# Q-Mixing Network for Multi-Agent Pathfinding in Partially Observable Grid Environments

Vasilii Davydov[2,3], Alexey Skrynnik[1], Konstantin S. Yakovlev[1,2], and Aleksandr I. Panov[1,2]

[1] Artificial Intelligence Research Institute FRC CSC RAS, Moscow, Russia
[2] Moscow Institute of Physics and Technology, Moscow, Russia
[3] Moscow Aviation Institute, Moscow, Russia

**Abstract.** In this paper, we consider the problem of multi-agent navigation in partially observable grid environments. This problem is challenging for centralized planning approaches as they typically rely on full knowledge of the environment. To this end, we suggest utilizing the reinforcement learning approach when the agents first learn the policies that map observations to actions and then follow these policies to reach their goals. To tackle the challenge associated with learning cooperative behavior, i.e. in many cases agents need to yield to each other to accomplish a mission. We use a mixing Q-network that complements learning individual policies. In the experimental evaluation, we show that such approach leads to plausible results and scales well to a large number of agents.

**Keywords:** Multi-agent Pathfinding · Reinforcement Learning · Mixing Q-network · Grid Environment

## 1 Introduction

Planning the coordinated movement for a group of intelligent agents is usually considered as a separate direction of behavior planning. Two classes of methods for solving this problem can be distinguished: centralized and decentralized. The methods of the first group are based on the assumption of the existence of a control center, which has access to complete information about the states and movements of agents at any time. In most cases, such methods are based either on reducing the multi-agent planning to other well-known problems, e.g. the boolean satisfiability (SAT-problems) [17], or on heuristic search. Among the latter, algorithms of the conflict-based search (CBS) family are actively developing nowadays. Original CBS algorithm [14] guarantees completeness and optimality. Many enhancements of CBS exist that significantly improve its performance while preserving the theoretical guarantees – ICBS [2], CBSH [5] etc. Other variants of CBS, such as the ones that take the kinematic constraints into account, target bounded-suboptimal solutions [1] are also known. Another widespread approach to centralized multi-agent navigation is prioritized planning [19]. Prioritized planners are extremely fast in practice, but they are incomplete in general. However, when certain conditions have been met the guarantee that any

problem will be solved by a prioritized planning method can be provided [3]. In practice these conditions are often met in the warehouse robotics domains, therefore, prioritized methods are actively used for logistics applications.

Methods of the second class, decentralized, assume that agents are controlled internally and their observations and/or communication capabilities are limited, e.g. they do not have direct access to the information regarding other agent's plans. These approaches are naturally better suited to the settings when only partial knowledge of the environment is available. In this work we focus on one such setting, i.e. we assume that each agent has a limited field of view and can observe only a limited myopic fragment of the environment.

Among the methods for decentralized navigation, one of the most widely used is the ORCA algorithm [18] and its numerous variants. These algorithms at each time step compute the current speed via the construction of the so-called velocity obstacle space. When certain conditions are met, ORCA guarantees that the collision between the agents is avoided, however, there is no guarantee that each agent will reach its goal. In practice, when navigating in a confined space (e.g. indoor environments with narrow corridors, passages, etc.), agents often find themselves in a dead-lock, when they reduce their speed to zero to avoid collisions and stop moving towards goals. It is also important to note that the algorithms of the ORCA family assume that velocity profiles of the neighboring agents are known. In the presented work, such an assumption is not made and it is proposed to use learning methods, in particular – reinforcement learning methods, to solve the considered problem.

The use of reinforcement learning algorithms for path planning in partially observable environments is not new [15,7]. In [9] the authors consider the single-agent case of a partially observable environment and apply the deep Q-network [8] (DQN) algorithm to solve it.

In [11] the multi-agent setting is considered. The authors suggest using the neural network approximator that fits parameters using one of the classic deep reinforcement learning algorithms. However, the full-fledged operation of the algorithm is possible only when the agent's experience is replenished with expert data selected based on the optimal path that is built by the classical scheduler. The approach does not use maximization of the general Q function but tries to solve the problem of multi-agent interaction by introducing various heuristics: an additional loss function for blocking other agents; a reward function that takes into account the collision of agents; other agents' goals encoding in the observation.

