Frontiers of Information Technology & Electronic Engineering
www.jzus.zju.edu.cn; engineering.cae.cn; www.springerlink.com
ISSN 2095-9184 (print); ISSN 2095-9230 (online)
E-mail: jzus@zju.edu.cn

Coach-assisted multi-agent reinforcement learning framework for unexpected crashed agents*

Jian ZHAO†1, Youpeng ZHAO[1], Weixun WANG[2], Mingyu YANG[1], Xunhan HU[1], Wengang ZHOU†‡1, Jianye HAO[2], Houqiang LI†‡1

[1]*School of Information Science and Technology, University of Science and Technology of China, Hefei 230026, China*

[2]*College of Intelligence and Computing, Tianjin University, Tianjin 300072, China*

†E-mail: zj140@mail.ustc.edu.cn; zhwg@ustc.edu.cn; lihq@ustc.edu.cn

Received Dec. 31, 2021; Revision accepted Apr. 8, 2022; Crosschecked May 26, 2022

Abstract: Multi-agent reinforcement learning is difficult to apply in practice, partially because of the gap between simulated and real-world scenarios. One reason for the gap is that simulated systems always assume that agents can work normally all the time, while in practice, one or more agents may unexpectedly "crash" during the coordination process due to inevitable hardware or software failures. Such crashes destroy the cooperation among agents and lead to performance degradation. In this work, we present a formal conceptualization of a cooperative multi-agent reinforcement learning system with unexpected crashes. To enhance the robustness of the system to crashes, we propose a coach-assisted multi-agent reinforcement learning framework that introduces a virtual coach agent to adjust the crash rate during training. We have designed three coaching strategies (fixed crash rate, curriculum learning, and adaptive crash rate) and a re-sampling strategy for our coach agent. To our knowledge, this work is the first to study unexpected crashes in a multi-agent system. Extensive experiments on grid-world and StarCraft II micromanagement tasks demonstrate the efficacy of the adaptive strategy compared with the fixed crash rate strategy and curriculum learning strategy. The ablation study further illustrates the effectiveness of our re-sampling strategy.

Key words: Multi-agent system; Reinforcement learning; Unexpected crashed agents
https://doi.org/10.1631/FITEE.2100594 **CLC number:** TP18

1 Introduction

Cooperative multi-agent systems widely exist in various domains, where a group of agents need to coordinate with each other to maximize the team's reward (Busoniu et al., 2008; Tuyls and Weiss, 2012). Such a setting can be broadly applied in the control and operation of robots, unmanned vehicles, mobile sensor networks, and the smart grid (Zhang et al., 2021). Recently, many researchers have devoted their efforts to leveraging reinforcement learning techniques in multi-agent systems (Rashid et al., 2018; Sunehag et al., 2018; Wang JH et al., 2020; Wang YP et al., 2020). Despite the remarkable advancement in academia, multi-agent reinforcement learning (MARL) is still difficult to apply in practice. One non-trivial reason is that there always exists a gap between simulated and real-world scenarios, which degrades the performance of the policies once the models are transferred into real-world applications (Zhao et al., 2020).

To close this sim-to-real gap and accomplish more efficient policy transfer, multiple research

‡ Corresponding authors

* Project supported by the National Natural Science Foundation of China (No. 61836011), the Youth Innovation Promotion Association of the Chinese Academy of Sciences (No. 2018497), and the GPU cluster built by the MCC Lab of Information Science and Technology Institution, USTC, China

ORCID: Jian ZHAO, https://orcid.org/0000-0003-4895-990X; Wengang ZHOU, https://orcid.org/0000-0003-1690-9836; Houqiang LI, https://orcid.org/0000-0003-2188-3028

efforts are now being directed to identifying the causes of the gap and proposing corresponding solutions. One main cause is the difference between the physics engine of the simulator and the real-world scenario. To alleviate the difference, research efforts have been directed to building up more realistic simulators using mathematical models (Todorov et al., 2012; Furrer et al., 2016; Dosovitskiy et al., 2017; Shah et al., 2018; McCord et al., 2019; Wang YP et al., 2021). Another cause is the mismatch between the simulated environment's data distribution and the real environment, which has inspired related research on domain adaptation (Higgins et al., 2017; Traoré et al., 2019; Arndt et al., 2020) and domain randomization (Tobin et al., 2017).

Generally, simulated systems always assume that agents can work normally all the time. However, this assumption is usually not in line with reality. Because of the inevitable hardware or software failures in practice, one or more agents may unexpectedly "crash" during the coordination process. If the agents are trained in an environment without crashes, they only master how to cooperate in a crash-free environment. Once some agents "break down" and take abnormal actions, the remaining agents can hardly maintain effective cooperation, which will lead to performance degradation. Take a two-agent system as an example: two agents are required to finish two tasks in coordination. In the crash-free scenario, the optimal solution is for each agent to take responsibility for one task. When applying such a policy to the real-world application, the cooperation cannot be accomplished if any agent encounters a crash. This example indicates the necessity of considering unexpected crashes during training to obtain well-trained agents with high robustness.

To our knowledge, this work is the first to study crashes in multi-agent systems, which is more consistent with real-world scenarios. In this study, we give a formal conceptualization of a cooperative MARL system with unexpected crashes, where any agent has a certain probability of crashing during operation. We assume that, for each agent, the probability of crashing independently follows a Bernoulli distribution. To enhance the robustness of the system to unexpected crashes, the agents should be trained in an environment that includes crashes. The key challenge is how to adjust the crash rate during training.

In this work, we propose a coach-assisted MARL framework that introduces a virtual coach agent into the system. The coach agent is responsible for adjusting the crash rate during training. One straightforward coaching strategy for the coach is to set a fixed crash rate during training. Considering that it may be too difficult for agents to cooperate initially (Narvekar et al., 2020), increasing the crash rate gradually is another feasible strategy. In addition to these basic strategies, an experienced coach can automatically adjust the crash rate corresponding to the overall performance during training. Specifically, if the performance exceeds the threshold, the crash rate is increased to increase the learning difficulty; otherwise, the crash rate should be decreased. In this way, agents can learn coordination skills progressively while being exposed to unexpected crashes.

