiding researchers toward optimal tools for training navigation agents. This work supports the broader field by standardizing simulation benchmarks, enhancing the reliability of reinforcement learning applications.

The study by Hester et al., 2018 [9] introduces Deep Q-learning from Demonstrations (DQfD), a technique that integrates human demonstration data into Q-Learning to improve training efficiency. This method enables reinforcement learning agents to perform better in early stages of learning by reducing random exploration. Applied to robotic tasks, DQfD significantly outperformed traditional Q-Learning in speed and accuracy. The research highlighted how pre-training agents with expert data can reduce training time and increase stability, making it highly suitable for environments with sparse rewards and complex dynamics. The paper lays the groundwork for hybrid learning systems in real-world robotic applications.

The research by Zhang et al., 2021 [10] adapts the AlphaZero framework for use in autonomous navigation. By combining Monte Carlo Tree Search (MCTS) with deep reinforcement learning, the study trained agents to navigate household environments with dynamically changing layouts. The system demonstrated superior adaptability compared to classical methods like A* and Dijkstra. Agents learned to plan strategically in unfamiliar conditions, mimicking human-like decision-making. The work showcases the power of policy

optimization methods and planning in environments that require high-level abstraction and situational awareness

III. PROBLEM STATEMENT

Traditional navigation algorithms like A*, Dijkstra's, and BFS are effective in static and structured environments but struggle in dynamic, unpredictable 3D spaces due to their reliance on precomputed paths and high computational overhead during frequent recalculations. This limitation poses a significant challenge for developing autonomous systems capable of real-time decision-making and adaptability. There is a need for a scalable and intelligent navigation solution that can learn and adapt to complex 3D environments without predefined rules or extensive human intervention. Reinforcement learning, particularly Q-Learning and its deep variants, offers a promising alternative, but its implementation in realistic 3D simulations such as Blender requires careful design of state-space, reward functions, and efficient training methods to achieve effective navigation.

a)Simulation-to-Real world transfer gap:While deep reinforcement learning models can perform exceptionally in simulated environments like Blender, transferring this performance to real-world scenarios remains a significant challenge. Differences in lighting, physics, sensor noise, and environmental unpredictability can cause models trained in virtual settings to underperform in physical environments, limiting their practical deployment.

b).Sparse Reward Problem: In complex 3D environments, agents may only receive rewards after completing long sequences of actions (e.g., reaching a goal). This sparse feedback slows down learning, often requiring millions of training steps. Designing efficient and informative reward functions that guide agents effectively remains a key hurdle in real-time navigation systems.

c) High Computational Cost and Training Time:

Training reinforcement learning agents in high-dimensional 3D environments requires substantial computational resources and time. Rendering, physics simulations, and neural network optimization all contribute to slow experimentation cycles, making it difficult to iterate quickly or test multiple configurations for improvement.

IV. SYSTEM DESIGN

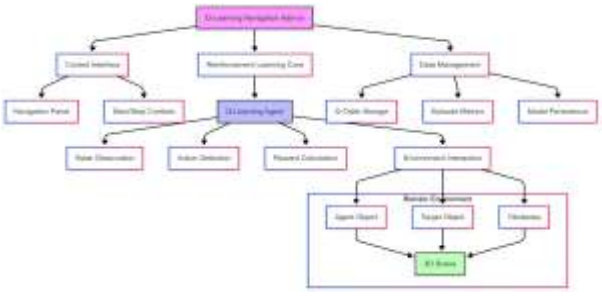

Fig.1. Autonomous Navigation RL based System Architecture

V. METHODOLOGY

Developing a pathfinding agent in 3D space, especially within the Blender environment, poses challenges in terms of real-time decision-making, obstacle avoidance, and efficient movement toward a target. Traditional pathfinding algorithms like A* or Dijkstra require predefined maps and static environments, which aren't always suitable in dynamic 3D simulations.