We propose to solve the problem using reinforcement learning methods. As a methodological base, we propose to use a mixing Q-network, which implements the principle of monotonic improvement of the overall assessment of the value of the current state based on the current assessments of the value of the state of individual agents. Learning the mixing mechanism based on a parameterized approximator allows to automatically generate rules for resolving conflict patterns when two or more agents pass intersecting path sections. We also propose the implementation of a flexible and efficient experimental environment with trajec-

tory planning for a group of agents with limited observations. The configurability and the possibility of implementing various behavioral scenarios by changing the reward function allow us to compare the proposed method with both classical methods of multi-agent path planning and with reinforcement learning methods designed for training one agent.

## 2 Problem statement

We model multi-agent pathfinding problem in partially observable environment as single-agent task with dynamic obstacles. The process of interaction between the agent and the environment will be modeled by the partially observable Markov decision process (POMDP). The result of the training will be a policy that maps the agent's observation into the action. This function will generate agent behavior and will take into account possible conflict situations.

Markov decision process is described as the set $< S, A, P, r, \gamma >$. $s \in S$ describes a state of the environment. Every step in the environment an agent executes action $a \in A$ and recieves reward with the following reward function $r(s, a) : S \times A \to \mathbb{R}$. This action initiates a state-transition in the environment $P(s'|s, a) : S \times A \times S \to [0, 1]$. The agent uses the information received from the environment to automatically generate the policy function $\pi(a|s) : A \times S \to [0, 1]$. In many cases, it is more efficient to form the value function $Q(a_t, s_t) = r(s_t, a_t) + \mathbb{E}(\sum_{i=1}^{\infty} \gamma^i r(s_{t+i}, a_{t+i}))$, where $\gamma$ is a discount factor.

The POMDP differs from the standard setting in that a separate set of possible observations of the agent $O$ is introduced, which in the general case is not the same as the set of states $S$: $< S, O, A, P, r, \gamma >$. In this case, the agent receives from the environment not its state, but some observation $o$. Accordingly, the strategy and the value function are formed from the observation: $\pi(a|o) : A \times O \to [0, 1]$, $Q(a_t, o_t) = r(s_t, a_t) + \mathbb{E}(\sum_{i=1}^{\infty} \gamma^i r(s_{t+i}, a_{t+i}))$.

In the problem of path-finding for a set of agents, the environment is a graph of a regular structure (cellular field), in which a subset of nodes is responsible for obstacles, another subset - positions available to agents, including goals for agents. The task of the agents is to build the optimal (shortest) path to their goal. In this problem, the state $s \in S$ describes the location of obstacles, agents, and goals of agents. Observation $o \in O$ describes information about the location of obstacles, agents, and goals in a radius around a specific agent. Agents can perform actions to move along the $X$ and $Y$ axes along the cell field.

## 3 Method

In this paper, we propose an original architecture for decision-making and agent training based on a mixing Q-network that uses deep neural network to parameterize the value function by analogy with deep Q-learning (DQN).

In deep Q-learning, the parameters of the neural network are optimized $Q(a, s|\theta)$. Parameters are updated for mini-batches of the agent's experience

data, consisting of sets $< s, a, r, s' >$, where $s'$ is the state in which the agent moved after executing action $a$ in the state $s$.

The loss function for the approximator is:

$$L = \sum_{i=1}^{b} [((r^i + \gamma max_{a^{i+1}} Q(s^{i+1}, a^{i+1}|\theta)) - Q(s^i, a^i|\theta))^2],$$

b is a batch size.

In the transition to multi-agent reinforcement learning, one of the options for implementing the learning process is independent execution. In this approach, agents optimize their own Q-functions for the actions of a single agent. This approach differs from DQN in the process of updating the parameters of the approximator when agents use the information received from other agents. In fact, agents decompose the value function (VDN) [16] and aim to maximize the total Q-function $Q_{tot}(\tau, u)$, which is the sum of the Q-functions of each individual agent $Q^i(a^i, s^i|\theta^i)$.

