ECAI 2023
K. Gal et al. (Eds.)

doi:10.3233/FAIA230380

Reinforcement Learning with Reward Machines in Stochastic Games

Jueming Hu[a], Jean-Raphaël Gaglione[b], Yanze Wang[a], Zhe Xu[a;*], Ufuk Topcu[b] and Yongming Liu[a]

[a]Arizona State University, Tempe, AZ
[b]University of Texas at Austin, Austin, TX

Abstract. We investigate multi-agent reinforcement learning for stochastic games with complex tasks, where the reward functions are non-Markovian. We utilize reward machines to incorporate high-level knowledge of complex tasks. We develop an algorithm called *Q-learning with reward machines for stochastic games (QRM-SG)*, to learn the *best-response strategy* at Nash equilibrium for each agent. In QRM-SG, we define the *Q-function at a Nash equilibrium* in *augmented state space*. The augmented state space integrates the state of the stochastic game and the state of reward machines. Each agent learns the Q-functions of all agents in the system. We prove that Q-functions learned in QRM-SG converge to the Q-functions at a Nash equilibrium if the stage game at each time step during learning has a global optimum point or a saddle point, and the agents update Q-functions based on the best-response strategy at this point. We use the Lemke-Howson method to derive the best-response strategy given current Q-functions. The three case studies show that QRM-SG can learn the best-response strategies effectively. QRM-SG learns the best-response strategies after around 7500 episodes in Case Study I, 1000 episodes in Case Study II, and 1500 episodes in Case Study III, while baseline methods such as Nash Q-learning and MADDPG fail to converge to the Nash equilibrium in all three case studies.

1 Introduction

In games, multiple agents interact with each other and the behavior of each agent can affect the performance of other agents. *Multi-agent reinforcement learning* (MARL) [1] is a framework for multiple agents to optimize their strategies by repeatedly interacting with the environment. The interactions between agents in MARL problems can be modeled as a *stochastic game* (SG), where Nash equilibrium can be a solution concept. In a Nash equilibrium, agents have *best-response* strategies considering other agents' actions.

In many complex games, the reward functions are non-Markovian (i.e., the reward of each agent can depend on the history of events). One way of encoding a non-Markovian reward function is by using a type of Mealy machine named *reward machines* [11]. In this paper, we focus on stochastic games where each agent intends to complete a complex task which can be represented by a reward machine capturing the temporal structure of the reward function.

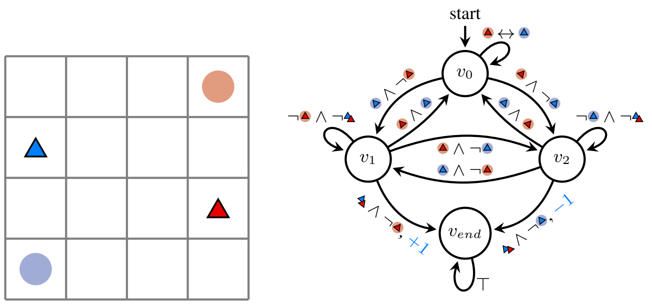
We introduce a variant of the PAC-MAN game as a motivational example, where two agents try to capture each other, and the task completion is defined based on complex conditions, as illustrated in Figure 1. Two agents 🔷 and 🔺 evolve in a grid-world by taking synchronous steps towards adjacent cells. We also introduce two power bases 🔵 and 🟠 at fixed locations. We track whether the two agents encounter logical propositions such as 🔵🔷, 🟠🔺, and 🔷🔺 or not to evaluate the lower-level dynamics and specify the high-level reward. For instance, the proposition 🔵🔷 is true when agent 🔷 is at power base 🔵. Similarly, the proposition 🔷🔺 is true when agents 🔷 and 🔺 meet on the same cell. In the simplest variant of the game, the goal of each agent is to first reach its own power base and then capture the other agent. Each agent obtains a reward of 1 when its task is completed. Whenever agent 🔷 arrives at its power base 🔵, agent 🔷 becomes more powerful than agent 🔺. However, if agent 🔺 then arrives at its power base 🟠, agent 🔺 becomes the more powerful agent. The more powerful agent is capable to capture the other agent.

We address the challenge of learning complex tasks in two-agent general-sum stochastic games with non-Markovian reward functions. We develop an algorithm called *Q-learning with reward machines for stochastic games (QRM-SG)*, where we use reward machines to specify the tasks and expose the structure of reward functions. The proposed approach defines the *Q-function at a Nash equilibrium* in *augmented state space* to adapt Q-learning to the setting of stochastic games when the tasks are specified by reward machines. The augmented state space integrates the state of the stochastic game and the state of reward machines. During learning, we formulate a stage game at each time step based on the current Q-functions in augmented state space and use the Lemke-Howson method [13] to derive a Nash equilibrium. Q-functions in augmented state are then updated according to the Nash equilibrium. QRM-SG enables the learning of the best-response strategy at a Nash equilibrium for each agent. Q-functions learned in QRM-SG converge to the Q-functions at a Nash equilibrium if the stage game at each time step during learning has a global optimum point or a saddle point, and the agents update Q-functions based on the best-response strategy at this point. We test QRM-SG in three case studies and compare QRM-SG with common baselines such as Nash Q learning [10], Nash Q-learning in augmented state space, multi-agent deep deterministic policy gradient (MADDPG) [16], and MADDPG in augmented state space. The results show that QRM-SG learns the best-response strategies at a Nash equilibrium effectively.

2 Related Work

Multi-agent reinforcement learning with stochastic games: Our work is closely related to multi-agent reinforcement learning in stochastic games. Many recent works of multi-agent reinforcement learning (MARL) focus on a cooperative setting [5–7,22–24,28,33], where all the agents share a common reward function. [6] addresses the nonstationarity issue introduced by independent Q-learning in

* Corresponding Author. Email: xzhe1@asu.edu.



(a) Map of the environment. Agents ▲ and ▲ can move synchronously to an adjacent cell (or stay put) at each step. Locations ● and ● are called power bases.

(b) Reward machine for agent ▲ with sparse reward (zero when not specified on a transition). We suppose that ▲, ▲ and ▲ cannot all be true at the same time, making this reward machine deterministic. ⊤ represents tautology.

Figure 1: The simplest variant of the PAC-MAN game is a symmetric zero-sum game where two agents try to capture each other. The agent that visited its own power base the most recently can capture the other agent. The low-level stochastic game plays out in a grid world (a) and the high-level reward is captured by (b).

multi-agent reinforcement learning by using the importance sampling technique since the multiple agents usually learn concurrently. With the multi-agent actor-critic method as a backbone, [5] proposes counterfactual multi-agent (COMA) policy gradients that use a centralized critic to estimate the Q-function and decentralized actors to optimize the agents' policies. Similar to both of these approaches that describe the multi-agent task in the stochastic game, but we tackle a non-cooperative multi-agent reinforcement learning problem in a stochastic game compared to [15, 18, 34, 35]. As opposed to [15, 35], which extends inverse reinforcement learning (IRL) to the non-cooperative stochastic game to let the agent learn the reward function from the observation of other agent/expert's behavior, we assume the reward function is given to all agents that participate in the stochastic game. However, our work shares the same objective as the works mentioned above, as each agent learns a policy at a Nash equilibrium in the stochastic game. Furthermore, the existing works rely on the Markovian property of the reward function. The definition of the stochastic game in this paper differs from the aforementioned works as we specify the reward function as non-Markovian, which is more applicable to many real-world applications.

Reinforcement learning with formal methods: This paper is also closely related to the work of using the formal methods in reinforcement learning (RL) with a non-Markovian reward function for single agent [3, 4, 8, 21, 25, 26, 30–32], and multi-agent [14, 19]. In the single-agent RL, [8] proposes DeepSynth which is a new algorithm that synthesizes deterministic finite automation to infer the unknown non-Markovian rewards of achieving a sequence of high-level objectives. However, DeepSynth is based on the general Markov decision process, which makes it incapable of solving MARL problems that are modeled as a stochastic game. In a similar case for multi-agent reinforcement learning, [19] tackles the MARL problem in a stochastic game with non-Markovian reward expressed in temporal logic. [14] incorporate the non-Markovian reward to MARL in an unknown environment, with the complex task specification also described in temporal logic. In this work, the reward function for all the agents in a stochastic game is encoded by the reward machine, which provides an automata-based representation that enables an agent to decompose an RL problem into structured subproblems that can be efficiently learned via off-policy learning [11].