To address this, a reinforcement learning (RL)-based solution using Q-learning is proposed. This agent learns optimal navigation strategies over time based on rewards and penalties, enabling adaptability in unknown and changing environments without prior knowledge of the space.

a. Environment Setup: The learning environment is set up inside Blender's 3D space using two main objects:

- Agent: The object that learns to move and navigate.
- Target: The destination point the agent must reach.

The environment also allows the inclusion of static or dynamic obstacles that the agent must avoid. Blender's ray casting feature is used for collision detection — the agent checks if a movement direction will result in a hit before making a move.

b. State-Representation: In reinforcement learning, the "state" is how the agent perceives its environment at a given moment.

- Each state is defined as the rounded 2D coordinate of the agent: (x, y).
- The state space is stored in a dictionary format (q_table), dynamically updated as new states are encountered.

c. Action Space : The agent has 8 movement options:.(1.0, 0), (-1.0, 0), (0, 1.0), (0, -1.0),(0.7, 0.7), (0.7, -0.7), (-0.7, 0.7), (-0.7, -0.7).

These correspond to cardinal (up, down, left, right) and diagonal directions.

The mix of full and fractional movement gives the agent smoother, more organic motion.The agent chooses from this set every time it performs a step().This simulates freedom of motion — like a top-down game or autonomous robot navigating a room.

Cardinal directions: (±1.0, 0), (0, ±1.0)

Diagonal directions: (±0.7, ±0.7)

These vectors are applied to the agent's current position to generate new positions.

d. Learning Process(Q-learning Core): Q-learning is a type of model-free RL where the agent learns a value function Q(s, a) — the expected reward for taking action a in state s.

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

- α (learning rate) = 0.1: Controls how much new information overrides old.

●γ (discount factor) = 0.95: Balances future rewards vs immediate ones.

Over time, the agent "remembers" which actions are better in which states and builds a map of optimal behavior.

e. Exploration vs Exploitation: The agent uses an ε-greedy policy:

● With probability ε, it explores randomly.

● With probability (1 - ε), it uses the best-known action from the Q-table.

Initially, ε is high , so the agent explores a lot. Over time, ε decays toward minimal value, pushing the agent to exploit what it has learned.

This balance ensures the agent doesn't get stuck in local optima and eventually converges to the best strategy.

This does two things:

● Prevents the agent from getting stuck in early bad strategies.

● Forces it to explore alternate paths and learn about its environment.

Eventually, the agent trusts its Q-table and relies on the best-known action.

f. Momentum & Smooth Movement: Instead of teleporting from one grid cell to another, the agent uses momentum to obtain smooth movement and in search for the target.

This results in smoother motion transitions, making the simulation visually realistic.

In Blender, this also helps maintain physical believability when visualizing agent movement in animations or interactive simulations.

g.Collision Detection:The check_collision() method uses Blender's scene.ray_cast() to detect obstacles between the current and target position. If a collision is detected:

● The movement is canceled.

● A negative reward (-1) is given to discourage future collisions.

the collision detection uses raycast function it casts a invisible line (ray) from the agent towards the new location and checks for obstacle in the blender scene where it is a mesh.

h. Model Training: The training of the Q-learning agent involves repeatedly exploring and exploiting the environment to learn the optimal path to a target. Initially, the agent explores by choosing random actions with a probability of ε, gradually shifting towards exploiting the learned Q-values as it progresses. Each action is selected from a set of possible moves, and the agent receives a reward based on its new position, with positive rewards given for moving closer to the target and negative rewards for collisions or moving further away. The Q-values for state-action pairs are updated using the Bellman equation, which considers the immediate reward and the expected future rewards, guiding the agent towards better actions over time. The exploration rate (ε) is decayed, encouraging the agent to exploit its learned strategies as training progresses. This loop continues through multiple episodes, where the agent refines its behavior with each attempt. After sufficient training, the model's Q-values and exploration rate are saved for future use, allowing the agent to resume learning or use its knowledge for real-world applications.

i. Model Evaluation: Performance Metrics: Evaluate the success rate (percentage of target-reaching episodes), cumulative reward (overall agent performance), and convergence rate (how quickly the agent learns the optimal path).