The Mixing Q-Network (QMIX) [10] algorithm works similarly to VDN. However, in this case, to calculate $Q_{tot}$, a new parameterized function of all Q-values of agents is used. More precisely, $Q_{tot}$ is calculated to satisfy the condition that $Q_{tot}$ increases monotonically with increasing $Q^i$:

$$\frac{\delta Q_{tot}}{\delta Q^i} \geq 0 \; \forall i \; 1 \leq i \leq num\, agents,$$

$Q_{tot}$ is parameterized using a so-called mixing neural Q-network. The weights for this network are generated using the hyper networks [4]. Each of the hyper networks consists of a single fully connected layer with an absolute activation function that guarantees non-negative values in the weights of the mixing network. Biases for the mixing network are generated similarly, however they can be negative. The final bias of the mixing network is generated using a two-layer hyper network with ReLU activation.

The peculiarities of the mixing network operation also include the fact that the agent's state or observation is not fed into the network, since $Q_{tot}$ is not obliged to increase monotonically when changing the state parameters $s$. Individual functions $Q^i$ receive only observation as input, which can only partially reflect the general state of the environment. Since the state can contain useful information for training, it must be used when calculating $Q_{tot}$, so the state $s$ is fed as the input of the hyper network. Thus, $Q_{tot}$ indirectly depends on the state of the environment and combines all Q-functions of agents. The mixing network architecture is shown in Figure 1.

The mixing network loss function looks similar to the loss function for DQN:

$$\sum_{i=1}^{b} [(y^i_{tot} - Q_{tot}(\tau^i, u^i, s^i|\theta))^2]$$

$$y_{tot} = r + \gamma max_{t'} Q_{tot}(\tau', u', s'|\theta^-).$$

Here $b$ is the batch size, $\tau'$ is the action to be performed at the next step after receiving the reward $r$, $u'$ is the observation obtained at the next step, $s'$ is the state obtained in the next step. $\theta^-$ are the parameters of the copy of the mixing Q-network created to stabilize the target variable.

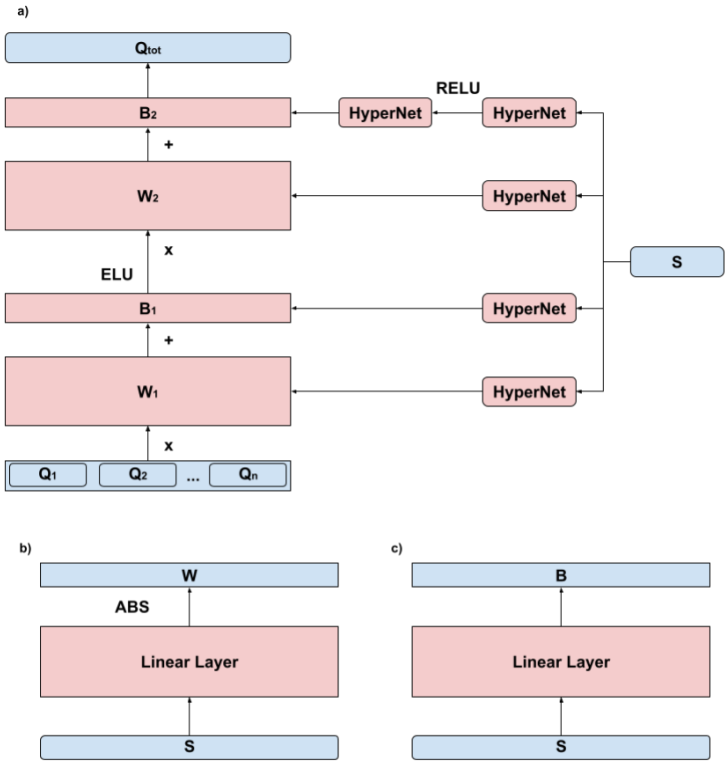


**Fig. 1.** a) Mixing network architecture. $W_1$, $W_2$, $B_1$, $B_2$ are the weights of the mixing network; $Q_1$, $Q_2$ ... $Q_n$ are the agents' Q values; $s$ is the environment state; $Q_{tot}$ is a common Q Value; b) Hyper network architecture for generating the weights matrix of the mixing Q-network. The hyper network consists of a single fully connected layer and an absolute activation function. c) Hyper network architecture for generating the biases of the mixing Q-network. The hyper network consists of a single fully connected layer.