To test the effectiveness of our method, we have conducted experiments on grid-world and StarCraft II micromanagement tasks. Compared to the fixed crash rate and curriculum learning strategies, the results demonstrate that an adaptive method achieves relatively stable performances with different crash rates. Furthermore, the ablation study shows the efficacy of our re-sampling strategy.

2 Related works

In this section, we briefly summarize the works related to cooperative MARL. With the development of this field, researchers are paying more and more attention to the MARL problem, which is more consistent with real-world settings.

Early efforts treat the agents in a team independently and regard the team reward as the individual reward (Tan, 1993; Mnih et al., 2015; Foerster et al., 2017; Omidshafiei et al., 2017). Consequently, the MARL task is transformed into multiple single-agent reinforcement learning tasks. Although trivially providing a possible solution, these approaches pay insufficient attention to an essential characteristic of MARL—coordination among agents. In other words, it will bring non-stationarity that agents cannot distinguish between the stochasticity of the environment and the exploitative behaviors of other co-learners (Lowe et al., 2017).

Another line of research focuses on centralized learning of joint actions, which can naturally consider coordination problems (Sukhbaatar et al., 2016;

Peng et al., 2017). Most of the centralized learning approaches require communication during execution. For instance, in CommNet (Sukhbaatar et al., 2016) a centralized network is designed for agents to exchange information. BicNet (Peng et al., 2017) leverages bi-directional recurrent neural networks (RNNs) for information sharing. Considering the communication constraint in practice, SchedNet (Kim et al., 2019) was proposed, in which agents learn how to schedule themselves for message passing and how to select actions based on received partial observations. Another challenge of centralized learning is the scalability issue because the joint action space grows exponentially as the number of agents increases. Some researchers investigated scalable strategies in centralized learning (Guestrin et al., 2001; Kok and Vlassis, 2006). Sparse cooperative Q-learning (Kok and Vlassis, 2006) allows only the necessary coordination between agents by encoding such dependencies. However, these methods require prior knowledge of the dependencies among agents, which is often inaccessible.

To study a more practical scenario with the partial observability and communication constraint, an emerging stream is the paradigm of centralized training with decentralized execution (CTDE) (Oliehoek et al., 2008; Kraemer and Banerjee, 2016). To our knowledge, value decomposition networks (VDNs) (Sunehag et al., 2018) make the first attempt to decompose a central state-action value function into a sum of individual Q-values to allow for decentralized execution. VDN simply assumes the equal contributions of agents and does not use additional state information during training. Based on VDN, QATTEN (Yang et al., 2020) uses a multi-head attention structure to distinguish the contributions of agents, and linearly integrates the individual Q-values into the central Q-value. Instead of using linear monotonic value functions, QMIX (Rashid et al., 2018) and QTRAN (Son et al., 2019) employ a mixing network that satisfies the individual-global-max (IGM) principle (Son et al., 2019) to combine the individual Q-values non-linearly by leveraging state information. QPLEX (Wang JH et al., 2020) introduces the duplex dueling structure and decomposes the central Q-value into the sum of individual value functions and a non-positive advantage function.

However, all of the existing works assume that agents can continuously maintain normal operations, which is inconsistent with real-world scenarios. As a matter of fact, it is a quite common phenomenon that some agents encounter unexpected crashes because of hardware or software failures. To this end, we aim to study a more practical problem by considering unexpected crashed agents in the cooperative MARL task.

3 Problem formulation

To better solve the problem of an unexpected crashed agent, we define a Crashed Dec-POMDP model, which is defined by a tuple $M =< N, \boldsymbol{S}, \boldsymbol{A}, \boldsymbol{\Omega}, P, O, R, \gamma, \alpha >$, where $\boldsymbol{S}$, $\boldsymbol{A}$, $\boldsymbol{\Omega}$, P, O, and R represent the state space, the action space, the observation space, a state transition function, an observation probability function, and a team reward, respectively. Each agent $g_i \in N \equiv \{g_1, g_2, \cdots, g_n\}$ has a probability of crashing and the crash rate is denoted as α. For simplicity, we assume that the crash occurs at the beginning of the episode and that the status of being crashed or not will not change throughout the episode. We define a binarized vector to denote the crashed state of n agents as $[c_i]_{i=1}^n$, where $c_i \sim \text{Bernoulli}(\alpha)$. When the i^{th} agent crashes, c_i is 1; otherwise, c_i is 0. Note that $[c_i]_{i=1}^n$ stays the same during an episode but may change throughout the task due to randomness.

At each time step, each agent g_i receives partial observation $o_i \in \boldsymbol{\Omega}$ according to the observation probability function $O(o_i|s)$. Each uncrashed agent chooses an action $a_i \in \boldsymbol{A}$ with the normal strategy, while the crashed agents take no-move or random actions, forming a joint action $\boldsymbol{a} = [a_i]_{i=1}^n$. Given the current state s, the joint action $\boldsymbol{a}$ of the agents transits the environment to the next state $s' \in \boldsymbol{S}$ according to the state transition function $P(s'|s, \boldsymbol{a})$. All of the agents share a team reward $R(s, \boldsymbol{a})$. The learning goal of MARL is to optimize every agent's individual policy $\pi_i(a_i|\tau_i)$, where $\tau_i = (o_i^0, a_i^0, o_i^1, a_i^1, \cdots, o_i^T, a_i^T)$ is an agent's action-observation history, to maximize the team reward accumulation $\sum_{t=0}^{\infty} \gamma^t R(s^t, \boldsymbol{a}^t)$, where $\gamma \in [0, 1)$ is a discount factor.

