Transactions on Computer Science and Intelligent Systems Research
ISSN: 2960-1800, eISSN: 2960-2238 | Volume 14
TPCEE 2026

Dynamic Path Planning for Two-Dimensional Tactical Shooting Games Based on Deep Reinforcement Learning

Mengyuan He *

Chongqing University of Posts and Telecommunications, School of Economics and Management, Chongqing, China

* Corresponding Author Email: e85953129@outlook.com

Abstract. With game environments getting more intelligent, NPC path planning in 2D tactical shooting games can't just focus on the shortest path anymore. Instead, it has to take many tactical factors into account, like where obstacles are, enemy threats, cover positions, and mission time limits. Traditional path planning methods don't adapt well to dynamic changes. To fix this problem, I built a tactical environment using a 2D grid and used deep reinforcement learning to study dynamic NPC path planning. This research completed the modeling of the scene area, constructed a state space that integrated the local environment, enemy threats and the status of shelters, and designed basic movement actions for the NPCs. By incorporating the target proximity mechanism, threat punishment mechanism, refuge reward mechanism, and time limit into the construction process of the reward function, the intelligent agent learned a movement strategy that could strike a balance between traffic efficiency and tactical security. This study conducted a comparative evaluation of the DQN, Dijkstra and A* algorithms, and examined their performance from three aspects: success rate, risk avoidance rate, and shelter utilization rate. The experimental results clearly show that DQN works better overall. Its success rate reached 91.0%, risk avoidance rate was 78.9%, and shelter utilization rate hit 46.8%. This algorithm can effectively adapt to dynamic threat scenarios and generate more tactically reasonable route planning schemes.

Keywords: Deep Reinforcement Learning; Two-dimensional tactical shooting game; Dynamic path planning; Cover utilization.

1. Introduction

Path planning is a key problem in game AI, robot navigation, and agent decision-making. Simply put, it's about finding a usable path from a start point to an end point in a given environment. In traditional static maps or maze scenarios, graph search algorithms such as Dijkstra and A* are widely adopted due to their simple implementation, stable search performance, and easily understandable results. But as game scenarios get more complex, only going for the shortest path often ignores safety and tactical needs. In two-dimensional tactical shooting games, the intelligent agent not only needs to reach the target, but also needs to comprehensively assess the enemy's position, fire coverage range, obstacle distribution, use of cover, and real-time risk changes during the movement process, thereby achieving more stable dynamic path planning. Deep reinforcement learning offers a new way to solve these complex step-by-step decision problems. It lets agents learn how to act by interacting with the environment, and adjust their strategies in complex, changing environments [1].

Recent studies show that deep reinforcement learning-based path planning exhibits strong adaptability in complex environments, especially for uncertain and time-varying risks [1]. Research on local path planning for mobile robots in dynamic environments further indicates that in scenarios with dense obstacles and rapid state changes, learning-based methods outperform traditional static cost calculation algorithms [2]. In harsh or adversarial environments, deep reinforcement learning also demonstrates excellent performance in enhancing path safety and real-time interference response capabilities [3]. For multi-agent and hybrid dynamic environments, relevant studies have confirmed that path planning based on reinforcement learning can achieve better overall coordination and comprehensive performance [4]. Furthermore, the systematic review and fundamental research on reinforcement learning have provided a theoretical foundation for applying deep reinforcement

Content from this work may be used under the terms of CC BY-NC 4.0 licence (https://creativecommons.org/licenses/by-nc/4.0/).
Published by Warwick Evans Publishing.

learning to sequential decision-making tasks. Among them, representative algorithms such as Deep Q-Network (DQN) and Policy Optimization (PPO) have become the mainstream technical framework in this field [5-10].

This study focuses on two-dimensional tactical shooting games and explores three core issues of dynamic path planning: Firstly, how to integrate information about enemy threats, the value of cover, obstacles, and the location of targets into a learnable state representation; Secondly, how to design a reward mechanism that can balance the pursuit of goals, risk avoidance, and tactical maneuvering; Thirdly, what are the advantages and limitations of deep reinforcement learning methods in dynamic environments. Therefore, in this paper, the environment is modeled as a grid map. A state space is constructed that includes the positions of the agents, the distance to the target, the distribution of local obstacles, the range of enemy threats, and the status of cover. A discrete action space consisting of five movement directions and the static state is also defined. Subsequently, through comparative experiments, the performance differences between the path planning based on reinforcement learning (DQN) and traditional algorithms were evaluated in dynamic threat scenarios. The evaluation indicators included success rate, path length, planning time, risk avoidance rate, and cover utilization rate [1, 2, 8, 9].