## 4 Experimental environment for multi-agent path finding

The environment is a grid field with agents, their goals, and obstacles located on it. Each agent needs to get to his goal, avoiding obstacles and other agents. An example of an environment showing partial observability for a single agent

is shown in Figure 2. This figure also shows an example of a multi-agent environment.

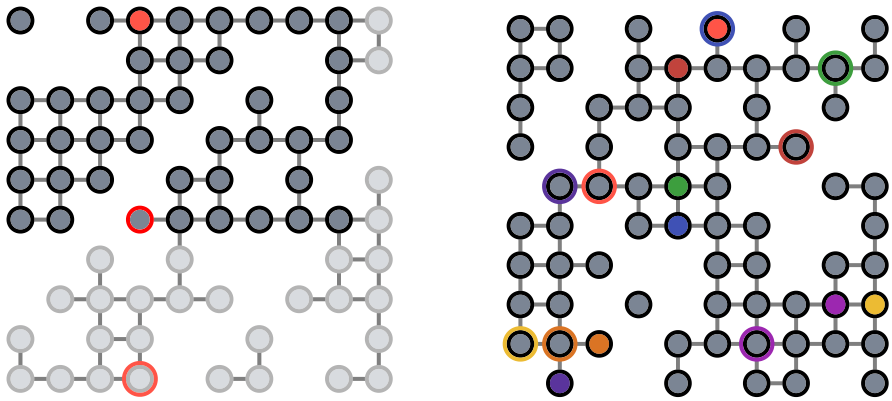

**Fig. 2.** The left figure shows an example of partial observability for a single agent environment: gray vertices are free cells along the edges of which the agent can move; a filled red circle indicates the position of the agent; the vertex with a red outline is the target of this agent, vertex with red border - projection of the agent's goal. The area that the agent cannot see is shown as transparent. The right figure shows an example of an environment for 8 agents, projections of agents' goals and partial observability are not shown for visibility purposes.

The input parameters for generating the environment are:

- field size $E_{size} \geq 2$,
- obstacle density $E_{density} \in [0, 1)$,
- number of agents in the environment $E_{agents} \geq 1$,
- observation radius: agents get $1 \leq R \leq E_{size}$ cells in each direction,
- the maximum number of steps in the environment before ending $E_{horizon} \geq 1$,
- the distance to the goal for each agent $E_{dist}$ (is an optional parameter, if it is not set, the distance to the goal for each agent is generated randomly).

Obstacle matrix is filled randomly by parameters $E_{size}$ and $E_{density}$. The positions of agents and their goals are also generated randomly, but with a guarantee of reachability.

The observation space $O$ of each agent is a multidimensional matrix: $O : 4 \times \big(2 \times R + 1\big) \times \big(2 \times R + 1\big)$, which includes the following 4 matrices. *Obstacle matrix*: 1 encodes an obstacle, and 0 encodes its absence. If any cell of the agent's field of view is outside the environment, then it is encoded 1. *Agents' positions matrix*: 1 encodes other agent in the cell, and 0 encodes his absence. The value in the center is inversely proportional to the distance to the agent's goal. *Other agents' goals matrix*: 1 if there is a goal of any agent in the cell, 0 – otherwise.

*Self agent's goal matrix* if the goal is inside the observation field, then there is 1 in the cell, where it is located, and 0 in other cells. If the target does not fall into the field of view, then it is projected onto the nearest cell of the observation field. As a cell for projection, a cell is selected on the border of the visibility area, which either has the same coordinate along with one of the axes as the target cell or if there are no such cells, then the nearest corner cell of the visibility area is selected. An example of an agent observation space is shown in Figure 3.

**Fig. 3.** An example of an observation matrix for a purple agent. In all observation cells, 1 means that there is an object (obstacle, agent, or goal) in this cell and 0 otherwise. a) Environment state. The agent for which the observation is shown is highlighted; b) Obstacle map. The central cell corresponds to the position of the agent, in this map the objects are obstacles; c) Agents map. In this map, the objects are agents; d) Other agents' goals map. In this map, the objects are the goals of other agents; e) Goal map. In this map, the object is the self-goal of the agent.