4 Methods

In this section, we present our coach-assisted MARL framework for the Crashed Dec-POMDP

problem and explain the rationality of our design.

4.1 Overall framework

To simulate crash scenarios during training, we introduce a virtual agent into the system to act as a coach. The coach is responsible for deciding the crash rate during training. At the beginning of each episode t, the coach sets up a crash rate α_t. We assume that the probability of being crashed for each agent follows a Bernoulli(α_t) distribution. Given the current crash rate α_t, some of the agents crash and cannot take rational actions. Then, the multi-agent system with crashed agents is trained for T steps to learn coordination. Then the coach can receive the performance of the agents under the current crash rate, denoted as e_t, and reset the crash rate α_{t+1} for the next episode. To sum up, the overall framework is illustrated in Fig. 1.

4.2 Coaching strategies

The main challenge for the coach is how to choose an effective crash rate during training. Here, we introduce three coaching strategies:

1. Fixed crash rate. The coach sets a fixed crash rate that is used throughout the training process. The agents, some of which are crashed, are required to learn coordination skills from scratch.

2. Curriculum learning. The coach linearly increases the crash rate during training. At the beginning, the agents are trained in a crash-free environment. For the t^{th} episode, the coach sets the crash rate to be $(t-1)\Delta\alpha$, where $\Delta\alpha$ is a hyperparameter. This approach gradually increases the cooperation difficulty.

3. Adaptive crash rate. For the first two strategies, the coach does not take full advantage of the performance of the cooperative agents. An advanced strategy for the coach is to adaptively adjust the crash rate to correspond to the performance of the agents at the current crash rate. The basic idea is that if the agents can cooperate well and achieve acceptable performance under the current crash situation, the crash rate should be increased; otherwise, the crash rate should be decreased. The adaptive strategy can be formulated as follows:

$$\alpha_{t+1} = F(\alpha_t, e_t, \beta), \tag{1}$$

where $F(\cdot)$ is a mapping function, and β represents the threshold of the performance of the specific evaluation metric. We can see that the fixed crash rate and curriculum learning are two special cases of the adaptive strategy. For the fixed crash rate strategy,

$$F(\alpha_t, e_t, \beta) = \alpha_t. \tag{2}$$

For the curriculum learning strategy,

$$F(\alpha_t, e_t, \beta) = \alpha_t + \Delta\alpha, \tag{3}$$

where $\alpha_1 = 0$.

In this work, we use the following adaptive function:

$$F(\alpha_t, e_t, \beta) = \alpha_t + \rho(I(e_t \geq \beta) - \alpha_t), \tag{4}$$

where ρ is the learning rate of the crash rate, and function $I(\cdot)$ is defined as follows:

$$I(e_t) = \begin{cases} 1, & e_t \geq \beta, \\ 0, & e_t < \beta. \end{cases} \tag{5}$$

In this adaptive function, if the performance of the system e_t does not reach threshold value β, the crash rate for the following training will be reduced, and vice versa. Therefore, the crash rate during training can fit the skills of the system, thus facilitating the learning process. Note that our method is not limited to the use of the above function. A more efficient adaptive function can be further investigated.

4.3 Re-sampling strategy

Randomly sampling from a Bernoulli(α) distribution may cause the proportion of the crashed agents to exceed or be smaller than the current crash ratio α. Therefore, we employ a re-sampling strategy to ensure that the number of crashed agents is not longer than the upper bound of $n \times \alpha$. Here, we explain the rationality behind the re-sampling strategy. For the samples with more crashed agents, it may be too difficult for the current model to learn the coordination skills, and thus these samples are discarded and new samples will be generated. Samples with fewer crashed agents than expected can help the agents remember how to deal with the easier scenarios, and are therefore used during training.

4.4 Overview of our algorithm

To give a clear description of our method, we use a coaching strategy with an adaptive crash rate as an example to give a complete description of our algorithm, which is shown in Algorithm 1.