3 Preliminaries

In this section, we introduce the necessary background on stochastic games, reward machines, and how to connect stochastic games and reward machines.

3.1 Stochastic Games

Definition 1 *A two-player general-sum stochastic game (SG) is a tuple $\mathcal{G} = (S, s_I, A_{\mathfrak{e}}, A_{\mathfrak{a}}, p, R_{\mathfrak{e}}, R_{\mathfrak{a}}, \gamma)$, where S is the finite state space (consisting of states $s = [s_{\mathfrak{e}}; s_{\mathfrak{a}}]$, $s_{\mathfrak{e}}$ and $s_{\mathfrak{a}}$ are the substates of the ego agent and the adversarial agent, respectively), $s_I \in S$ is the initial state, $A_{\mathfrak{e}}$ and $A_{\mathfrak{a}}$ are the finite set of actions for the ego agent and the adversarial agent, respectively, and $p\colon S \times A_{\mathfrak{e}} \times A_{\mathfrak{a}} \times S \to [0, 1]$ is the probabilistic transition function. Reward functions $R_{\mathfrak{e}} : (S \times A_{\mathfrak{e}} \times A_{\mathfrak{a}})^+ \times S \to \mathbb{R}$ and $R_{\mathfrak{a}} : (S \times A_{\mathfrak{e}} \times A_{\mathfrak{a}})^+ \times S \to \mathbb{R}$ specify the non-Markovian rewards to the ego and adversarial agents, respectively. $\gamma \in (0, 1]$ is the discount factor.*

We use the subscripts $\mathfrak{e}$ and $\mathfrak{a}$ to represent the ego agent and the adversarial agent, respectively. Our definition of the SG differs from the "usual" definition used in reinforcement learning (e.g., [29]) in that the reward functions $R_{\mathfrak{e}}$ and $R_{\mathfrak{a}}$ are defined over the whole history (i.e., the rewards are *non-Markovian*). A *trajectory* is a sequence of states and actions $s_0 a_{\mathfrak{e},0} a_{\mathfrak{a},0} s_1 \ldots s_k a_{\mathfrak{e},k} a_{\mathfrak{a},k} s_{k+1}$, with $s_0 = s_I$. Its corresponding *reward sequence* for agent $\mathfrak{i}$ ($\mathfrak{i} \in \{\mathfrak{e}, \mathfrak{a}\}$) is $r_{\mathfrak{i},1} \ldots r_{\mathfrak{i},k}$, where $r_{\mathfrak{i},t} = R_{\mathfrak{i}}(s_0 a_{\mathfrak{e},0} a_{\mathfrak{a},0} \ldots s_t a_{\mathfrak{e},t} a_{\mathfrak{a},t} s_{t+1})$, for each $t \leq k$. For both the ego agent and the adversarial agent, they observe the same trajectory $s_0 a_{\mathfrak{e},0} a_{\mathfrak{a},0} \ldots s_k a_{\mathfrak{e},k} a_{\mathfrak{a},k} s_{k+1}$. The ego agent achieves a discounted cumulative reward $\sum_{t=0}^{k} \gamma^t r_{\mathfrak{e},t}$ for the trajectory $s_0 a_{\mathfrak{e},0} a_{\mathfrak{a},0} \ldots s_k a_{\mathfrak{e},k} a_{\mathfrak{a},k} s_{k+1}$. Similarly, the adversarial agent achieves a discounted cumulative reward $\sum_{t=0}^{k} \gamma^t r_{\mathfrak{a},t}$ for the trajectory $s_0 a_{\mathfrak{e},0} a_{\mathfrak{a},0} \ldots s_k a_{\mathfrak{e},k} a_{\mathfrak{a},k} s_{k+1}$.

In an SG, we consider each agent to be equipped with a Markovian strategy $\pi_{\mathfrak{i}} : S \times A_{\mathfrak{i}} \to [0, 1], \mathfrak{i} \in \{\mathfrak{e}, \mathfrak{a}\}$, mapping the state to the probability of selecting each possible action. The objective of the ego agent and the adversarial agent is to maximize the expected discounted cumulative reward $\tilde{v}_{\mathfrak{e}}$ and $\tilde{v}_{\mathfrak{a}}$, respectively.

$$\tilde{v}_{\mathfrak{i}}(s, \pi_{\mathfrak{e}}, \pi_{\mathfrak{a}}) = \sum_{t=0}^{\infty} \gamma^t \mathbb{E}(r_{\mathfrak{i},t} | \pi_{\mathfrak{e}}, \pi_{\mathfrak{a}}, s_0 = s) \tag{1}$$

As a solution concept of an SG, a Nash equilibrium [20] is a collection of strategies for each of the agents such that each agent cannot improve its own reward by changing its own strategy, or we say that, each agent's strategy is a *best-response* to the other agents' strategies.

Definition 2 *For an SG $\mathcal{G} = (S, s_I, A_{\mathfrak{e}}, A_{\mathfrak{a}}, R_{\mathfrak{e}}, R_{\mathfrak{a}}, p, \gamma)$, the strategies of the ego agent and the adversarial agent $\pi_{\mathfrak{e}}^*$ and $\pi_{\mathfrak{a}}^*$ are at a Nash equilibrium of the SG if*

$$\tilde{v}_{\mathfrak{e}}(s, \pi_{\mathfrak{e}}^*, \pi_{\mathfrak{a}}^*) \geq \tilde{v}_{\mathfrak{e}}(s, \pi_{\mathfrak{e}}, \pi_{\mathfrak{a}}^*)$$
$$\tilde{v}_{\mathfrak{a}}(s, \pi_{\mathfrak{e}}^*, \pi_{\mathfrak{a}}^*) \geq \tilde{v}_{\mathfrak{a}}(s, \pi_{\mathfrak{e}}^*, \pi_{\mathfrak{a}})$$

hold for any $\pi_{\mathfrak{e}}$ and $\pi_{\mathfrak{a}}$.

A Nash equilibrium illustrates the stable point where each agent reacts optimally to the behavior of other agents. In a Nash equilibrium, given other agents' strategies, the learning agent is not able to get a higher reward by unilaterally deviating from its current strategy. We refer to $\pi_{\mathfrak{i}}^*$ as the strategy of agent $\mathfrak{i}$ that constitutes a Nash equilibrium. The goal of each learning agent is to find a strategy that maximizes its discounted cumulative reward considering the other agent's action once it reaches the game's Nash equilibrium.

3.2 Reward Machines

In this paper, we use reward machines to express the task specification. Reward machines [2, 12] encode a (non-Markovian) reward in a type of finite-state machine. [1] Technically, a reward machine is a special instance of a Mealy machine [27], the one that takes subsets of propositional variables as its input and outputs real numbers as reward values.

Definition 3 *A* reward machine *(RM) $\mathcal{A} = (V, v_I, 2^{\mathcal{P}}, \mathbb{R}, \delta, \sigma)$ consists of a finite, nonempty set V of RM states, an initial RM state $v_I \in V$, an input alphabet $2^{\mathcal{P}}$ where $\mathcal{P}$ is a finite set of propositional variables, an output alphabet $\mathbb{R}$, a (deterministic) transition function $\delta \colon V \times 2^{\mathcal{P}} \to V$, and an output function $\sigma \colon V \times 2^{\mathcal{P}} \to \mathbb{R}$.*

By using reward machines, we decompose a complex task into several stages. For example, considering the reward machine of the ego agent (▲) in the motivational example (Figure 1b), the initial RM state is v_0. At v_0, if the proposition 🔵 becomes true and 🔴 is not reached, the machine transitions to v_1 and outputs a reward of 0, or if the proposition 🔴 becomes true and 🔵 is not reached, the machine transitions to v_2 and outputs a reward of 0. At v_1, after ▲ is reached and the adversarial agent is not at its power base ($\neg$🔴), the ego agent completes its task, and the machine transitions to v_{end} and outputs a reward of 1. Similarly, at v_2, after ▲ is reached and $\neg$🔵 is true, the machine transitions to v_{end} and outputs a reward of -1. Additionally, as any agent can be the more powerful of the two (i.e. with the power to capture the other agent) once it arrives at its own power base, the machine can transition between v_1 and v_2. We also have self-loops if the propositions do not lead to a change in the stage of the task.