Each agent has 5 actions available: stay in place and move vertically (up or down) or horizontally (right or left). An agent can move to any free cell that is not occupied by an obstacle or other agent. If an agent moves to a cell with his own goal, then he is removed from the map and the episode is over for him.

Agents receive a reward of 0.5 if he follows one of the optimal routes to his goal, $-1$ if the agent has increased his distance to the target and $-0.5$ if the agent stays in place.

## 5 Experiments

This section compares QMIX with the Proximal Policy Optimization (PPO), single-agent reinforcement learning algorithm [13]. We chose PPO because it showed better results in a multi-agent setting compared to other modern reinforcement learning algorithms. Also, this algorithm significantly outperformed other algorithms, including QMIX, in the single-agent case.

The algorithms were trained on environments with the following parameters: $E_{size} = 15 \times 15$, $E_{density} = 0.3$, $E_{agents} = 2$, $R = 5$, $E_{horizon} = 30$, $E_{dist} = 8$. As the main network architecture of each of the algorithms, we used architecture with two hidden layers of 64 neurons, with ReLU activation function for QMIX and Tanh for PPO. We trained each of the algorithms using 1.5M steps in the environment.

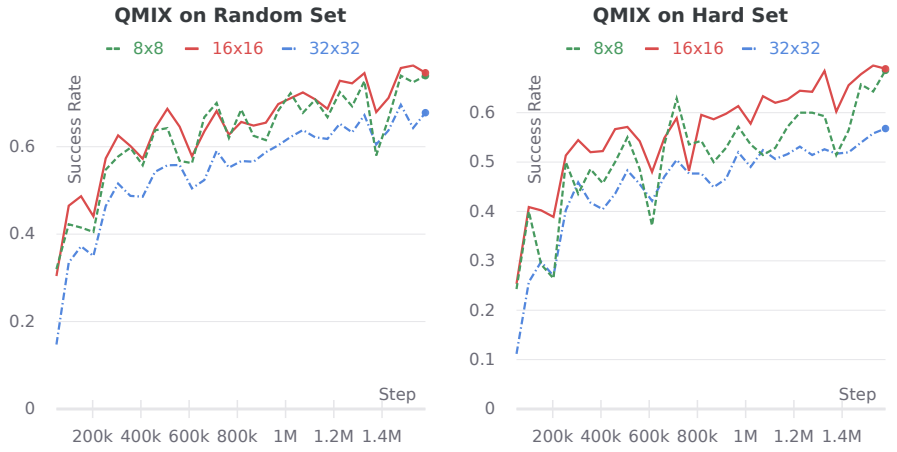


**Fig. 4.** The graphs show separate curves for different environment sizes. For environment sizes $8 \times 8$; $16 \times 16$; $32 \times 32$, we used 2, 6, 16 agents, respectively. The left graph shows the success rate for the QMIX algorithm in random environments. The right graph shows the success rate for the QMIX algorithm in complex environments.

The results of training of the QMIX algorithm are shown in Figure 4. We evaluated the algorithm for every $10^5$ step on a set of unseen environments. The evaluation set was fixed throughout the training. This figure also shows evaluation curves for complex environments. We generated a set of complex environments so that agents needed to choose actions cooperatively, avoiding deadlocks. An example of complex environments for a environment size of $8 \times 8$ is shown in Figure 5. This series of experiments aimed to test the ability of QMIX agents to act sub-optimally for a greedy policy, but optimal for a cooperative policy.

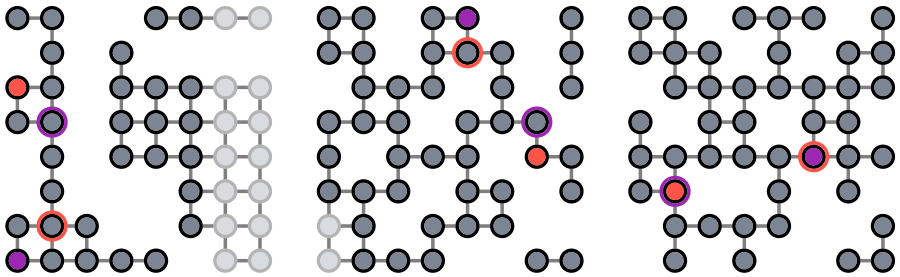

**Fig. 5.** Examples of complex $8 \times 8$ environments where agents need to use a cooperative policy to reach their goals. In all examples, the optimal paths of agents to their goals intersect, and one of them must give way to the other. Vertices that are not visible to agents are shown as transparent.