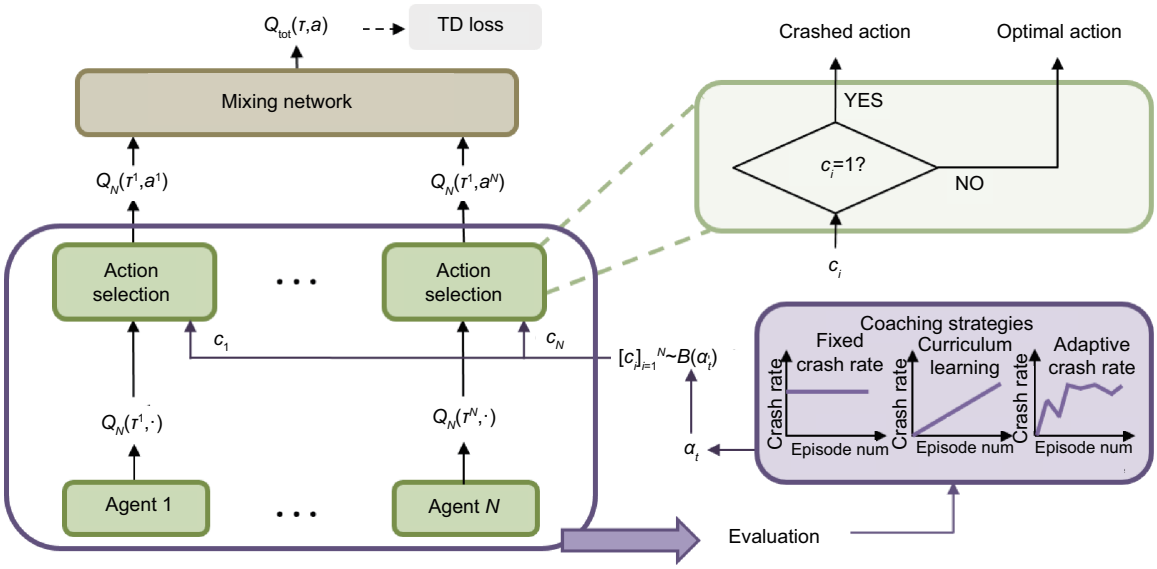

Fig. 1 An overview of the adaptive framework

5 Experiments

In this section, we discuss the experiments we conducted to demonstrate the effectiveness of the methods that we propose. First, we conducted experiments in a grid-world environment as a toy example. Then we used the StarCraft Multi-Agent Challenge (SMAC) environment (Samvelyan et al., 2019) as the test-bed to evaluate our methods, which has become a commonly used benchmark for evaluating state-of-the-art MARL approaches. All experiments were conducted on a Ubuntu 18.04 server with four Intel® Xeon® Gold® 6252 CPUs @ 2.10 GHz and a GeForce RTX 2080Ti GPU. Our codes are available at https://github.com/youpengzhao/Crashed_Agent.

5.1 Grid-world example

5.1.1 Settings

We used the grid-world example to intuitively show the consequences without considering unexpected crashes in real-world scenarios. We set a 10×10 grid where two agents needed to touch two buttons within a limited number of steps. The game terminated after 20 steps or when both buttons had been touched. The default reward at each step was –1 and if one button was touched, the agents were assigned a reward of five at this step. In this way, the agents were encouraged to touch the button as quickly as possible. At each step, each agent had five possible actions including up, down, left, right, and staying still. If there was an unexpected crash during the test, the crashed agent remained still in the whole episode, so only one agent crashed during the test. For simplicity, the initial locations of agents and buttons were fixed, so the environment was deterministic. In addition, the observation of each agent was its own location, and the global state contained the locations of the two buttons and agents, so the agents did not know whether their partner crashed based on its own observation during execution.

We used QMIX (Rashid et al., 2018), a state-of-the-art value-based MARL algorithm, as the base model in this toy example, and adopted the adaptive approach for comparison. Our implementation was based on the Pymarl Algorithm Library (Samvelyan et al., 2019), and the training schedules, optimizer, and training hyperparameters were kept the same as the default ones used in Pymarl. Our method includes two additional hyperparameters: one is the performance threshold β to decide whether to increase or decrease the crash rate during training; the other is the learning rate of the crash rate ρ to control the step size of adjustment of α. We set β as 0.75 and ρ as 0.01 in this experiment. These tasks were separately trained for two million steps.

Algorithm 1 Coach-assisted MARL framework with the adaptive crash rate strategy

Input: performance threshold β, learning rate of the crash rate ρ, test interval T, number of agents n, and maximum number of steps $t_{\max}$.

Output: individual Q-values for agents.

Initialize $t_{\text{last\_test}}=0$, $t_{\text{step}}=0$, and $\alpha=0$;

while $t_{\text{step}}<t_{\max}$ **do**
  For each episode, obtain the initial state s_0, and set $t=0$;
  Sample a binarized n-dimensional vector $\boldsymbol{c}$ according to a distribution of Bernoulli(α), such that $||\boldsymbol{c}||_1<\lceil n\alpha\rceil$;
  while not terminal **do**
    for every agent i **do**
      if $c_i==1$ **then**
        Agent i is crashed and a random action is taken;
      else
        Agent i executes ϵ-greedy normally;
      end if
    end for
    Interact with the environment and obtain state s_{t+1};
    $t\leftarrow t+1$;
  end while
  $t_{\text{step}}\leftarrow t_{\text{step}}+t$;
  Use the data for training;
  if $t_{\text{step}}-t_{\text{last\_test}}\geq T$ **then**
    $t_{\text{last\_test}}=t_{\text{step}}$;
    Evaluate the model with crash rate α and obtain the performance ω;
    if $\omega>\beta$ **then**
      $\alpha\leftarrow\alpha+\rho(1-\alpha)$;
    else
      $\alpha\leftarrow\alpha+\rho(0-\alpha)$;
    end if
  end if
end while

5.1.2 Performance evaluation and discussion

After training for the same number of steps, agents trained using these two methods managed to complete this task under normal scenarios. However, when an unexpected crash occurred, things were different. The results are illustrated in Fig. 2. The agent trained using QMIX learned to touch the button near it in the shortest path, but after that, it wandered aimlessly. Due to partial observation, the normal agent failed to know that its partner was out of control and it did not try to touch another button. We assumed that agents trained by QMIX learned to efficiently cooperate to complete this task, and therefore they just needed to touch the closest button. However, their excessive reliance on cooperation made the system fragile and they failed to deal with the unexpected crash, which is common in realistic scenarios. In fact, the optimal strategy for the system when encountering an unexpected crash is for each agent to touch another button after touching the button nearest itself. In this way, even if one agent "breaks down," the system can still complete the task. As shown in Fig. 2, our method takes possible crashes into account during the training, so it can still fulfill the task even when one of the two agents encounters a crash. This toy example illustrates the drawback of overreliance on cooperation and the necessity of considering possible crashes when training the multi-agent system.

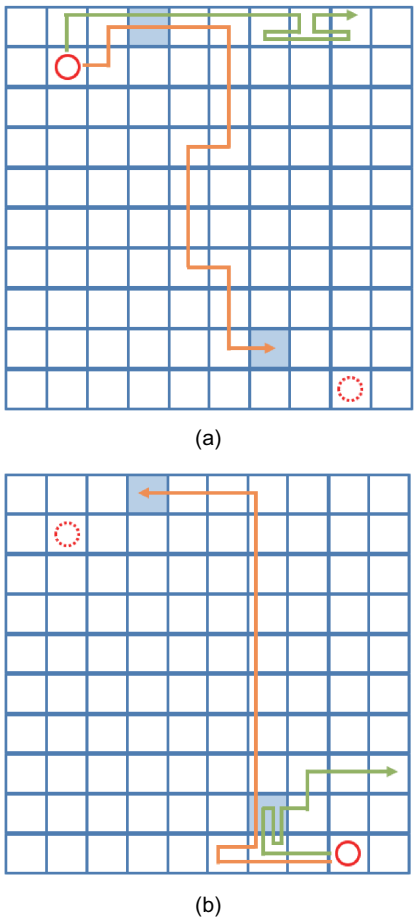

Fig. 2 The trajectory of agents during the test when one of them is crashed: (a) agent 1 is crashed; (b) agent 2 is crashed. The agents are represented with circles; the crashed one is marked using a dotted line and the normal one is marked using a solid line. The colored grids symbolize the two buttons. The green arrow line is the trajectory of agents trained using the original QMIX and the orange one is achieved by our method. References to color refer to the online version of this figure