3.3 Connecting Stochastic Games and Reward Machines

To build a bridge between the reward machine and the SG, a labeling function $L : S \times A_{\mathfrak{e}} \times A_{\mathfrak{a}} \times S \to 2^{\mathcal{P}}$ is required to map the states in an SG to the *high-level events*. High-level events represent expert knowledge of what is relevant for successfully executing a task and are assumed to be available to both the ego and adversarial agents. At time step t, we have $l_t = L(s_t, a_{\mathfrak{e},t}, a_{\mathfrak{a},t}, s_{t+1})$ to determine the set of relevant high-level events that the agents detect in the environment. l_t is a set consisting of the propositions in $\mathcal{P}$ that are true given $(s_t, a_{\mathfrak{e},t}, a_{\mathfrak{a},t}, s_{t+1})$. We assume that the agents share the same set of propositional variables $\mathcal{P}$, and the agents observe the same sequences of high-level events for the same trajectory. Specifically, for the trajectory $s_0 a_{\mathfrak{e},0} a_{\mathfrak{a},0} \ldots s_k a_{\mathfrak{e},k} a_{\mathfrak{a},k} s_{k+1}$, its corresponding *event sequence* is $l_0 l_1 \ldots l_k$. The reward machine $\mathcal{A}$ receives l_t as an input and outputs a reward $r_t = \sigma(v_t, l_t)$. To connect the input and output of $\mathcal{A}$, we write $\mathcal{A}(l_0 l_1 \cdots l_k) = r_0 r_1 \cdots r_k$. We call $(l_0 l_1 \ldots l_k, r_0 r_1 \cdots r_k)$ a trace and we consider finite traces (with possibly unbounded length) in this paper as a reward machine maps any sequence of events to a sequence of rewards. Given the detected high-level event l_t at time step t, the RM state transitions to $v_{t+1} = \delta(v_t, l_t)$.

Definition 4 *A reward machine $\mathcal{A}_{\mathfrak{i}} = (V_{\mathfrak{i}}, v_{I,\mathfrak{i}}, 2^{\mathcal{P}}, \mathbb{R}_{\mathfrak{i}}, \delta_{\mathfrak{i}}, \sigma_{\mathfrak{i}})$ ($\mathfrak{i} \in \{\mathfrak{e}, \mathfrak{a}\}$) encodes the non-Markovian reward function $R_{\mathfrak{i}}$ of the agent $\mathfrak{i}$ in the SG $\mathcal{G} = (S, s_I, A_{\mathfrak{e}}, A_{\mathfrak{a}}, R_{\mathfrak{e}}, R_{\mathfrak{a}}, p, \gamma)$, if for every trajectory $s_0 a_{\mathfrak{e},0} a_{\mathfrak{a},0} \ldots s_k a_{\mathfrak{e},k} a_{\mathfrak{a},k} s_{k+1}$ and the corresponding event sequence $l_0 l_1 \cdots l_k$, the reward sequence $r_{\mathfrak{i},0} \cdots r_{\mathfrak{i},k}$ of the agent $\mathfrak{i}$ equals $\mathcal{A}_{\mathfrak{i}}(l_0 l_1 \cdots l_k)$.*

[1] The reward machines we are utilizing are the so-called *simple reward machines* in the parlance of [12], where every output symbol is a real number.

We use reward machines $\mathcal{A}_{\mathfrak{e}}$ and $\mathcal{A}_{\mathfrak{a}}$ to encode the non-Markovian reward functions $R_{\mathfrak{e}}$ and $R_{\mathfrak{a}}$ in the SG $\mathcal{G}$ for the ego agent and the adversarial agent, respectively.

4 Problem formulation

We focus on a two-agent general-sum stochastic game (SG) with non-Markovian reward functions, $\mathcal{G} = (S, s_I, A_{\mathfrak{e}}, A_{\mathfrak{a}}, R_{\mathfrak{e}}, R_{\mathfrak{a}}, p, \gamma)$. Each agent is given a task to complete, and the task completion depends on the other agent's behavior. We use a separate reward machine $\mathcal{A}_{\mathfrak{i}} = (V_{\mathfrak{i}}, v_{I,\mathfrak{i}}, 2^{\mathcal{P}}, \mathbb{R}, \delta_{\mathfrak{i}}, \sigma_{\mathfrak{i}})$ for agent $\mathfrak{i}$ ($\mathfrak{i} \in \{\mathfrak{e}, \mathfrak{a}\}$) to specify a task that the agent intends to achieve. Moreover, $\mathcal{A}_{\mathfrak{i}}$ encodes the non-Markovian reward function $R_{\mathfrak{i}}$ in $\mathcal{G}$. An agent obtains a high discounted cumulative reward if its task is completed. Therefore, seeking to accomplish the task is consistent with maximizing the discounted cumulative reward for the learning agent.

In the two-player general-sum stochastic game, the agents operate in an adversarial environment, e.g., in the motivational example, the task of the ego agent is accomplished if the ego agent captures the adversarial agent. Similarly, the task of the adversarial agent is accomplished if the adversarial agent captures the ego agent.

In this paper, we aim to find the best-response strategy for each agent in the two-player general-sum stochastic game with reward functions encoded by reward machines. Agents try to maximize their own discounted cumulative reward. We have the following problem formulation.

Problem 1 *Given an SG $\mathcal{G} = (S, s_I, A_{\mathfrak{e}}, A_{\mathfrak{a}}, R_{\mathfrak{e}}, R_{\mathfrak{a}}, p, \gamma)$, where the non-Markovian reward functions $R_{\mathfrak{e}}$ and $R_{\mathfrak{a}}$ are encoded by reward machines $\mathcal{A}_{\mathfrak{e}} = (V_{\mathfrak{e}}, v_{I,\mathfrak{e}}, 2^{\mathcal{P}}, \mathbb{R}_{\mathfrak{e}}, \delta_{\mathfrak{e}}, \sigma_{\mathfrak{e}})$ and $\mathcal{A}_{\mathfrak{a}} = (V_{\mathfrak{a}}, v_{I,\mathfrak{a}}, 2^{\mathcal{P}}, \mathbb{R}_{\mathfrak{a}}, \delta_{\mathfrak{a}}, \sigma_{\mathfrak{a}})$, respectively, learn the strategies of the ego agent and the adversarial agent $\pi^*_{\mathfrak{e}}$ and $\pi^*_{\mathfrak{a}}$ at a Nash equilibrium of the stochastic game.*

We assume that the state, RM state, selected action, and earned reward of both agents are observable to each agent. The set of propositional variables $\mathcal{P}$ is the same in $\mathcal{A}_{\mathfrak{e}}$ and $\mathcal{A}_{\mathfrak{a}}$. The high-level events received in the reward machines are a function of both agents' states and actions. Therefore, the learning processes of the two agents are coupled.

5 Q-learning with Reward Machine for Stochastic Game (QRM-SG)

In this section, we introduce the proposed QRM-SG algorithm for stochastic games with reward functions encoded by reward machines. We first define a stochastic game with reward machines and formulate it as a product stochastic game to obtain Markovian rewards. Then, we use QRM-SG to learn a strategy for each agent that constitutes a Nash equilibrium.

Definition 5 *Given a SG $\mathcal{G} = (S, s_I, A_{\mathfrak{e}}, A_{\mathfrak{a}}, R_{\mathfrak{e}}, R_{\mathfrak{a}}, p, \gamma)$, where the reward functions $R_{\mathfrak{e}}$ and $R_{\mathfrak{a}}$ are encoded by reward machines $\mathcal{A}_{\mathfrak{e}} = (V_{\mathfrak{e}}, v_{I,\mathfrak{e}}, 2^{\mathcal{P}}, \mathbb{R}_{\mathfrak{e}}, \delta_{\mathfrak{e}}, \sigma_{\mathfrak{e}})$ and $\mathcal{A}_{\mathfrak{a}} = (V_{\mathfrak{a}}, v_{I,\mathfrak{a}}, 2^{\mathcal{P}}, \mathbb{R}_{\mathfrak{a}}, \delta_{\mathfrak{a}}, \sigma_{\mathfrak{a}})$, respectively, we define a stochastic game with reward machines (SGRM) as a product stochastic game $\mathcal{H} = (S', s'_I, A_{\mathfrak{e}}, A_{\mathfrak{a}}, R'_{\mathfrak{e}}, R'_{\mathfrak{a}}, p', \gamma)$, where*

- $S' = S \times V_{\mathfrak{e}} \times V_{\mathfrak{a}}$,
- $s'_I = s_I \times v_{I,\mathfrak{e}} \times v_{I,\mathfrak{a}}$,
- $p'(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}, s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}) =$
$$\begin{cases} p(s, a_{\mathfrak{e}}, a_{\mathfrak{a}}, s') & \textit{if } v'_{\mathfrak{e}} = \delta_{\mathfrak{e}}(v_{\mathfrak{e}}, L(s, a_{\mathfrak{e}}, a_{\mathfrak{a}}, s')) \\ & \textit{and } v'_{\mathfrak{a}} = \delta_{\mathfrak{a}}(v_{\mathfrak{a}}, L(s, a_{\mathfrak{e}}, a_{\mathfrak{a}}, s')), \\ 0 & \textit{otherwise,} \end{cases}$$

- $R'_{\mathfrak{e}} = \sigma_{\mathfrak{e}}(v_{\mathfrak{e}}, L(s, a_{\mathfrak{e}}, a_{\mathfrak{a}}, s'))$,
- $R'_{\mathfrak{a}} = \sigma_{\mathfrak{a}}(v_{\mathfrak{a}}, L(s, a_{\mathfrak{e}}, a_{\mathfrak{a}}, s'))$.

where L is the labeling function $L : S \times A_{\mathfrak{e}} \times A_{\mathfrak{a}} \times S \to 2^{\mathcal{P}}$.