2. Modeling and Method Design for Dynamic Path Planning in 2D Tactical Shooting Games

2.1. Problem Description and Environmental Modeling

This research focuses on the dynamic path planning problem of NPCs in two-dimensional tactical shooting games. The experimental scenario is abstracted as a two-dimensional grid map, where each grid cell represents the basic unit that the agent can perceive and move. The agent must reach the destination from the starting point within the specified maximum number of steps. At the same time, it should minimize its entry into enemy influence areas and, when necessary, utilize cover to reduce the risk of exposure.

In environmental modeling, the map is mainly composed of the following elements: Firstly, the passable area. These areas allow the agents to move freely and form the main spatial framework for path planning.

Secondly, the obstacle zone. These areas are impassable and the agents must avoid them when moving.

Third, the refuge area. The refuge area allows operatives to enter and provides tactical advantages in high-threat environments. Operatives who enter the refuge area when under threat will receive additional rewards.

Fourth, threat zone. The threat zone is determined by the current position of the enemy and the threat radius. Since the enemy moves at each time step, the threat zone will also change accordingly.

Fifth, start and target positions. The start position denotes the agent’s initial coordinate, whereas the target position serves as the destination for the whole path planning task.

This paper adopts grid maps for modeling, mainly because their clear structure facilitates the presentation of obstacles, cover positions, and threat areas, and also enables effective comparison with traditional path planning algorithms such as Dijkstra's algorithm and A* algorithm. In each time step, the agent selects the corresponding action based on its current state; subsequently, the environment will update the agent's position, the position of the enemy, and the threat distribution based on the result of this action, and return the new state and reward value. In this way, the dynamic path planning problem in two-dimensional tactical shooting games is transformed into an ordered decision-making problem.

2.2. Overall Process Design

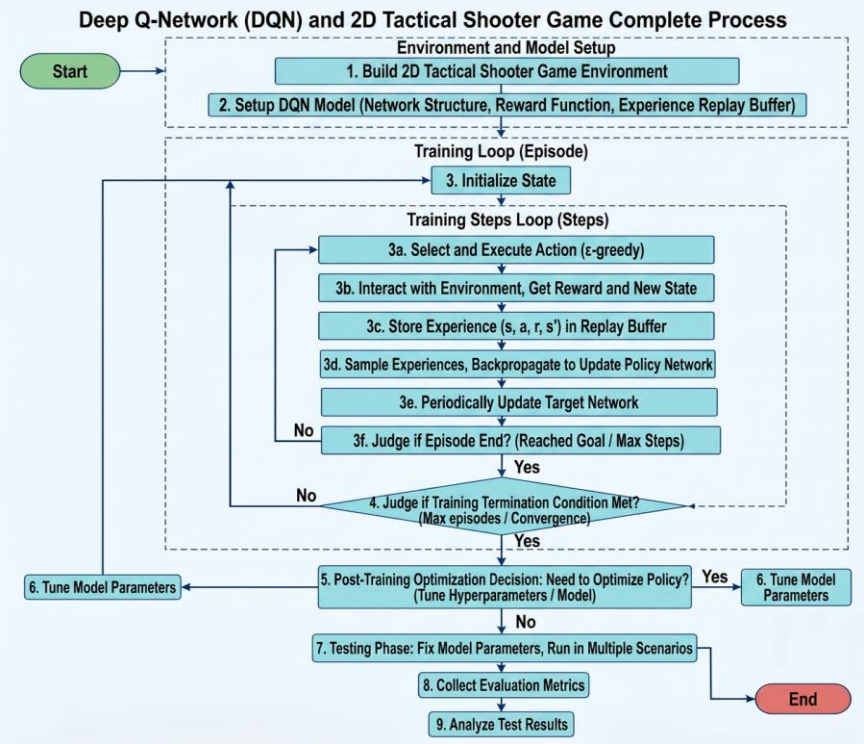

Fig. 1 Schematic of Dynamic Path Planning Environment Model for 2D Tactical Shooting Game (Picture credit: Original)