5.2 StarCraft Multi-Agent Challenge

5.2.1 Settings

In addition to the grid-world experiment, we conducted experiments on StarCraft II decentralized micromanagement tasks to show the effectiveness of our method. In this environment, we assumed that the crashed agents would take random actions. In this experiment, we also used QMIX (Rashid et al., 2018) as the base model. Then we compared the performance of QMIX and our coach-assisted framework with fixed crash rate, curriculum learning, and adaptive crash rate coaching strategies. Our implementation was also based on the Pymarl Algorithm Library (Samvelyan et al., 2019) without changing the default training schedules. For the variants of QMIX with the fixed crash rate, we randomly sampled the crashed agents with a Bernoulli distribution during each episode; thus, the actual number of crashed agents ranged from 0 to n. In the curriculum learning coaching strategy, the crash rate increased from 0 linearly and the upper limit was set to 0.1 as we test the models in scenarios whose crash rate was at most 0.1. We set the two hyperparameters β in $\{0.60, 0.65, 0.70, 0.75\}$ and ρ in $\{0.001, 0.003, 0.005, 0.015\}$, and selected their optimal values based on a grid search when adopting our adaptive method. We repeated the experiments in each setting over five runs with different seeds and reported the average results. For all the compared methods, each task was separately trained for two million steps. To obtain a relatively robust evaluation result, each model was tested 128 times.

We chose two standard maps and designed two different maps in the experiment: 3s_vs_5z, 3s5z_vs_3s5z, 8m_vs_5z, and 8s_vs_3s5z. The two standard maps were well-matched in strength, so a crash could result in some imbalance. To comprehensively show the performance of our method, we also designed two maps that guaranteed an appropriate gap in strength between the two sides, so that unexpected crashes would not lead to a significant change in difficulty. For more details about the maps, please refer to Samvelyan et al. (2019).

5.2.2 Observation

In this part, we discuss the observations from the scenario with crashed agents on StarCraft II micromanagement tasks and show what must be considered to deal with the crash scenario.

The agents in Figs. 3a and 3b play the role of Marines (ours) that are good at long-range attacks, while Zealots (opponents) can attack only in a short range and they have the same moving speed. Because the health point of Marines is only half that of Zealots, the optimal strategy is to alternate fire to attract the enemies. In Fig. 3a, one agent (highlighted with a red rectangle) is out of control and starts to take random actions, while one of the remaining agents (highlighted with a yellow rectangle) is disrupted so that it cannot take a reasonable action. This case illustrates that the random crashes of some agents will undermine the coordination among the rest of the agents in the team, which is likely to cause a drop in the win rate. However, it can be observed in Fig. 3b that agents trained with our method can avoid such effects of an unexpected crash because they may be familiar with abnormal observations.

Figs. 3c and 3d describe another situation where Stalkers (ours) play against Zealots and Stalkers (opponents) in the map 8s_vs_3s5z. Stalkers are good at long-range attacks, while Zealots are skilled in short-range attacks, and Stalkers move faster than Zealots. Stalkers can win the game by simply attacking when the number of normal agents is sufficient, and they will fail if they use the same strategy in crashed scenarios. Fig. 3c illustrates that the Stalkers trained by QMIX only learn to attack continuously because this simple policy can achieve good performance in normal scenarios. However, if they can be split into two groups, i.e., some of them attract Zealots and do kitting (i.e., attack and step back) repeatedly while others focus fire to eliminate Stalkers and then attack the remaining enemies together, they are likely to achieve better performance (Fig. 3d). This case indicates that once a simple winning strategy exists, the learning algorithm has little incentive to explore other optimal strategies, leading to poor capability in the event of crashed agents. The observation implies that increasing the challenge during training may drive the agents to learn better policies.

5.2.3 Performance evaluation and discussion

We evaluated the performance of the compared methods by testing the win rate with different crash