Lemma 1 *A stochastic game with reward machines (SGRM) has Markovian reward functions to express the non-Markovian reward functions in the stochastic game encoded by reward machines.*

In a stochastic game with reward machines (SGRM) $\mathcal{H} = (S', s'_I, A_{\mathfrak{e}}, A_{\mathfrak{a}}, R'_{\mathfrak{e}}, R'_{\mathfrak{a}}, p', \gamma)$, the rewards received by the agents are Markovian with respect to the augmented state space S', the Cartesian product state set $S \times V_{\mathfrak{e}} \times V_{\mathfrak{a}}$. Also, the rewards are the same as the reward outputs by reward machines $\mathcal{A}_{\mathfrak{e}}$ and $\mathcal{A}_{\mathfrak{a}}$. The agents consider S' to select actions.

At each time step, the agents select actions simultaneously and execute the actions $a_{\mathfrak{e}}$ and $a_{\mathfrak{a}}$ respectively. Then the state s moves to s'. The labeling function L returns the high-level events given current state and actions, and next state. Given the events, reward machines deliver the transition of RM states, $(v_{\mathfrak{e}}, v_{\mathfrak{a}})$ moving to $(v'_{\mathfrak{e}}, v'_{\mathfrak{a}})$, where $v'_{\mathfrak{e}} = \delta_{\mathfrak{e}}(v_{\mathfrak{e}}, L(s, a_{\mathfrak{e}}, a_{\mathfrak{a}}, s'))$, $v'_{\mathfrak{a}} = \delta_{\mathfrak{a}}(v_{\mathfrak{a}}, L(s, a_{\mathfrak{e}}, a_{\mathfrak{a}}, s'))$. The transition $((s, v_{\mathfrak{e}}, v_{\mathfrak{a}})$ to $(s, v'_{\mathfrak{e}}, v'_{\mathfrak{a}}))$ leads to a reward of $\sigma_{\mathfrak{e}}(v_{\mathfrak{e}}, L(s, a_{\mathfrak{e}}, a_{\mathfrak{a}}, s'))$ for the ego agent and $\sigma_{\mathfrak{a}}(v_{\mathfrak{a}}, L(s, a_{\mathfrak{e}}, a_{\mathfrak{a}}, s'))$ for the adversarial agent.

To utilize Q-learning and considering a Nash equilibrium as a solution concept, we consider the optimal Q-function for agent $\mathfrak{i}$ ($\mathfrak{i} \in \{\mathfrak{e}, \mathfrak{a}\}$) in an SGRM is the *Q-function at a Nash equilibrium* inspired by [10], which is the expected discounted cumulative reward obtained by agent $\mathfrak{i}$ when both agents follow a joint Nash equilibrium strategy from the next period on. Therefore, Q-function at a Nash equilibrium depends on both agent's actions. Moreover, we define the Q-function at a Nash equilibrium in augmented state space $S \times V_{\mathfrak{e}} \times V_{\mathfrak{a}}$. Let $q^*_{\mathfrak{i}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}})$ denote the Q-function at a Nash equilibrium for agent $\mathfrak{i}$. Mathematically,

$$q^*_{\mathfrak{i}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}) = r_{\mathfrak{i}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}) + \gamma \sum_{\substack{s' \in S \\ v'_{\mathfrak{e}} \in V_{\mathfrak{e}} \\ v'_{\mathfrak{a}} \in V_{\mathfrak{a}}}} p(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}} | s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}) \tilde{v}_{\mathfrak{i}}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}, \pi^*_{\mathfrak{e}}, \pi^*_{\mathfrak{a}}) \quad (2)$$

where $\tilde{v}_{\mathfrak{i}}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}, \pi^*_{\mathfrak{e}}, \pi^*_{\mathfrak{a}})$ is the expected discounted cumulative reward starting from the augmented state $(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})$ over infinite periods when agents follow the best-response strategies $\pi^*_{\mathfrak{e}}$ and $\pi^*_{\mathfrak{a}}$. When multiple equilibria are derived, different Nash strategy profile may lead to different Q-function at a Nash equilibrium.

The proposed algorithm QRM-SG tries to learn $q^*_{\mathfrak{i}}$ ($\mathfrak{i} \in \{\mathfrak{e}, \mathfrak{a}\}$) for a Nash equilibrium strategy. At a Nash equilibrium, the strategy of one agent is optimal when considering the other agent's behavior. Each agent maintains two Q-functions — learning the Q-functions of both itself and the other agent. $q_{\mathfrak{ij}}$ represents the Q-function for agent $\mathfrak{i}$ and learned by agent $\mathfrak{j}$. For instance, the ego agent is equipped with $q_{\mathfrak{ee}}$ and $q_{\mathfrak{ae}}$. Similarly, $\pi_{\mathfrak{ij}}$ is the strategy of agent $\mathfrak{i}$ learned by agent $\mathfrak{j}$. The agents learn the Q-functions through exploration and by observing state and RM state transitions, actions, and rewards. At each time step during learning, we can formulate a *stage game* for agent $\mathfrak{i}$ given the Q-functions $q_{\mathfrak{ei}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}})$ and $q_{\mathfrak{ai}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}})$ estimated by agent $\mathfrak{i}$.

Definition 6 *A two-player stage game is defined as* $(r_{\mathfrak{e}}, r_{\mathfrak{a}})$, *where* $r_{\mathfrak{i}}$ ($\mathfrak{i} \in \{\mathfrak{e}, \mathfrak{a}\}$) *is agent* $\mathfrak{i}$*'s reward function* $R_{\mathfrak{i}}$ *over the space of joint actions,* $r_{\mathfrak{i}} = \{R_{\mathfrak{i}}(a_{\mathfrak{e}}, a_{\mathfrak{a}}) | a_{\mathfrak{e}} \in A_{\mathfrak{e}}, a_{\mathfrak{a}} \in A_{\mathfrak{a}}\}$.

Given the stage game $(q_{\mathfrak{ei}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}), q_{\mathfrak{ai}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}))$ during learning, we derive the best-response strategies at a Nash equilibrium and Q-functions are updated based on the expectation that agents would take best-response actions.