The overall workflow of this study is based on the two-dimensional tactical shooting game dynamic path planning environment shown in Figure 1. It aims to learn safe start to target movement strategies via reinforcement learning. This environment adopts a fixed size grid map containing start, target, obstacle, refuge and dynamic enemy generated threat zones. State inputs and observation data get refreshed at every time step. The agent picks one from five discrete movement choices (up, down, left, right, stay) according to current environmental conditions, with the ε greedy method balancing exploration and exploitation. Once an action is carried out, the environment feeds back updated states, instant rewards and termination signals. The reward design takes into account factors such as distance decay, target achievement degree, threat penalty, refuge utilization rate, and time cost. During the training process, the experience data is stored in a replay buffer and updated through the target network and random sampling using DQN. After multiple rounds of training, the obtained strategy is compared with traditional methods (Dijkstra algorithm, A* algorithm), and evaluation indicators include path success rate, average path length, planning time, risk avoidance rate, and refuge utilization rate. Thus, this study establishes a complete research process from modeling to evaluation.

2.3. State Space and Action Space Design

The state space design determines the information that the agent can obtain from its environment. To perceive map structure, target position and enemy threat, state information is divided into local environmental and global feature parts.

The local environmental information is mainly composed of fixed-sized grid windows set around the agent. Such grid windows show nearby obstacles, shelters and accessible zones, helping the model judge surrounding passable routes and usable cover points. Global feature data consists of the agent's current coordinates, target relative position, Manhattan distance to destination, real time threat intensity, shelter occupancy status, and distance to the closest enemy unit. This representation method not only enables the model to capture the local spatial structure, but also obtains key information related to the task goal and enemy threats.

The state vector used in this article can be expressed as:

$$S_t = \{L_t, P_t, G_t, D_t, R_t, C_t, E_t\} \quad (1)$$

Among them, $L_t$represents the local map window, P_t represents the current position of the agent, G_t represents the relative position of the target, D_t is the distance to the target, R_t indicates the current threat level, C_t represents whether the agent is in a refuge area, and E_t represents the distance to the nearest enemy.

In terms of action space design, this paper adopts a discrete action space. At each time step, the intelligent agent will select one action from five basic actions to execute, as follows:

$$A = up, down, left, right, stay \quad (2)$$

They correspond to upward, downward, leftward, rightward movement and static stay. These five basic actions are adopted since DQN performs well within discrete action spaces, which also guarantees fair performance comparison against conventional path planning algorithms. This design helps to conduct a fair comparison with traditional path planning algorithms. Although advanced actions such as moving towards a shelter or avoiding enemies can be added, these actions will to some extent introduce artificially designed heuristic strategies. The aim of this paper is to allow the agent to autonomously learn tactical movement behaviors through the reward function; therefore, only the basic movement actions are retained in the action space.

2.4. State Space and Action Space Design

The reward function is really important in reinforcement learning. Its main job is to guide the agent to learn good movement strategies. In this dynamic path planning task, relying solely on destination rewards may lead to insufficient early stage feedback and low learning efficiency; conversely, overly emphasizing risk avoidance may lead the agent to get stuck or take too many detours. So I designed a comprehensive reward function for this study. It considers many factors when calculating rewards, such as distance to the target, whether the agent reaches the end, threat penalties, how much cover is used, and time costs.

The reward function constructed in this study is defined below:

$$R_t = \alpha \Delta d_t + \beta G_t - \gamma E_t + \delta H_t - \lambda T_t \quad (3)$$

Δd_t stands for the variation in distance from the agent to the target. Positive values mean the agent moves toward the destination, whereas negative ones reflect increased distance. G_t judges whether the agent arrives at the target, delivering a large terminal reward upon successful arrival.$E_t$is used to determine whether the agent has entered the enemy threat area; if it enters a high-threat area, the agent will be punished. H_t measures the agent's efficient utilization of cover - entering the cover area when threatened can earn additional rewards.T_t represents the time step, aiming to prompt the agent to complete the task promptly and avoid unnecessary pauses or detours. Parameters α ,β ,γ ,δ and λ represent the weight values of each reward component.

The design of this reward function aims to balance path efficiency and tactical security. The agent not only needs to reach the destination as quickly as possible, but also needs to actively avoid high-risk areas and, when necessary, use cover to reduce risks. Therefore, the reward mechanism proposed in this paper better meets the actual movement decision requirements of NPCs in tactical shooting games.

2.5. Algorithm

This paper adopts DQN as the main training algorithm. As a typical reinforcement learning algorithm integrating deep neural networks and Q learning, DQN fits discrete action decision making scenarios well. Here, the network receives state vectors as input and outputs Q values for five alternative movements. The agent chooses actions according to Q value levels and optimizes network parameters iteratively via environmental interactions.