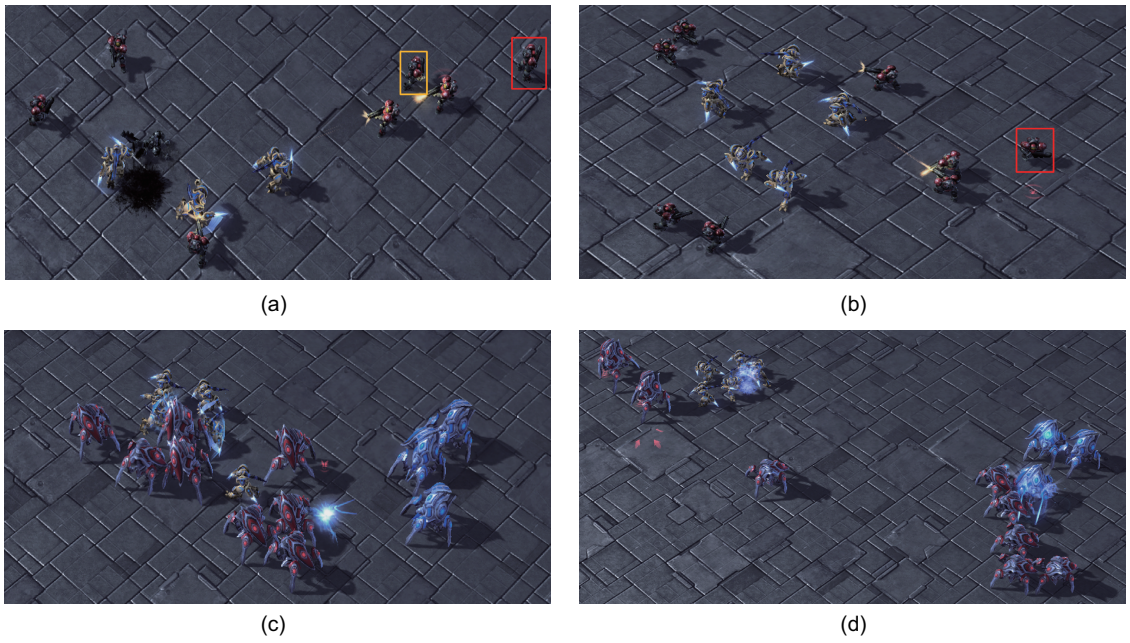

Fig. 3 Illustration of the actions taken by agents who are trained using the original QMIX (on the left) and our adaptive approach (on the right) when they are tested in scenarios with crashes. In (a) and (b), the agent highlighted with a red rectangle represents the crashed one and the agent highlighted with a yellow rectangle is the one that is affected. In (c) and (d), the system shows different behavior patterns after being trained with different approaches. References to color refer to the online version of this figure

rates and the results are shown in Table 1. It can be observed that in standard maps, even using a simple fixed crash rate strategy can help improve performance. In contrast, in our designed maps, this approach works badly when the crash rate is low. We assume this occurs because the maps we designed are relatively simple so that even the original MARL algorithms can handle the scenarios with a low crash rate. In this case, fixing a low crash rate may instead introduce noise, which affects the learning process. However, in scenarios with a high crash rate, this method still has a positive effect. The curriculum learning strategy tends to perform well in scenarios with a low crash rate. In summary, these two straight methods can help the system be more robust in the face of an unexpected crash to some degree, but they all have some limits. In contrast, our adaptive approach can help improve the performance in different maps and crash rates, which demonstrates the effectiveness and generalization of our approach.

When compared with the baseline algorithm, our adaptive method tends to gain a greater margin when the crash rate increases, indicating the superiority of our adaptive strategy in dealing with unexpected crashes. This finding further implies the rationality of our adaptive strategy, which allows agents to learn how to handle the crash scenarios step by step. In addition, the performance achieved by our method with re-sampling is consistently superior, compared to the performance achieved without adopting this strategy. We think this can be attributed to the fact that without a re-sampling strategy, there may be samples that contain more crashed agents, thus creating more difficulty during training. This finding also proves the importance of adopting a re-sampling strategy in our coach-assisted framework.

5.2.4 Hyperparameter analysis

In our adaptive framework, the performance threshold β and the learning rate of crash rate ρ, which jointly decide the updating of the adaptive crash rate, are of vital importance to the performance of our method. In this subsection, we further analyze the influence of these two hyperparameters on the overall performance, with other parameters unchanged.

Here, we take the map 3s_vs_5z as an example. Table 2 reports the results of our method under different values of β and ρ. Given the same ρ, a large

Table 1 The performance of the compared methods in terms of the win rate (including mean and standard deviation) under different crash rates

Method	Win rate (%)					
	3s_vs_5z			3s5z_vs_3s5z		
	Crash rate=0.01	0.05	0.10	Crash rate=0.01	0.05	0.10
Baseline	67.0 ± 15.5	61.9 ± 15.7	56.9 ± 15.3	85.3 ± 11.0	64.8 ± 8.3	43.6 ± 11.3
Fix-0.01	84.8 ± 11.8	74.7 ± 14.5	72.0 ± 13.8	86.9 ± 2.1	63.9 ± 1.7	45.8 ± 2.0
Fix-0.05	84.8 ± 7.5	78.0 ± 12.6	72.7 ± 9.6	86.3 ± 3.1	65.8 ± 2.9	46.9 ± 6.2
Fix-0.10	86.9 ± 8.0	81.1 ± 5.4	74.8 ± 8.2	83.6 ± 3.0	64.8 ± 7.0	48.1 ± 4.2
Curriculum	84.4 ± 6.0	81.1 ± 8.1	74.7 ± 5.8	87.8 ± 2.5	66.1 ± 2.9	48.0 ± 1.6
Adaptive-	82.5 ± 8.5	77.8 ± 8.4	71.6 ± 10.3	85.9 ± 4.5	65.2 ± 3.0	46.1 ± 1.9
Adaptive	$\mathbf{88.6 \pm 3.6}$	$\mathbf{83.3 \pm 6.5}$	$\mathbf{79.2 \pm 6.7}$	$\mathbf{88.0 \pm 3.2}$	$\mathbf{67.0 \pm 2.4}$	$\mathbf{51.7 \pm 2.2}$
Method	Win rate (%)					
	8m_vs_5z			8s_vs_3s5z		
	Crash rate=0.01	0.05	0.10	Crash rate=0.01	0.05	0.10
Baseline	94.1 ± 2.3	82.3 ± 4.5	71.6 ± 2.9	88.6 ± 5.7	75.8 ± 7.0	68.6 ± 4.9
Fix-0.01	86.9 ± 5.4	79.8 ± 5.0	65.6 ± 4.4	87.5 ± 5.8	77.3 ± 6.6	62.0 ± 7.3
Fix-0.05	89.1 ± 2.4	84.2 ± 4.4	68.4 ± 6.4	91.1 ± 6.4	80.0 ± 7.2	66.7 ± 7.2
Fix-0.10	90.0 ± 5.0	83.9 ± 7.1	78.0 ± 4.8	88.9 ± 9.4	79.7 ± 11.3	70.0 ± 9.6
Curriculum	94.1 ± 1.8	82.5 ± 2.8	72.3 ± 2.9	92.0 ± 2.9	79.8 ± 5.4	66.6 ± 5.8
Adaptive-	91.3 ± 4.1	84.8 ± 2.1	78.6 ± 3.1	91.7 ± 3.1	80.0 ± 4.6	69.1 ± 6.0
Adaptive	$\mathbf{94.2 \pm 2.2}$	$\mathbf{89.4 \pm 2.4}$	$\mathbf{81.1 \pm 3.1}$	$\mathbf{93.9 \pm 4.5}$	$\mathbf{84.5 \pm 10.0}$	$\mathbf{71.3 \pm 12.2}$