Algorithm 1: QRM-SG

```
1  Hyperparameter: episode length eplength, $\gamma$, $\epsilon$
2  Input: Reward machines $\mathcal{A}_{\mathfrak{e}}, \mathcal{A}_{\mathfrak{a}}$
3  $s \leftarrow InitialState(); v_{\mathfrak{e}} \leftarrow v_{I,\mathfrak{e}}; v_{\mathfrak{a}} \leftarrow v_{I,\mathfrak{a}}$
4  $q_{\mathfrak{ee}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}), q_{\mathfrak{ae}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}),$
   $q_{\mathfrak{ea}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}), q_{\mathfrak{aa}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}})$
   $\leftarrow InitialQfunctions()$
5  for $episode = 1, 2, \cdots$ do
6      for $0 \leq t < eplength$ do
7          for $\mathfrak{i} \in \{\mathfrak{e}, \mathfrak{a}\}$ do
8              $\bar{\epsilon}$ = GenerateRandomValue()
9              if $\bar{\epsilon} < \epsilon$ then
10                 $a_{\mathfrak{i}} \leftarrow$ ChooseActionRandomly()
11             else
12                 $\pi_{\mathfrak{ei}}(\cdot|s, v_{\mathfrak{e}}, v_{\mathfrak{a}}), \pi_{\mathfrak{ai}}(\cdot|s, v_{\mathfrak{e}}, v_{\mathfrak{a}}) \leftarrow$
                     CalculateNashEquilibrium($q_{\mathfrak{ei}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}})$,
                     $q_{\mathfrak{ai}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}})$)
13                 $a_{\mathfrak{i}} \leftarrow \underset{a \in A_{\mathfrak{i}}}{\text{argmax}}\, \pi_{ii}(a|s, v_{\mathfrak{e}}, v_{\mathfrak{a}})$
14         $s' \leftarrow$ ExecuteAction($s, a_{\mathfrak{e}}, a_{\mathfrak{a}}$)
15         $l_t \leftarrow L(s, a_{\mathfrak{e}}, a_{\mathfrak{a}}, s')$
16         for $\mathfrak{i} \in \{\mathfrak{e}, \mathfrak{a}\}$ do
17             $v'_{\mathfrak{i}} \leftarrow \delta_{\mathfrak{i}}(v_{\mathfrak{i}}, l_t)$
18             $r_{\mathfrak{i},t} \leftarrow \sigma_{\mathfrak{i}}(v_{\mathfrak{i}}, l_t)$
19         for $\mathfrak{i} \in \{\mathfrak{e}, \mathfrak{a}\}$ do
20             $\pi_{\mathfrak{ei}}(\cdot|s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}), \pi_{\mathfrak{ai}}(\cdot|s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}) \leftarrow$
               CalculateNashEquilibrium($q_{\mathfrak{ei}}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})$,
               $q_{\mathfrak{ai}}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})$)
21             $\bar{q}_{\mathfrak{ei}}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}) \leftarrow$
               $\pi_{\mathfrak{ei}}(\cdot|s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})\pi_{\mathfrak{ai}}(\cdot|s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})q_{\mathfrak{ei}}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})$
22             $\bar{q}_{\mathfrak{ai}}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}) \leftarrow$
               $\pi_{\mathfrak{ei}}(\cdot|s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})\pi_{\mathfrak{ai}}(\cdot|s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})q_{\mathfrak{ai}}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})$
23             $q_{\mathfrak{ei}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}) \leftarrow$
               $(1 - \alpha_t)q_{\mathfrak{ei}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}) + \alpha_t(r_{\mathfrak{e},t} +$
               $\gamma\bar{q}_{\mathfrak{ei}}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}))$
24             $q_{\mathfrak{ai}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}) \leftarrow$
               $(1 - \alpha_t)q_{\mathfrak{ai}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}) + \alpha_t(r_{\mathfrak{a},t} +$
               $\gamma\bar{q}_{\mathfrak{ai}}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}))$
25         $s \leftarrow s'; v_{\mathfrak{e}} \leftarrow v'_{\mathfrak{e}}; v_{\mathfrak{a}} \leftarrow v'_{\mathfrak{a}}$
26 return $(q_{\mathfrak{ee}}, q_{\mathfrak{ae}}, q_{\mathfrak{ea}}, q_{\mathfrak{aa}})$
```

Algorithm 1 shows the pseudocode for QRM-SG. QRM-SG begins with initializations of Q-functions (line 4). Within an episode, each agent interacts with the environment (lines 7 to 18), and uses the observations perceived from the environment to update Q-functions (lines 19 to 24). The game restarts when the number of time steps reaches the threshold or at least one agent completes the task.

We have three loops within one episode in the QRM-SG. The first loop (lines 7 to 13) demonstrates how the agents select an action, where $\epsilon -$ greedy policy [29] is adopted to balance the exploration and exploitation. A random floating point number $\bar{\epsilon}$ is uniformly sampled in the range $[0.0, 1.0)$ (line 8). When $\bar{\epsilon} < \epsilon$ which has a probability of ϵ (line 9), the learning agent takes a uniformly random action (line 10). With probability $1 - \epsilon$, the learning agent takes the Nash equilibrium action (lines 12 and 13). In implementation, we use the Lemke-Howson method [13] to derive a Nash equilibrium. In line 12, the Lemke-Howson algorithm takes the learned Q-functions $q_{\mathfrak{e}i}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}})$ and $q_{\mathfrak{a}i}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}})$ with respect to the current augmented state $(s, v_{\mathfrak{e}}, v_{\mathfrak{a}})$ as the input, and returns a Nash equilibrium which specifies probabilities of each available action. Then, the agent selects the action with the maximum probability (line 13). In line 14, agents execute the selected actions and the state s transitions to s' according to the function p in the stochastic game $\mathcal{G}$.

The second loop (lines 16 to 18) shows the transitions in the corresponding reward machine for each agent. The labeling function first detects the high-level event (line 15). Given the current RM state and detected event, for each agent, we track the transition of RM state (line 17) and compute the reward that the agent would receive according to its reward machine (line 18).

The third loop (lines 19 to 24) is the learning loop. For agent $\mathfrak{i}$, a Nash equilibrium at $(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})$ is first computed using the Q-functions learned by agent $\mathfrak{i}$ (line 20). Then in lines 21 and 22, we calculate the discounted cumulative reward $\overline{q}_{\mathfrak{j}\mathfrak{i}}$ of agent $\mathfrak{j}$ ($\mathfrak{j} \in \{\mathfrak{e}, \mathfrak{a}\}$) when the agents follow the Nash equilibrium obtained in line 20 at $(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})$. $\overline{q}_{\mathfrak{j}\mathfrak{i}}$ uses the corresponding Q-functions $q_{\mathfrak{j}\mathfrak{i}}$ to derive the discounted cumulative reward at $(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})$.

$$\overline{q}_{\mathfrak{j}\mathfrak{i}}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}) = \pi_{\mathfrak{e}i}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})\pi_{\mathfrak{a}i}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})q_{\mathfrak{j}\mathfrak{i}}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}) \tag{3}$$

The Q-function $q_{\mathfrak{j}\mathfrak{i}}$ for agent $\mathfrak{j}$ estimated by agent $\mathfrak{i}$, ($\mathfrak{i}, \mathfrak{j} \in \{\mathfrak{e}, \mathfrak{a}\}$), is updated as follows (lines 23 and 24).

$$\begin{aligned} q_{\mathfrak{j}\mathfrak{i}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}) = (1-\alpha_t)q_{\mathfrak{j}\mathfrak{i}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}) \\ +\alpha_t(r_{\mathfrak{i},t} + \gamma \overline{q}_{\mathfrak{j}\mathfrak{i}}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})) \end{aligned} \tag{4}$$

where α_t is the learning rate at time step t.

Figure 2 shows the structure of QRM-SG and focuses on the procedures in one time step.

QRM-SG is guaranteed to converge to best-response strategies in the limit, as stated in the following theorem and proven in the extended version [9].

Assumption 1 *Every state $s \in S$, RM states $v_{\mathfrak{e}} \in V_{\mathfrak{e}}, v_{\mathfrak{a}} \in V_{\mathfrak{a}}$, and actions $a_{\mathfrak{e}} \in A_{\mathfrak{e}}, a_{\mathfrak{a}} \in A_{\mathfrak{a}}$, are visited infinitely often when the number of episodes goes to infinity.*

Assumption 2 *The learning rate α_t satisfies the following conditions for all t:*

1. *$0 < \alpha_t < 1, \sum_{t=0}^{\infty} \alpha_t(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}) = \infty, \sum_{t=0}^{\infty}[\alpha_t(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}})]^2 < \infty$, and the latter two hold uniformly and with probability 1.*
2. *$\alpha_t(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}) = 0$ if $(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}) \neq (s^t, v^t_{\mathfrak{e}}, v^t_{\mathfrak{a}}, a^t_{\mathfrak{e}}, a^t_{\mathfrak{a}})$.*

Assumptions 1 and 2 are standard assumptions and similar to those in Q-learning [17]. Condition 2 in Assumption 2 requires that at each step, only the Q-function elements related to the current state, RM state, and actions are updated.