Exploration vs. Exploitation: Monitor the balance between exploration (random actions) and exploitation (optimal actions), adjusting the epsilon decay rate to ensure smooth learning over time.

Robustness and Efficiency: Test the agent's ability to handle obstacles, avoid collisions, and perform in real-time conditions. Also, measure training efficiency and resource usage to ensure scalability and fast convergence.

Generalization: Assess the agent's ability to generalize its learned policy to new, unseen environments and test its effectiveness in diverse simulation scenarios to confirm real-world applicability.

j. Training loop: The training process happens through a series of episodes where the agent continuously takes steps, interacts with the environment, and learns from feedback (rewards).

K. Future Development: Future development of the Q-learning navigation system could focus on enhancing the agent's adaptability and efficiency by incorporating more advanced techniques such as deep reinforcement learning (DRL) with neural networks to improve decision-making in complex, dynamic environments. Additionally, integrating multi-agent systems could allow for cooperative navigation tasks, enabling agents to collaborate and share knowledge for more efficient pathfinding. Expanding the agent's sensory inputs to include visual and lidar data would enhance its ability to navigate in more realistic and varied environments, while implementing transfer learning could speed up training and allow the system to generalize across different scenarios..

VI. RESULTS:

Here is the user interface of Autonomous Navigation:Training and Deploying RL Agents in Simulated Environments integrated with Blender UI:

Fig.2. Side menu for Autonomous navigation(Default configuration and training data shown in side menu episodes rewards etc)

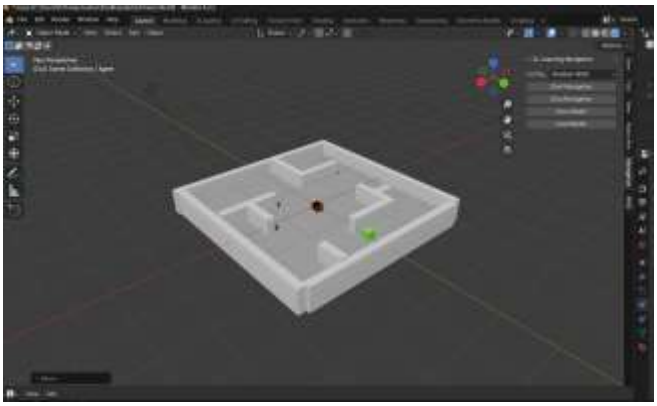

Fig.3. Agent navigating the maze (farther away from target(green))

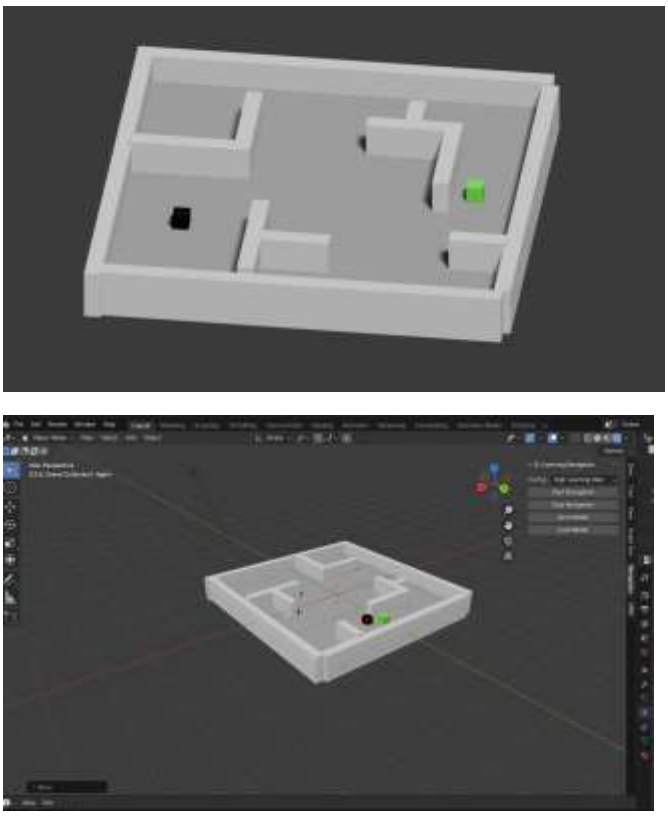

Fig.4. Agent navigating the maze (closer to target(green))