The results of the evaluation are shown in Table 1 for random environments and in Table 2 for complex environments. As a result of training, the QMIX significantly outperforms the PPO algorithm on both series of experiments, which shows the importance of using the mixing network for training.

**Table 1.** Comparison of the algorithms on a set of 200 environments (for each parameter set) with randomly generated obstacles. The last two columns show the success rate for PPO and QMIX, respectively. The results are averaged over three runs of each algorithm in each environment. QMIX out-performs PPO due to the use of the mixing Q-function $Q_{tot}$.

| $E_{size}$ | $E_{agents}$ | $R$ | $E_{horizon}$ | $E_{dist}$ | $E_{density}$ | PPO | QMIX |
|---|---|---|---|---|---|---|---|
| $8 \times 8$ | 2 | 5 | 16 | 5 | 0.3 | 0.539 | **0.738** |
| $16 \times 16$ | 6 | 5 | 32 | 6 | 0.3 | 0.614 | **0.762** |
| $32 \times 32$ | 16 | 5 | 64 | 8 | 0.5 | 0.562 | **0.659** |

**Table 2.** Comparison on a set of 70 environments (for each parameter set) with complex obstacles. The last two columns show the success rate for PPO and QMIX, respectively. The results are averaged over ten runs of each algorithm in each environment. QMIX, as in the previous experiment, out-performs PPO.

| $E_{size}$ | $E_{agents}$ | $R$ | $E_{horizon}$ | $E_{dist}$ | $E_{density}$ | PPO | QMIX |
|---|---|---|---|---|---|---|---|
| $8 \times 8$ | 2 | 5 | 16 | 5 | 0.3 | 0.454 | **0.614** |
| $16 \times 16$ | 6 | 5 | 32 | 6 | 0.3 | 0.541 | **0.66** |
| $32 \times 32$ | 16 | 5 | 64 | 8 | 0.5 | 0.459 | **0.529** |

## 6 Conclusion

In this work, we considered the problem of multi-agent pathfinding in the partially observable environment. This formulation with incomplete information makes it impossible to apply the classical methods of multi-agent pathfinding, but it allows the use of reinforcement learning algorithms. It was proposed to apply a mixing Q-network with a neural network approximation to solve this problem, which selects the parameters of a unifying Q-function that combines the Q-functions of individual agents. An experimental environment was developed for launching experiments with learning algorithms. In this environment, the efficiency of the proposed method was demonstrated and its effectiveness outperforms the on-policy reinforcement learning algorithm design only for the single-agent case. It should be noted that the comparison was carried out under conditions of limiting the number of episodes of interaction between the

agent and the environment. If such a sample efficiency constraint is removed, the on-policy method can outperform the proposed off-policy Q-mixing network algorithm. In future work, we plan to combine the advantages of better-targeted behavior generated by the on-policy method and the ability to take into account the actions of another agent when resolving local conflicts using QMIX. The model-based reinforcement learning approach seems promising, in which it is possible to plan and predict the behavior of other agents and objects in the environment [12,6]. We also assume that using adaptive task composition for agent training (curriculum learning) will also give a significant performance boost for tasks with a large number of agents