Assumption 3 *One of the following conditions holds during learning.*

1. *Every stage game $(q_{\mathfrak{e}i}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}), q_{\mathfrak{a}i}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}))$, $\mathfrak{i} \in \{\mathfrak{e}, \mathfrak{a}\}$, for all $t, s, v_{\mathfrak{e}}$, and $v_{\mathfrak{a}}$, has a global optimal point $(\tilde{\pi}_{\mathfrak{e}i}, \tilde{\pi}_{\mathfrak{a}i})$, (i.e., $\tilde{\pi}_{\mathfrak{j}\mathfrak{i}}q_{\mathfrak{j}\mathfrak{i}} \geq \tilde{\pi}'_{\mathfrak{j}\mathfrak{i}}q_{\mathfrak{j}\mathfrak{i}}$ for any $\tilde{\pi}'_{\mathfrak{j}\mathfrak{i}}$, $\mathfrak{j} \in \{\mathfrak{e}, \mathfrak{a}\}$) and agents' rewards in this equilibrium are used to update their Q-functions.*
2. *Every stage game $(q_{\mathfrak{e}i}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}), q_{\mathfrak{a}i}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}))$, $\mathfrak{i} \in \{\mathfrak{e}, \mathfrak{a}\}$, for all $t, s, v_{\mathfrak{e}}$, and $v_{\mathfrak{a}}$, has a saddle point $(\tilde{\pi}_{\mathfrak{e}i}, \tilde{\pi}_{\mathfrak{a}i})$, (i.e., $\tilde{\pi}_{\mathfrak{j}\mathfrak{i}}\tilde{\pi}_{-\mathfrak{j}\mathfrak{i}}q_{\mathfrak{j}\mathfrak{i}} \geq \tilde{\pi}'_{\mathfrak{j}\mathfrak{i}}\tilde{\pi}_{-\mathfrak{j}\mathfrak{i}}q_{\mathfrak{j}\mathfrak{i}}$ for any $\tilde{\pi}'_{\mathfrak{j}\mathfrak{i}}$, $\tilde{\pi}_{\mathfrak{j}\mathfrak{i}}\tilde{\pi}_{-\mathfrak{j}\mathfrak{i}}q_{\mathfrak{j}\mathfrak{i}} \leq \tilde{\pi}_{\mathfrak{j}\mathfrak{i}}\tilde{\pi}'_{-\mathfrak{j}\mathfrak{i}}q_{\mathfrak{j}\mathfrak{i}}$ for any $\tilde{\pi}'_{-\mathfrak{j}\mathfrak{i}}$, $\mathfrak{j} \in \{\mathfrak{e}, \mathfrak{a}\}$), and agents' rewards in this equilibrium are used to update their Q-functions.*

In Assumption 3, $\tilde{\pi}_{\mathfrak{j}\mathfrak{i}}$ denotes the strategy of agent $\mathfrak{j}$ at either the global optimal point or the saddle point, given the Q-functions learned by agent $\mathfrak{i}$. $\tilde{\pi}_{-\mathfrak{j}\mathfrak{i}}$ is the strategy of the agent other than agent $\mathfrak{j}$, i.e., $\tilde{\pi}_{-\mathfrak{e}i} = \tilde{\pi}_{\mathfrak{a}i}, \tilde{\pi}_{-\mathfrak{a}i} = \tilde{\pi}_{\mathfrak{e}i}$. Assumption 3 requires that the stage game at each time step has either a global optimal point or a saddle point.

Theorem 1 *Under Assumptions 1, 2 and 3, the sequence $q_{\mathfrak{i}t} = (q^t_{\mathfrak{e}i}, q^t_{\mathfrak{a}i})$ at time t, $\mathfrak{i} \in \{\mathfrak{e}, \mathfrak{a}\}$ updated by*

$$\begin{aligned} q^{t+1}_{\mathfrak{j}\mathfrak{i}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}) = (1-\alpha_t)q^t_{\mathfrak{j}\mathfrak{i}}(s, v_{\mathfrak{e}}, v_{\mathfrak{a}}, a_{\mathfrak{e}}, a_{\mathfrak{a}}) \\ +\alpha_t\big(r_{\mathfrak{j},t} + \gamma\pi_{\mathfrak{e}i}(\cdot|s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})\pi_{\mathfrak{a}i}(\cdot|s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})q^t_{\mathfrak{j}\mathfrak{i}}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}})\big) \end{aligned} \tag{5}$$

for $\mathfrak{j} \in \{\mathfrak{e}, \mathfrak{a}\}$, where $(\pi_{\mathfrak{e}i}(\cdot|s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}), \pi_{\mathfrak{a}i}(\cdot|s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}))$ is the appropriate type of Nash equilibrium solution for the stage game $(q^t_{\mathfrak{e}i}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}), q^t_{\mathfrak{a}i}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}))$, converges to the Q-functions at a Nash equilibrium $(q^*_{\mathfrak{e}i}, q^*_{\mathfrak{a}i})$.

Note that $(\pi_{\mathfrak{e}i}(\cdot|s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}), \pi_{\mathfrak{a}i}(\cdot|s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}))$ is the appropriate type of Nash equilibrium solution if the strategy corresponds to the global optimal point or the saddle point for the stage game $(q^t_{\mathfrak{e}i}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}), q^t_{\mathfrak{a}i}(s', v'_{\mathfrak{e}}, v'_{\mathfrak{a}}))$. $q^*_{\mathfrak{i}\mathfrak{j}}$ is the Q-function at a Nash equilibrium for agent $\mathfrak{i}$ and learned by agent $\mathfrak{j}$.

6 Experiments

In this section, we evaluate the effectiveness of the proposed QRM-SG method in three case studies. We compare QRM-SG with following baseline methods:

- Nash-Q: we perform Nash Q-learning algorithm developed in [10] and the agents' locations are taken as the state.
- Nash-QAS (Nash Q-learning in augmented state space): to have more information on high-level events, we include an extra binary vector in the state representing whether each event has been encountered or not.
- MADDPG-SG (multi-agent deep deterministic policy gradient with state of the stochastic game): we adopt one of the state-of-the-art MARL baselines —- MADDPG algorithm developed in [16] — and include the agents' locations as the state.
- MADDPG-AS (multi-agent deep deterministic policy gradient in augmented state space): similar to Nash-QAS, we include an extra binary vector in the state to represent high-level information for MADDPG [16].

In each case study, each agent is given a task with sparse rewards to achieve, which can be specified as a reward machine. Each agent receives a reward of one if and only if the task is completed. We have two agents in a 6×6 grid world for three case studies, as shown in Figure 3. Following the notations in the motivational example, we use the blue color to indicate the locations corresponding to the ego agent and the red color for the adversarial agent. The power bases are represented by circles and starting locations are represented by triangles. At each time step, each agent can select from: {up, down, left, right}. Each action has a slip rate of 0.5% and ϵ is set to 0.25.

The first case study is designed as an 'appetizer' to test QRM-SG. Then, we make the ego agent's task more demanding in the second

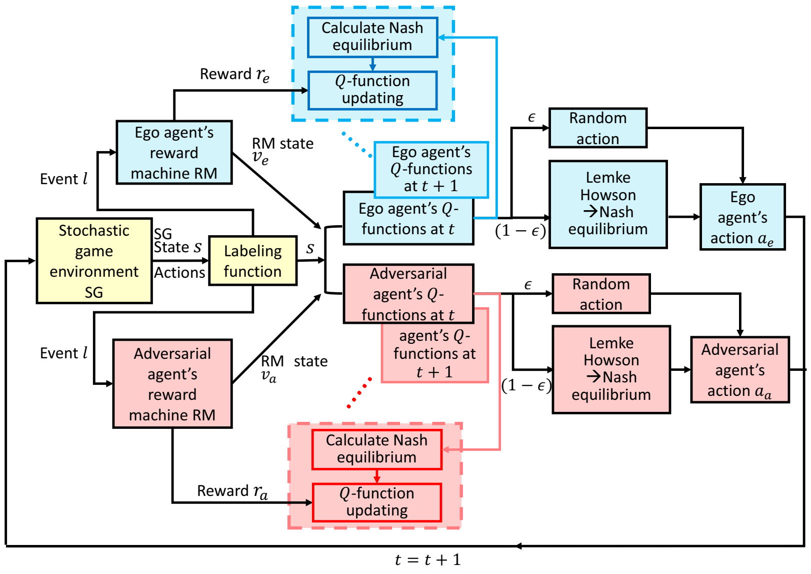

Calculate Nash equilibrium
Q-function updating
Reward r_e
Ego agent's reward machine RM
RM state v_e
Ego agent's Q-functions at $t+1$
Ego agent's Q-functions at t
Event l
Stochastic game environment SG
SG State s
Actions
Labeling function
s
ϵ
Random action
Lemke Howson →Nash equilibrium
$(1-\epsilon)$
Ego agent's action a_e
Adversarial agent's Q-functions at t
agent's Q-functions at $t+1$
Event l
Adversarial agent's reward machine RM
RM state v_a
ϵ
Random action
Lemke Howson →Nash equilibrium
$(1-\epsilon)$
Adversarial agent's action a_a
Calculate Nash equilibrium
Q-function updating
Reward r_a
$t = t + 1$

Figure 2: Flowchart of QRM-SG. Following the motivational example, blue indicates the elements related to the ego agent and red indicates the elements related to the adversarial agent.