Fix-i represents the variants of QMIX, indicating that the crashed rate is fixed to i during training. Adaptive- represents the results gained by adopting our adaptive method, but without the re-sampling strategy

β means that agents must learn quite well under the current crash rate before exploring a more difficult scenario. We can see that given $\rho = 0.003$, the overall win rate first increases and then decreases as β increases from 0.60 to 0.75, and the best performance is achieved when $\beta = 0.65$. Given the same β, we can see that the performance first increases and then degrades as ρ increases. The reason may be that, if ρ is too small, the crash rate α will be adjusted too slowly, so the agents cannot learn well within a limited number of steps. If ρ is too large, sharply increasing the crash rate may be too difficult for agents to learn coordination and the adjustment of the difficulty will be rough. In summary, the hyperparameters indeed have some effect on our framework, but our method can achieve a relatively stable performance if the hyperparameters are varied in a small range, which proves the robustness of our method.

6 Conclusions

Considering a common phenomenon that some agents may unexpectedly crash in real-world scenarios, this work is dedicated to a coach-assisted MARL framework that can close this sim-to-real gap. Our method simulates different random crash rates during the training process with the help of a coach, so that agents can master the skills necessary to deal with crashes. We conducted experiments on grid-world and StarCraft II micromanagement tasks to show the necessity of considering crashes during operation and tested the effectiveness of our framework using three coaching strategies in scenarios with unexpected crashes. The results demonstrated the efficacy and generalization of our method under different crash rates. In the future, we will further investigate the case in which crashed agents may take other abnormal actions in addition to random actions and other more efficient coaching strategies.

Contributors

Jian ZHAO designed the research and Weixun WANG gave advice. Youpeng ZHAO and Mingyu YANG conducted the experiments. Jian ZHAO and Youpeng ZHAO drafted the paper. Xunhan HU helped prepare figures. Wengang ZHOU, Jianye HAO, and Houqiang LI revised and finalized the paper.

Compliance with ethics guidelines

Jian ZHAO, Youpeng ZHAO, Weixun WANG, Mingyu YANG, Xunhan HU, Wengang ZHOU, Jianye HAO, and Houqiang LI declare that they have no conflict of interest.

Table 2 The impact of performance threshold β and learning rate ρ

β	Win rate at $\rho = 0.003$ (%)			ρ	Win rate at $\beta = 0.65$ (%)		
	Crash rate=0.01	0.05	0.10		Crash rate=0.01	0.05	0.10
0.60	84.7 ± 9.0	80.5 ± 9.6	74.1 ± 8.3	0.001	80.9 ± 9.4	75.6 ± 6.9	65.0 ± 13.0
0.65	88.6 ± 3.6	83.3 ± 6.5	79.2 ± 6.7	0.003	88.6 ± 3.6	83.3 ± 6.5	79.2 ± 6.7
0.70	81.4 ± 7.8	77.7 ± 6.2	72.3 ± 7.1	0.005	88.4 ± 4.5	83.6 ± 4.7	75.5 ± 8.6
0.75	79.5 ± 10.2	77.8 ± 8.5	67.0 ± 8.2	0.015	78.4 ± 7.8	73.4 ± 8.5	68.9 ± 6.0

The results show the win rate (including mean and standard deviation) under different settings across five different random seeds. The experiment takes QMIX as the base model on the 3s_vs_5z task after two million training steps

References

Arndt K, Hazara M, Ghadirzadeh A, et al., 2020. Meta reinforcement learning for sim-to-real domain adaptation. Proc IEEE Int Conf on Robotics and Automation, p.2725-2731. https://doi.org/10.1109/ICRA40945.2020.9196540

Busoniu L, Babuska R, de Schutter B, 2008. A comprehensive survey of multiagent reinforcement learning. *IEEE Trans Syst Man Cybern Part C Appl Rev*, 38(2):156-172. https://doi.org/10.1109/TSMCC.2007.913919

Dosovitskiy A, Ros G, Codevilla F, et al., 2017. Carla: an open urban driving simulator. Proc 1st Conf on Robot Learning, p.1-16.

Foerster J, Nardelli N, Farquhar G, et al., 2017. Stabilising experience replay for deep multi-agent reinforcement learning. Proc 34th Int Conf on Machine Learning, p.1146-1155. https://doi.org/10.5555/3305381.3305500

Furrer F, Burri M, Achtelik M, et al., 2016. RotorS—a modular Gazebo MAV simulator framework. In: Koubaa A (Ed.), Robot Operating System (ROS): the Complete Reference. Volume 1, Springer, Cham, p.595-625. https://doi.org/10.1007/978-3-319-26054-9_23

Guestrin C, Koller D, Parr R, 2001. Multiagent planning with factored MDPs. Proc 14th Int Conf on Neural Information Processing Systems: Natural and Synthetic, p.1523-1530. https://doi.org/10.5555/2980539.2980737

Higgins I, Pal A, Rusu A, et al., 2017. DARLA: improving zero-shot transfer in reinforcement learning. Proc 34th Int Conf on Machine Learning, p.1480-1490.

Kim D, Moon S, Hostallero D, et al., 2019. Learning to schedule communication in multi-agent reinforcement learning. Proc 7th Int Conf on Learning Representations, p.1-17.