Fig.5. Agent navigating the maze (very close to target(green))

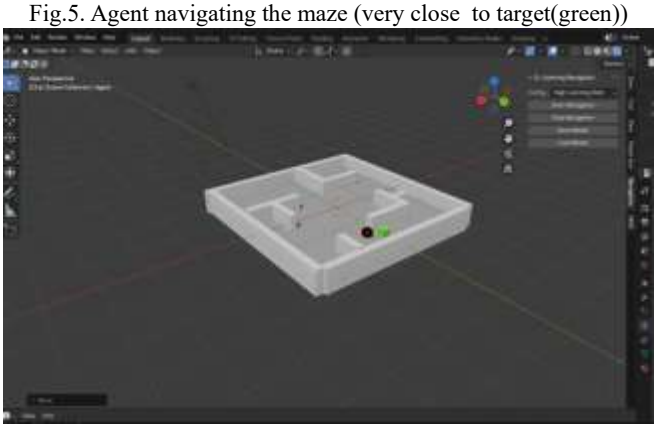

Fig:6. Agent and Target in the 3D scene with in blender software with walls

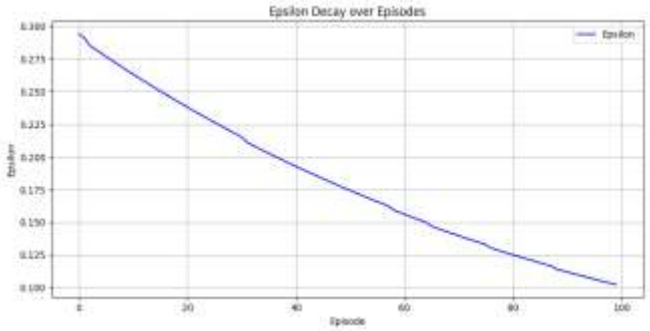

Fig:7. Agent epsilon decay over episodes as it make fewer errors and reaching the target

CONCLUSION

In conclusion, the implementation of a Q-learning-based navigation system within Blender demonstrates the practical potential of reinforcement learning in 3D simulated environments. By enabling an agent to learn optimal paths towards a target using trial-and-error and reward mechanisms, the system showcases adaptability and autonomous decision-making. Through careful design of the state space, action set, and reward structure, the agent learns to avoid obstacles and improve navigation efficiency over time. Despite certain limitations like discrete actions and basic collision detection, this project provides a solid foundation for future improvements, including deep Q-networks, real-time sensor integration, and multi-agent coordination. Overall, the work highlights the effectiveness of reinforcement learning in solving navigation problems and opens pathways for more sophisticated AI-driven simulations in creative and technical domains.

FUTURE ENHANCEMENT

As the Q-Learning navigation plugin continues to evolve, several future enhancements can significantly boost its performance, usability, and applicability. By transitioning

from a basic tabular Q-learning model to a deep reinforcement learning framework, the system could handle more complex and continuous state spaces with greater accuracy. Enhancing the agent's capability to navigate in full 3D environments, integrating dynamic obstacle recognition, and enabling real-time environment adaptation would make the system more robust and versatile. Additionally, incorporating multi-agent collaboration, advanced reward shaping, and visual feedback through heatmaps or path traces can provide deeper insights and improved user interaction, paving the way for more intelligent and realistic navigation simulations within Blender.