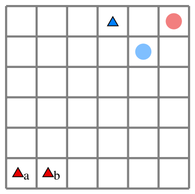
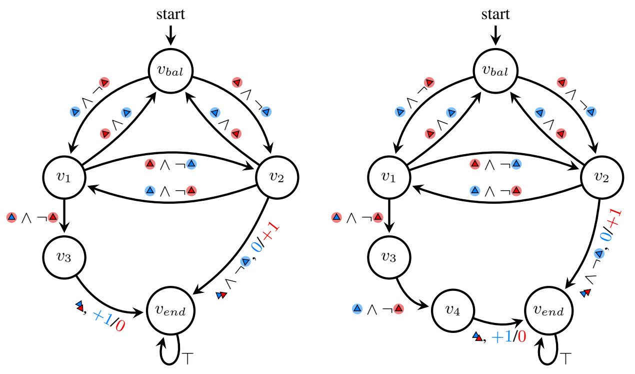


Figure 3: The grid world for case studies. In Case Study I and II, the adversarial agent starts from location 'a'. In Case Study III, the adversarial agent starts from location 'a' or 'b'.

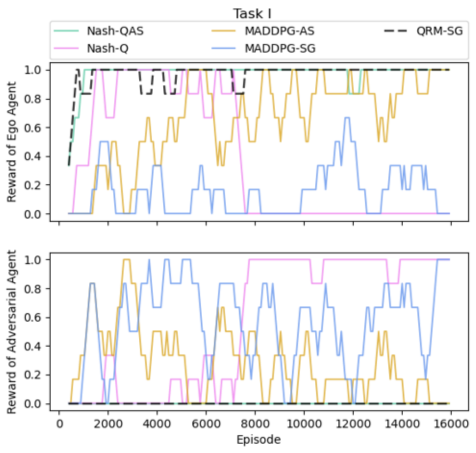


(a) Case study I. (b) Case studies II and III.

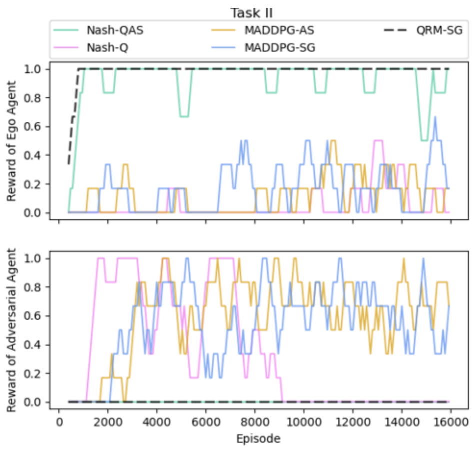
Figure 4: The reward machines used in the case studies. In each case study, the reward machine of each agent has the same structure, only the reward differs. Hence, we present the rewards for agent ▲ in blue, and the rewards for agent ▲ in red. The reward is sparse (zero for both agents when not specified on a transition). Omitted transitions are self-looping transitions with reward 0.

and third case studies. Randomnesses in the starting locations of the adversarial agent are included in the third case study. We define that one agent is captured by the other agent when the distance between two agents is smaller than 2. Each agent is considered to complete the task after capturing the other agent. Only when an agent is the more powerful of the two agents, it has the capability to capture the other agent. We set different conditions for the ego agent to be the more powerful in different case studies. The tasks for agents to complete are specified as reward machines, which are demonstrated in Figure 4. The adversarial agent is given the same task in three case studies, which requires it to be the more powerful by reaching its power base (▲) and then capture the ego agent (▲). In Case Study I, the ego agent is required to first reach its own power base (▲), then destroy/reach the adversarial agent's power base (▲) to be the more powerful, and capture the adversarial agent afterward (▲). In Case Study II, the required sequential events for the ego agent to be powerful are: reaching its power base (▲), reaching the adversarial agent's power base (▲), reaching its power base (▲). These events demonstrate the scenario in that the ego agent first gets energy at its power base, destroys the adversarial agent's power base using most of its energy, and then gets recharged to capture the adversarial agent. Case Study III is different from Case Study II in that the adversarial agent randomly samples the starting location from 2 possible locations. In all case studies, we define that after the adversarial agent arrives at its power base, the ego agent has to reach its own power base again to be able to destroy the adversarial agent's power base. After a power base is destroyed, the corresponding agent is not possible to be more powerful.

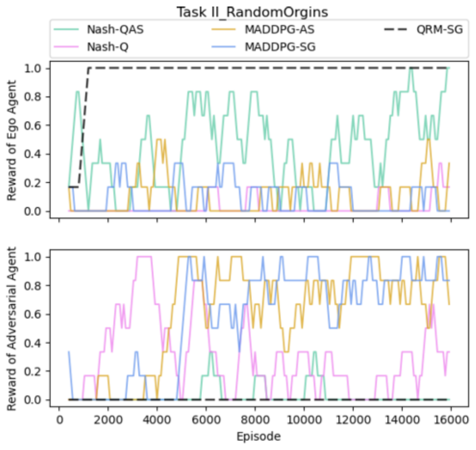
To better analyze the results, each case study is designed to have best-response strategies for agents. The ego agent is expected to complete the required sequence of events to be the more powerful agent, capture the adversarial agent, and complete the task resulting in a cumulative reward of 1. As the adversarial agent is far away from its power base, its power base would be destroyed by the ego agent before it arrives at a Nash equilibrium. Thus, the adversarial agent would fail to complete its task. The learning processes of each agent in three case studies are plotted in Figures 5a to 5c. Every 80 episodes, we stop learning, test the algorithms' performance, and



(a) Case study I.



(b) Case study II.



(c) Case study III.

Figure 5: Cumulative reward comparison for each case study. Smoothed plot with a rolling window of size 6. QRM-SG is the proposed method.

save the cumulative rewards of each agent. In Case Study I, QRM-SG finds the Nash equilibrium in around 7500 episodes. Nash-QAS finds the Nash equilibrium after 12000 episodes, which indicates that using the augmented state can be sufficient for the ego agent to learn to complete the task. When learning by Nash-Q, the adversarial agent completes the task. The reason can be that the task for the adversarial agent is much easier. MADDPG-SG and MADDPG-AS do not converge within 16000 episodes. In MADDPG-SG, the ego agent needs more episodes for completing the task compared to MADDPG-AS. One possible reason is that MADDPG-AS perceives more information represented in the augmented state space. In Case Study II, the policies of agents reach the Nash equilibrium after around 1000 episodes utilizing QRM-SG, while baselines fail to converge to the Nash equilibrium. In Case Study III, QRM-SG finds the Nash equilibrium in around 1500 episodes, while baselines have difficulty converging to the Nash equilibrium. The ego agent learned by Nash-QAS receives a higher reward than other baselines. The ego agent learned by other baselines rarely finishes the task. From the results of the three case studies, QRM-SG outperforms the four baseline methods, where the ego agent can accomplish the task at a Nash equilibrium in several thousand episodes.

An analysis of the effect of ϵ on the performance of QRM-SG is available in the extended version [9]. Additionally, an extensive evaluation of QRM-SG on a 12×12 grid world is conducted in the extended version [9].

7 Conclusions

In this paper, we introduce the utilization of reward machines to expose the structure of non-Markovian reward functions for the learning agents in two-agent general-sum stochastic games. Each task is specified by a reward machine. We propose QRM-SG as a variant of Q-learning for the setting of stochastic games and integrated with reward machines to learn best-response strategies at a Nash equilibrium for each agent. We prove that QRM-SG converges to Q-functions at a Nash equilibrium under certain conditions. Three case studies are conducted to evaluate the performance of QRM-SG.

This paper opens the door for using reward machines in stochastic games. First, one immediate extension is to jointly learn reward machines and best-response strategies during RL. Second, extending the methodology to general-sum stochastic games with more agents is worth further investigation. Finally, the same methodology can be readily applied to other forms of RL, such as model-based RL, or actor-critic methods.

Acknowledgements

This work is supported by NASA University Leadership Initiative program (Contract No. NNX17AJ86A, PI: Yongming Liu, Technical Officer: Anupa Bajwa). This work is also supported by grants NSF CNS 2304863 and ONR N00014-23-1-2505 (for Zhe Xu and his students), and ONR N00014-22-1-2254 and ARO W911NF-20-1-0140 (for Ufuk Topcu and his students).