## References


1. Barer, M., Sharon, G., Stern, R., and Felner, A. Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem. In *Proceedings of The 7th Annual Symposium on Combinatorial Search (SoCS 2014)* (July 2014), pp. 19–27.
2. Boyarski, E., Felner, A., Stern, R., Sharon, G., Betzalel, O., Tolpin, D., and Shimony, E. ICBS: Improved conflict-based search algorithm for multi-agent pathfinding. In *Proceedings of The 24th International Joint Conference on Artificial Intelligence (IJCAI 2015)* (2015), pp. 740–746.
3. Čáp, M., Novák, P., Kleiner, A., and Selecký, M. Prioritized planning algorithms for trajectory coordination of multiple mobile robots. *IEEE Transactions on Automation Science and Engineering 12*, 3 (2015), 835–849.
4. D. Ha, A. Dai, Q. L. Hypernetworks. In *Proceedings of the International Conference on Learning Representations* (2016).
5. Felner, A., Li, J., Boyarski, E., Ma, H., Cohen, L., Kumar, T. S., and Koenig, S. Adding heuristics to conflict-based search for multi-agent path finding. In *Proceedings of the 28th International Conference on Automated Planning and Scheduling (ICAPS 2018)* (2018), pp. 83–87.
6. Gorodetskiy, A., Shlychkova, A., and Panov, A. I. Delta Schema Network in Model-based Reinforcement Learning. In *Artificial General Intelligence. AGI 2020. Lecture Notes in Computer Science* (2020), B. Goertzel, A. Panov, A. Potapov, and R. Yampolskiy, Eds., vol. 12177, Springer, pp. 172–182.
7. Martinson, M., Skrynnik, A., and Panov, A. I. Navigating Autonomous Vehicle at the Road Intersection Simulator with Reinforcement Learning. In *Artificial Intelligence. RCAI 2020. Lecture Notes in Computer Science* (2020), S. O. Kuznetsov, A. I. Panov, and K. S. Yakovlev, Eds., vol. 12412, Springer International Publishing, pp. 71–84.
8. Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., et al. Human-level control through deep reinforcement learning. *nature 518*, 7540 (2015), 529–533.
9. Panov, A. I., Yakovlev, K. S., and Suvorov, R. Grid path planning with deep reinforcement learning: Preliminary results. *Procedia computer science 123* (2018), 347–353.
10. Rashid, T., Samvelyan, M., Schroeder, C., Farquhar, G., Foerster, J., and Whiteson, S. Qmix: Monotonic value function factorisation for deep multi-

agent reinforcement learning. In *International Conference on Machine Learning* (2018), PMLR, pp. 4295–4304.

11. Sartoretti, G., Kerr, J., Shi, Y., Wagner, G., Kumar, T. S., Koenig, S., and Choset, H. Primal: Pathfinding via reinforcement and imitation multi-agent learning. *IEEE Robotics and Automation Letters 4*, 3 (2019), 2378–2385.
12. Schrittwieser, J., Hubert, T., Mandhane, A., Barekatain, M., Antonoglou, I., and Silver, D. Online and Offline Reinforcement Learning by Planning with a Learned Model.
13. Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).
14. Sharon, G., Stern, R., Felner, A., and Sturtevant., N. R. Conflict-based search for optimal multiagent path finding. *Artificial Intelligence Journal 218* (2015), 40–66.
15. Shikunov, M., and Panov, A. I. Hierarchical Reinforcement Learning Approach for the Road Intersection Task. In *Biologically Inspired Cognitive Architectures 2019. BICA 2019. Advances in Intelligent Systems and Computing* (2020), A. V. Samsonovich, Ed., vol. 948, Springer, pp. 495–506.
16. Sunehag, P., Lever, G., Gruslys, A., Czarnecki, W. M., Zambaldi, V., Jaderberg, M., and Lanctot, M. Value-decomposition networks for cooperative multi-agent learning based on team reward. In *Proceedings of the 17th International Conference on Autonomous Agents and Multiagent Systems* (2017).
17. Surynek, P., Felner, A., Stern, R., and Boyarski, E. Efficient sat approach to multi-agent path finding under the sum of costs objective. In *Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI 2016)* (2016), IOS Press, pp. 810–818.
18. Van Den Berg, J., Guy, S. J., Lin, M., and Manocha, D. Reciprocal n-body collision avoidance. In *Robotics research*. Springer, 2011, pp. 3–19.
19. Yakovlev, K., Andreychuk, A., and Vorobyev, V. Prioritized multi-agent path finding for differential drive robots. In *Proceedings of the 2019 European Conference on Mobile Robots (ECMR 2019)* (2019), IEEE, pp. 1–6.