Kok JR, Vlassis N, 2006. Collaborative multiagent reinforcement learning by payoff propagation. *J Mach Learn Res*, 7:1789-1828. https://doi.org/10.5555/1248547.1248612

Kraemer L, Banerjee B, 2016. Multi-agent reinforcement learning as a rehearsal for decentralized planning. *Neurocomputing*, 190:82-94. https://doi.org/10.1016/j.neucom.2016.01.031

Lowe R, Wu Y, Tamar A, et al., 2017. Multi-agent actor-critic for mixed cooperative-competitive environments. Proc 31st Int Conf on Neural Information Processing Systems, p.6382-6393. https://doi.org/10.5555/3295222.3295385

McCord C, Queralta JP, Gia TN, et al., 2019. Distributed progressive formation control for multi-agent systems: 2D and 3D deployment of UAVs in ROS/Gazebo with rotors. Proc European Conf on Mobile Robots, p.1-6. https://doi.org/10.1109/ECMR.2019.8870934

Mnih V, Kavukcuoglu K, Silver D, et al., 2015. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529-533. https://doi.org/10.1038/nature14236

Narvekar S, Peng B, Leonetti M, et al., 2020. Curriculum learning for reinforcement learning domains: a framework and survey. *J Mach Learn Res*, 21(181):1-50.

Oliehoek FA, Spaan MTJ, Vlassis N, 2008. Optimal and approximate Q-value functions for decentralized POMDPs. *J Artif Intell Res*, 32:289-353.

Omidshafiei S, Pazis J, Amato C, et al., 2017. Deep decentralized multi-task multi-agent reinforcement learning under partial observability. Proc 34th Int Conf on Machine Learning, p.2681-2690.

Peng P, Wen Y, Yang YD, et al., 2017. Multiagent bidirectionally-coordinated nets: emergence of human-level coordination in learning to play starcraft combat games. Proc 34th Int Conf on Machine Learning, p.2681-2690.

Rashid T, Samvelyan M, de Witt SC, et al., 2018. QMIX: monotonic value function factorisation for deep multi-agent reinforcement learning. Proc 35th Int Conf on Machine Learning, p.4292-4301.

Samvelyan M, Rashid T, de Witt CS, et al., 2019. The StarCraft Multi-agent Challenge. Proc 18th Int Conf on Autonomous Agents and Multiagent Systems, p.2186-2188. https://doi.org/10.5555/3306127.3332052

Shah S, Dey D, Lovett C, et al., 2018. AirSim: high-fidelity visual and physical simulation for autonomous vehicles. 11th Int Conf on Field and Service Robotics, p.621-635. https://doi.org/10.1007/978-3-319-67361-5_40

Son K, Kim D, Kang WJ, et al., 2019. QTRAN: learning to factorize with transformation for cooperative multi-agent reinforcement learning. Proc 36th Int Conf on Machine Learning, p.5887-5896.

Sukhbaatar S, Szlam A, Fergus R, 2016. Learning multi-agent communication with backpropagation. Proc 30th Int Conf on Neural Information Processing Systems, p.2252-2260. https://doi.org/10.5555/3157096.3157348

Sunehag P, Lever G, Gruslys A, et al., 2018. Value-decomposition networks for cooperative multi-agent learning based on team reward. Proc 17th Int Conf on Autonomous Agents and Multiagent Systems, p.2085-2087. https://doi.org/10.5555/3237383.3238080

Tan M, 1993. Multi-agent Reinforcement Learning: Independent vs. Cooperative Agents. Morgan Kaufmann, p.330-337. https://doi.org/10.1016/B978-1-55860-307-3.50049-6

Tobin J, Fong R, Ray A, et al., 2017. Domain randomization for transferring deep neural networks from simulation to the real world. Proc IEEE/RSJ Int Conf on Intelligent Robots and Systems, p.23-30. https://doi.org/10.1109/IROS.2017.8202133

Todorov E, Erez T, Tassa Y, 2012. MuJoCo: a physics engine for model-based control. Proc IEEE/RSJ Int Conf on Intelligent Robots and Systems, p.5026-5033. https://doi.org/10.1109/IROS.2012.6386109

Traoré R, Caselles-Dupré H, Lesort T, et al., 2019. Continual reinforcement learning deployed in real-life using policy distillation and Sim2Real transfer. https://arxiv.org/abs/1906.04452

Tuyls K, Weiss G, 2012. Multiagent learning: basics, challenges, and prospects. *AI Mag*, 33(3):41. https://doi.org/10.1609/aimag.v33i3.2426

Wang JH, Ren ZZ, Liu T, et al., 2020. QPLEX: duplex dueling multi-agent Q-learning. Proc 9th Int Conf on Learning Representations, p.1-16.

Wang YP, Zheng KX, Tian DX, et al., 2020. Cooperative channel assignment for VANETs based on multiagent reinforcement learning. *Front Inform Technol Electron Eng*, 21(7):1047-1058. https://doi.org/10.1631/FITEE.1900308

Wang YP, Zheng KX, Tian DX, et al., 2021. Pre-training with asynchronous supervised learning for reinforcement learning based autonomous driving. *Front Inform Technol Electron Eng*, 22(5):673-686. https://doi.org/10.1631/FITEE.1900637

Yang YD, Hao JY, Liao B, et al., 2020. QATTEN: a general framework for cooperative multiagent reinforcement learning. https://arxiv.org/abs/2002.03939

Zhang KQ, Yang ZR, Basar T, 2021. Decentralized multiagent reinforcement learning with networked agents: recent advances. *Front Inform Technol Electron Eng*, 22(6):802-814. https://doi.org/10.1631/FITEE.1900661

Zhao WS, Queralta JP, Westerlund T, 2020. Sim-to-real transfer in deep reinforcement learning for robotics: a survey. Proc IEEE Symp Series on Computational Intelligence, p.737-744. https://doi.org/10.1109/SSCI47803.2020.9308468