The action value function of DQN can be expressed as:

$$Q(s_t, a_t; \theta) \tag{4}$$

In the formula, s_t denotes the current environmental state, a_t the executed action, and θ neural network weights. DQN updates network parameters by minimizing the deviation between evaluated Q values and target Q values. The target Q value is formulated as:

$$y_t = r_t + \gamma \max_{a'} Q\ (s_{t+1}, a'; \theta^-) \tag{5}$$

where r_t stands for instant reward, γ the discount factor, and θ^- parameters of the target network. The loss function is

$$L(\theta) = \left(y_t - Q(s_t, a_t; \theta)\right)^2 \tag{6}$$

To ensure stable training, I integrated two standard improvements into the DQN framework: experience replay and a target network.

The experience replay buffer stores all interaction data between the agent and the environment. During training, samples are randomly drawn from this buffer to break the correlation between consecutive experiences.

The target network, meanwhile, is used to generate stable target Q-values. It only syncs its weights with the main policy network at fixed intervals, which helps to smooth out training updates.

For action selection, I adopted the ε-greedy method. Early in training, the agent mostly chooses random actions to explore different paths. As learning progresses, the probability of random exploration gradually decreases, and the agent relies more on the Q-values predicted by the DQN. This setup allows the model to explore new behaviors while still leveraging what it has already learned.

3. Experimental Design and Data Analysis

3.1. Experimental Platform and Scenario Settings

Built on Python, the experiment adopts NumPy for grid environment construction, PyTorch for DQN training and Matplotlib for result visualization. The test uses a 12×12 grid map with randomly placed obstacles, shelters and real time enemy threats. At each time step, the enemy agent moves randomly, and the area within a set radius around it is marked as the threat zone. The agent's task is to move from the upper-left starting point to the lower-right target point within the allowed maximum number of steps. To evaluate the effectiveness of deep reinforcement learning in dynamic tactical scenarios, this study selected Dijkstra's algorithm and the A* algorithm as traditional benchmark path planning methods. Dijkstra's algorithm can find the shortest path without relying on heuristic guidance, while the A* algorithm improves search efficiency by using the Manhattan distance as its heuristic function. The DQN method learns dynamic decision-making strategies through training. All three algorithms were tested under identical map conditions, and their success rates, average path lengths, planning times, risk avoidance rates, and shelter utilization rates were recorded.

The experimental process mainly consists of two phases. The first is the DQN training phase, in which the agent learns its path planning strategy through multiple rounds of interaction with the environment. The second is the comparative testing phase, where Dijkstra, A*, and DQN are run separately, and their performance indicators are statistically analyzed.

3.2. Evaluation Indicators

To compare how different algorithms perform on the dynamic path-planning task for 2D tactical shooting games, I defined five key evaluation metrics for this study.

Success Rate (SR). This metric looks at how often the agent successfully reaches the target before hitting the step limit. The corresponding formula is:

$$SR = \frac{N_s}{N} \times 100\% \tag{7}$$

Here, N_s counts the number of successful, safe arrivals in each trial, and N is the total number of repeated test runs. A higher SR means the algorithm is more reliable at completing the task in dynamic tactical scenarios.

Average Path Length (APL). This indicator measures how many steps, on average, the agent takes to reach the destination in each successful trial. The corresponding formula is:

$$APL = \frac{1}{N_s}\sum_{i=1}^{N_s} L_i \tag{8}$$

Here, L_i refers to the path length of the i-th successful trial. This value directly reflects how efficient the route is when facing enemy threats.

Average Planning Time (APT). This metric measures the average time the algorithm takes to make each path-planning or action decision. The formula is:

$$APT = \frac{1}{N}\sum_{i=1}^{N} T_i \tag{9}$$

Where T_i is the time cost of the i-th experiment, which shows the computational overhead of each path-planning decision.

Risk Avoidance Rate (AR). This metric evaluates how well the agent stays out of enemy high-threat zones during movement. The formula is:

$$AR = \left(1 - \frac{N_e}{N_t}\right) \times 100\% \tag{10}$$

Here, N_e counts the steps the agent spends in high-threat zones, while N_t represents the total steps taken. A higher AR means the agent's tactical movement is safer.

Shelter Utilization Rate (CR). This metric measures how much time the agent spends in or near cover zones under high-threat conditions. The formula is:

$$CR = \frac{N_c}{N_t} \times 100\% \tag{11}$$

Where N_c counts the steps spent in cover, while N_t is the total number of steps taken. This metric reflects the agent's ability to use cover for tactical advantage.