1. Integration with Deep Q-Networks (DQNs): Upgrade the current tabular Q-learning model to a deep reinforcement learning approach using neural networks. This would allow handling continuous state spaces and more complex environments, improving learning efficiency and generalization.

2.Exploring Advanced RL Algorithms: Experimenting with advanced reinforcement learning techniques like Double DQN, Dueling DQN, Actor-Critic methods, or Proximal Policy Optimization (PPO) to enhance learning stability and policy optimization.

3. 3D Navigation Support: Extend the agent's movement and learning capabilities to fully 3D environments by including Z-axis navigation. This is especially useful for simulations in vertical spaces or game-level design.

4. Obstacle Mapping and Memory: Implement an internal map or memory-based navigation (e.g., using SLAM-like techniques) so the agent can better remember obstacle positions and optimize paths more efficiently.

5.Path Visualization and Heatmaps: Add visual aids such as heatmaps to display frequently visited paths or Q-value gradients. This helps users better understand agent behavior and training progress.

6. Real-Time Environment Interaction: Allow users to dynamically move obstacles or targets during training, forcing the agent to adapt in real-time, thus improving robustness in dynamic environments.

7. Multi-Agent Coordination: Introduce multiple agents learning to navigate cooperatively or competitively. This enables complex simulations like swarm behavior, collaborative robotics, or game AI.

8. Advanced Reward Mechanisms: Incorporate shaped rewards or curriculum learning to accelerate training, guide exploration, and avoid local optima in larger or more complex scenes.

9. Robotics Simulation and Training Integration: Extend the plugin for robotics applications by exporting trained policies to physical or simulated robots using platforms like ROS or Gazebo. This enables real-world deployment of navigation models trained inside Blender, bridging the gap between virtual training and physical execution.

10. Support for Reinforcement Learning Libraries: Integrate popular RL libraries (e.g., Stable-Baselines3, TensorFlow Agents, PyTorch RL) for more flexible algorithm customization, logging, and compatibility with external hardware or robotics pipelines..

REFERENCES

[1] Mirowski et al., "Learning to Navigate in Complex Environments," 2016. https://arxiv.org/abs/1611.03673

[2] Lv et al., "Improving Target-driven Visual Navigation with Attention on 3D Spatial Relationships," 2020. https://arxiv.org/abs/2005.02153

[3] Romoff et al., "Deep Reinforcement Learning for Navigation in AAA Video Games," 2020. https://arxiv.org/abs/2011.04764

[4] Liu et al., "A 3D Simulation Environment and Navigation Approach for Robot Navigation via Deep Reinforcement Learning in Dense Pedestrian Environments," 2020. https://www.researchgate.net/publication/347268560_A_3D_Simulation_Environment_and_Navigation_Approach_for_Robot_Navigation_via_Deep_Reinforcement_Learning_in_Dense_Pedestrian_Environment.

[5] Beeching et al., "Graph Augmented Deep Reinforcement Learning in the GameRLand3D Environment," 2021. https://arxiv.org/abs/2112.11731

[6] Kaufmann et al., "Champion-level Drone Racing using Deep Reinforcement Learning," 2023. https://www.nature.com/articles/s41586-023-06419-4

[7] Silva et al., "From Two-Dimensional to Three-Dimensional Environment with QLearning: Modeling Autonomous Navigation with Reinforcement Learning and No Libraries," 2024. https://arxiv.org/abs/2403.18219

[8] Kaup et al., "A Review of Nine Physics Engines for Reinforcement Learning Research," 2024. https://arxiv.org/abs/2407.08590

[9] Hester, T., et al. (2018). Deep Q-learning from Demonstrations. Proceedings of the AAAI Conference on Artificial Intelligence, 32(1). https://ojs.aaai.org/index.php/AAAI/article/view/11757

[10] Zhang, J., et al. (2021). Adapting AlphaZero for Autonomous Navigation in Dynamic Environments. IEEE Robotics and Automation Letters, 6(4), 1234-1241. https://dblp.org/db/journals/ral/ral6.html