References

[1] Lucian Buşoniu, Robert Babuška, and Bart De Schutter, 'Multi-agent reinforcement learning: An overview', in *Innovations in Multi-Agent Systems and Applications - 1*, 183–221, Springer Berlin Heidelberg, (2010).

[2] Alberto Camacho, Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Valenzano, and Sheila A. McIlraith, 'LTL and beyond: Formal languages for reward function specification in reinforcement learning', in *IJCAI'2019*, pp. 6065–6073, (7 2019).

[3] Giuseppe De Giacomo, Marco Favorito, Luca Iocchi, Fabio Patrizi, and Alessandro Ronca, 'Temporal logic monitoring rewards via transducers', in *Proceedings of the International Conference on Principles of*

Knowledge Representation and Reasoning, volume 17, pp. 860–870, (2020).

[4] Giuseppe De Giacomo, Luca Iocchi, Marco Favorito, and Fabio Patrizi, ‘Restraining bolts for reinforcement learning agents’, in *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pp. 13659–13662, (2020).

[5] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson, ‘Counterfactual multi-agent policy gradients’, in *Proceedings of the AAAI conference on artificial intelligence*, volume 32, (2018).

[6] Jakob Foerster, Nantas Nardelli, Gregory Farquhar, Triantafyllos Afouras, Philip HS Torr, Pushmeet Kohli, and Shimon Whiteson, ‘Stabilising experience replay for deep multi-agent reinforcement learning’, in *International conference on machine learning*, pp. 1146–1155. PMLR, (2017).

[7] Jayesh K Gupta, Maxim Egorov, and Mykel Kochenderfer, ‘Cooperative multi-agent control using deep reinforcement learning’, in *International conference on autonomous agents and multiagent systems*, pp. 66–83. Springer, (2017).

[8] Mohammadhosein Hasanbeig, Natasha Yogananda Jeppu, Alessandro Abate, Tom Melham, and Daniel Kroening, ‘Deepsynth: Automata synthesis for automatic task segmentation in deep reinforcement learning’, in *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 7647–7656, (2021).

[9] Jueming Hu, Jean-Raphaël Gaglione, Yanze Wang, Zhe Xu, Ufuk Topcu, and Yongming Liu, ‘Reinforcement learning with reward machines in stochastic games’, *arXiv preprint arXiv:2305.17372*, (2023).

[10] Junling Hu and Michael P Wellman, ‘Nash q-learning for general-sum stochastic games’, *Journal of machine learning research*, **4**(Nov), 1039–1069, (2003).

[11] Rodrigo Toro Icarte, Toryn Klassen, Richard Valenzano, and Sheila McIlraith, ‘Using reward machines for high-level task specification and decomposition in reinforcement learning’, in *International Conference on Machine Learning*, pp. 2107–2116. PMLR, (2018).

[12] Rodrigo Toro Icarte, Toryn Q. Klassen, Richard Anthony Valenzano, and Sheila A. McIlraith, ‘Using reward machines for high-level task specification and decomposition in reinforcement learning’, in *ICML’2018*, pp. 2112–2121, (2018).

[13] Carlton E Lemke and Joseph T Howson, Jr, ‘Equilibrium points of bimatrix games’, *Journal of the Society for industrial and Applied Mathematics*, **12**(2), 413–423, (1964).

[14] Borja G León and Francesco Belardinelli, ‘Extended markov games to learn multiple tasks in multi-agent reinforcement learning’, *arXiv preprint arXiv:2002.06000*, (2020).

[15] Xiaomin Lin, Stephen C Adams, and Peter A Beling, ‘Multi-agent inverse reinforcement learning for certain general-sum stochastic games’, *Journal of Artificial Intelligence Research*, **66**, 473–502, (2019).

[16] Ryan Lowe, Yi I Wu, Aviv Tamar, Jean Harb, OpenAI Pieter Abbeel, and Igor Mordatch, ‘Multi-agent actor-critic for mixed cooperative-competitive environments’, *Advances in neural information processing systems*, **30**, (2017).

[17] Francisco S Melo, ‘Convergence of q-learning: A simple proof’, *Institute Of Systems and Robotics, Tech. Rep*, 1–4, (2001).

[18] David H Mguni, Yutong Wu, Yali Du, Yaodong Yang, Ziyi Wang, Minne Li, Ying Wen, Joel Jennings, and Jun Wang, ‘Learning in nonzero-sum stochastic games with potentials’, in *Proceedings of the 38th International Conference on Machine Learning*, eds., Marina Meila and Tong Zhang, volume 139 of *Proceedings of Machine Learning Research*, pp. 7688–7699. PMLR, (18–24 Jul 2021).

[19] Devaprakash Muniraj, Kyriakos G. Vamvoudakis, and Mazen Farhood, ‘Enforcing signal temporal logic specifications in multi-agent adversarial environments: A deep q-learning approach’, in *2018 IEEE Conference on Decision and Control (CDC)*, pp. 4141–4146, (2018).

[20] John Nash, ‘Non-cooperative games’, *Annals of mathematics*, 286–295, (1951).

[21] Daniel Neider, Jean-Raphael Gaglione, Ivan Gavran, Ufuk Topcu, Bo Wu, and Zhe Xu, ‘Advice-guided reinforcement learning in a non-markovian environment’, in *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pp. 9073–9080, (2021).

[22] Shayegan Omidshafiei, Jason Pazis, Christopher Amato, Jonathan P How, and John Vian, ‘Deep decentralized multi-task multi-agent reinforcement learning under partial observability’, in *International Conference on Machine Learning*, pp. 2681–2690. PMLR, (2017).

[23] Gregory Palmer, Karl Tuyls, Daan Bloembergen, and Rahul Savani, ‘Lenient multi-agent deep reinforcement learning’, *arXiv preprint arXiv:1707.04402*, (2017).

[24] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson, ‘Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning’, in *International conference on machine learning*, pp. 4295–4304. PMLR, (2018).

[25] Gavin Rens and Jean-François Raskin, ‘Learning non-markovian reward models in mdps’, *arXiv preprint arXiv:2001.09293*, (2020).

[26] Gavin Rens, Jean-François Raskin, Raphaël Reynouad, and Giuseppe Marra, ‘Online learning of non-markovian reward models’, *arXiv preprint arXiv:2009.12600*, (2020).

[27] Jeffrey O. Shallit, *A Second Course in Formal Languages and Automata Theory*, Cambridge University Press, 2008.

[28] Peter Sunehag, Guy Lever, Audrunas Gruslys, Wojciech Marian Czarnecki, Vinicius Zambaldi, Max Jaderberg, Marc Lanctot, Nicolas Sonnerat, Joel Z Leibo, Karl Tuyls, et al., ‘Value-decomposition networks for cooperative multi-agent learning’, *arXiv preprint arXiv:1706.05296*, (2017).

[29] Richard S Sutton and Andrew G Barto, *Reinforcement learning: An introduction*, MIT press, 2018.

[30] Rodrigo Toro Icarte, Ethan Waldie, Toryn Klassen, Rick Valenzano, Margarita Castro, and Sheila McIlraith, ‘Learning reward machines for partially observable reinforcement learning’, *Advances in neural information processing systems*, **32**, (2019).

[31] Zhe Xu, Ivan Gavran, Yousef Ahmad, Rupak Majumdar, Daniel Neider, Ufuk Topcu, and Bo Wu, ‘Joint inference of reward machines and policies for reinforcement learning’, in *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 30, pp. 590–598, (2020).

[32] Zhe Xu, Bo Wu, Aditya Ojha, Daniel Neider, and Ufuk Topcu, ‘Active finite reward automaton inference and reinforcement learning using queries and counterexamples’, in *International Cross-Domain Conference for Machine Learning and Knowledge Extraction*, pp. 115–135. Springer, (2021).

[33] Kaiqing Zhang, Zhuoran Yang, Han Liu, Tong Zhang, and Tamer Basar, ‘Fully decentralized multi-agent reinforcement learning with networked agents’, in *International Conference on Machine Learning*, pp. 5872–5881. PMLR, (2018).

[34] Tianhao Zhang, Qiwei Ye, Jiang Bian, Guangming Xie, and Tie-Yan Liu, ‘Mfvfd: A multi-agent q-learning approach to cooperative and non-cooperative tasks.’, in *IJCAI*, pp. 500–506, (2021).

[35] Xiangyuan Zhang, Kaiqing Zhang, Erik Miehling, and Tamer Basar, ‘Non-cooperative inverse reinforcement learning’, in *Advances in Neural Information Processing Systems*, eds., H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, volume 32. Curran Associates, Inc., (2019).