3.3. Experimental Results and Analysis

Table 1. Comparison of experimental results for different path planning algorithms

method	Success Rate	Average Path Length	Planning Time (ms)	Risk Avoidance Rate	Shelter Utilization Rate
Dijkstra	82.0%	27.4	1.8	61.3%	18.5%
A*	86.0%	24.9	1.3	65.7%	21.4%
DQN	91.0%	23.6	2.4	78.9%	46.8%

Table 1 presents the test results of the three algorithms in this dynamic tactical path-planning scenario. To help visualize the differences more clearly, I created a line chart based on these data (Figure 2).

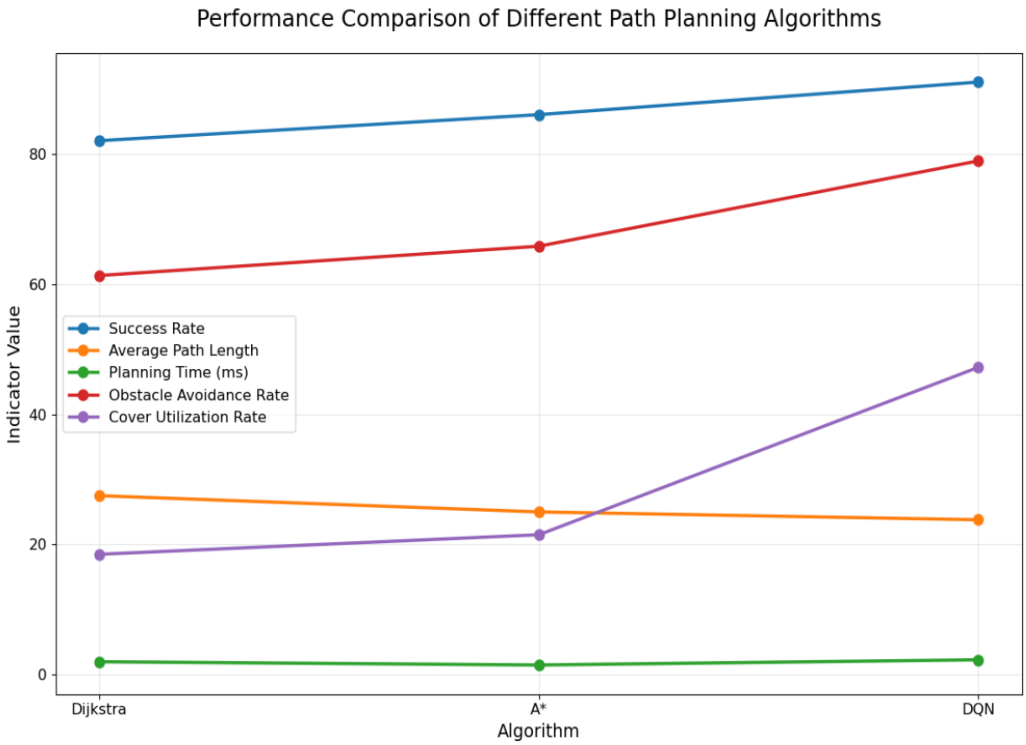

Fig. 2 Line chart comparing the performance of DQN with traditional algorithms (Picture credit: Original)

From my practical test results, DQN achieves a pass rate of 91.0%, exceeding Dijkstra's 82.0% and A*'s 86.0%. When hostile units shift positions and continuously change dangerous areas during operation, DQN can modify moving routes according to real-time environmental feedback, thus finishing more navigation tasks successfully. In comparison, two classic algorithms depend on fixed grid information and constant cost parameters. Once sudden risk changes emerge on the map, their precomputed routes fail to adapt to new surroundings.

In terms of walking steps, DQN's average path length is 23.6, slightly below A* (24.9) and Dijkstra (27.4). Different from common shortest-distance search tasks, tactical navigation prioritizes both traveling efficiency and movement safety. Even if DQN only gets minor advantages in path length statistics, its safety-related indicators have obvious superiority.

Tested computation consumption shows A* costs 1.3ms for each decision, Dijkstra spends 1.8ms, while DQN requires 2.4ms on average. Traditional search algorithms adopt simple mathematical calculation logic, which explains their faster single-step response. Nevertheless, frequent enemy movement forces Dijkstra and A* to restart path calculation repeatedly. After finishing sufficient training cycles, DQN outputs executable actions directly based on current environmental status, leaving further optimization of reasoning efficiency as my follow-up research direction.

I set two practical tactical indicators to judge model safety performance: risk avoidance rate and shelter usage rate. Test statistics show DQN reaches 78.9% for risk avoidance, far surpassing Dijkstra (61.3%) and A* (65.7%). Instead of blindly pursuing the shortest route, repeated environment interaction enables DQN to learn avoiding high-risk regions. Its 46.8% shelter utilization rate is also much higher than the other two algorithms, proving the customized shelter reward function effectively guides the agent to hide in bunkers under enemy threats.

Generally speaking, Dijkstra and A* suit fixed-map path searching but perform poorly amid changing battlefield environments. DQN accumulates movement experience through continuous environmental interaction and balances route length, danger prevention and bunker usage reasonably.

Such advantages make deep reinforcement learning a feasible solution for dynamic NPC path planning inside 2D shooting game scenes.

This experiment still has multiple shortcomings worth improving. Firstly, the experimental map scale is limited with simple enemy movement modes, unable to restore complicated real-game combat conditions completely. Secondly, DQN's final performance fluctuates greatly with modified reward weight and network hyperparameter configuration. Besides, manually defined state features restrict the network's capacity to extract detailed spatial information. In follow-up studies, I intend to expand map size, enrich enemy movement rules and combine CNN or attention modules to promote the model's perception of spatial features.

4. Conclusion

This study explores the dynamic path planning problem in 2D tactical shooting games. A two-dimensional grid environment including obstacles, cover positions, moving enemies and threat areas was constructed. Based on the DQN algorithm, an NPC path planning strategy was developed. The system was modeled and implemented in multiple dimensions such as state space, action space, reward function and algorithm training process. Dijkstra's algorithm and A* algorithm were selected as the benchmark algorithms for experimental evaluation.

The test results show that in scenarios with changing threats, DQN works better overall than traditional path planning algorithms: the success rate is 91.0%, the avoidance rate is 78.9%, and the utilization rate of shelters is 46.8%. This indicates that the deep reinforcement learning method can effectively reduce risk exposure when approaching the target and possesses tactical mobility. In contrast, although the traditional algorithms perform well in terms of planning efficiency, their adaptability to dynamic threats and the value of shelters is still relatively limited.

This study still has some limitations. The experimental environment is small-scale, and enemy behavior is simplistic; the model's generalization needs further verification in diverse scenarios. Future research will increase the complexity of the map, diversify the types of enemy behaviors, and introduce position prediction optimization (PPO), convolutional neural networks or attention mechanisms to enhance the agent's understanding of dynamic spatial information. This study provides a feasible solution for the movement mechanism of intelligent NPCs in two-dimensional tactical shooting games and offers a reference basis for path planning methods in dynamic environments.

References

[1] Xu L., Zhang W. Survey on Path Planning Based on Deep Reinforcement Learning. 2025 2nd International Conference on Machine Learning and Intelligent Computation, 2025: 685-695.

[2] Tao B., Kim J.-H. Deep reinforcement learning-based local path planning in dynamic environments for mobile robot. Journal of King Saud University – Computer and Information Sciences, 2024, 36(10): 102254.

[3] Khalil A. A., Rahman M. A. FED-UP: Federated Deep Reinforcement Learning-based UAV Path Planning against Hostile Defense System. 2022 18th International Conference on Network and Service Management (CNSM), 2022: 268-274.

[4] Liu Z., Chen B., Zhou H., Koushik G., Hebert M., Zhao D. MAPPER: Multi-Agent Path Planning with Evolutionary Reinforcement Learning in Mixed Dynamic Environments. arXiv preprint arXiv:2007.15724, 2020.

[5] Li Y. Deep Reinforcement Learning: An Overview. arXiv preprint arXiv:1701.07274, 2018.

[6] Arulkumaran K., Deisenroth M. P., Brundage M., Bharath A. A. A Brief Survey of Deep Reinforcement Learning. arXiv preprint arXiv:1708.05866, 2017.

[7] Mnih V., et al. Playing Atari with Deep Reinforcement Learning. arXiv preprint arXiv:1312.5602, 2013.

[8] Mnih V., et al. Human-level control through deep reinforcement learning. Nature, 2015, 518(7540): 529-533.

[9] Schulman J., Wolski F., Dhariwal P., Radford A., Klimov O. Proximal Policy Optimization Algorithms. arXiv preprint arXiv:1707.06347, 2017.

[10] Sutton R. S., Barto A. G. Reinforcement Learning: An Introduction (2nd ed.). MIT Press, 2018.