Complex & Intelligent Systems (2024) 10:3273–3282
https://doi.org/10.1007/s40747-023-01326-7

ORIGINAL ARTICLE

Embedding multi-agent reinforcement learning into behavior trees with unexpected interruptions

Xianglong Li[1] · Yuan Li[1] · Jieyuan Zhang[1] · Xinhai Xu[1] · Donghong Liu[1]

Received: 2 September 2022 / Accepted: 2 December 2023 / Published online: 25 January 2024

Abstract
Behavior trees have attracted great interest in computer games and robotic applications. However, it lacks the learning ability for dynamic environments. Previous works combining behavior trees with reinforcement learning either need to construct an independent sub-scenario or train the learning method over the whole game, which is not suited for complex multi-agent games. In this paper, a framework is proposed, named as MARL-BT, that embeds multi-agent reinforcement learning methods into behavior trees. Following the running mechanism of behavior trees, we design the way of collecting samples and the training procedure. Further, we point out a special phenomenon in MARL-BT, i.e., the unexpected interruption, and present an action masking technique to remove its harmful effect on learning performance. Finally, we make extensive experiments on the 11 versus 11 full game in Google Research Football. The introduced MARL-BT framework could get an 11.507% improvement compared to pure BT for certain scenarios. The action masking technique could greatly improve the performance of the learning method, i.e., the final reward is improved around 100% times for a sub-task.

Keywords Behavior trees · Multi-agent reinforcement learning · Unexpected interruption · Complex multi-agent game tasks

Introduction

Behavior trees (BTs) is a popular control tool in robotics and computer games, such as real-time strategy games [1, 2], unmanned aerial vehicles [3, 4], mobile robots [5–7] and so on. It is widely appreciated for its modularity, scalability and reactivity [8]. However, it needs much human knowledge and takes lots of effort to design a good one for a specific scenario [9]. The behavioral structure of the manual design is fixed which lacks the ability to dynamically adapt to the decision-making environment.

Some works have focused on integrating reinforcement learning with the design of BTs to enhance adaptability of BTs in dynamic environments. The work in Ref. [10] learns the fallback node to decide whether a child node should be executed. The work in Ref. [11] uses the Q-learning method to optimize the structure of the tree by selecting a proper node for each control node in the tree. Much works has been done to substitute an action node in the tree with an entire reinforcement learning model. The work in Ref. [12] uses a Q-learning method and Ref. [13] considers Proximal Policy Optimization to learn an action node. Further, the work in Ref. [14] proposes a hierarchical reinforcement learning approach MAXQ, which is used to optimize control nodes in the higher layer and learn the action node in the low layer. However, this kind of work needs to construct a sub-scenario to train the reinforcement learning model and then embed the model into the tree. They are only feasible for simple problems, which is hard to be used for complex games. Such problems often involves multiple agents. Different multi-agent reinforcement learning (MARL) methods have been proposed, such as MADDPG [15], QMIX [16], MAPPO [17]. However, they are only examined on mini-games like StarCraft II micromanagement tasks [18] and simple ball games [19]. With the number of agents increasing, all MARL methods will fail to obtain a reasonable solution in a finite time.

In this paper we consider to combine MARL methods with BTs for solving complex problem. The BT is a good tool to decompose a large task into multiple smaller sub-tasks [20], which could be solved by MARL methods. In this way, we do not have to solve the complex problem directly with MARL methods, which would need quite a lot of time

✉ Yuan Li
nudt_liyuan@163.com

✉ Donghong Liu
liu_donghong@sina.cn

1 Academy of Military Sciences, Beijing 100091, China

and expensive resources. We propose a framework, named as MARL-BT, to embed MARL methods into BTs. Different from previous works, we neither need to construct an independent sub-scenario nor run over the whole game. We design a procedure to train the MARL model during the run of BTs. Samples for the MARL method are collected when corresponding sub-task is activated. The episode for the MARL method does not correspond to the full run of the game but is just a segment, which improves the efficiency of collecting samples.

Further, we notice a phenomenon that happened among sub-tasks when combining BTs with MARL. The sub-tasks decomposed by BTs have different priorities and may control some common agents. The conflict happens between the learning-based task with lower priority and the rule-based task with higher priority, which control the same agent. Once the two tasks are both activated, the action of the agent computed by the learning-based task will not be executed. For example, in the StarCraft II game, some sappers on mining tasks may be urgently dispatched to perform an offensive task. We define such conflicts as unexpected interruptions. When interruptions happen, actions computed by the agent network are in fact not executed. However, we store such samples into the replay buffer. Training the network with such bad samples will lead to higher value prediction errors. This error will also be accumulated during the computation, which would hinder finding a good policy. Therefore, we design an action masking technique that could remove the impact of actions generated by other sub-tasks.

To clearly clarify the problem, we provide an example which is shown in Fig. 1. The BT decomposes the soccer backfield task into penalty-area defense, left-side, middle and right-side sub-tasks, which is shown as the right sub-figure. In BT, the middle sub-task is realized by the MARL method, while others not. The penalty-area defense sub-task has the highest priority, which means players belonging to other tasks may be reassigned to this sub-task. The conflict using of a common player between two sub-tasks is called the unexpected interruption.

Finally, we make extensive experiments on the 11 versus 11 full game in Google Research Football to examine the proposed methods. With the framework MARL-BT, we use the MARL method to replace certain sub-task of BTs. We could find that MARL converges quickly to perform better than the rules. With the trained model, the performance of BTs is also significantly improved. Compared with pure BTs, MARL-BT could improve the win rate by around 11.507% for certain scenarios. The action masking technique could greatly improve the performance of the learning method, i.e., the final reward is improved around 100% times for a sub-task.

Background

Behavior trees

BTs originated within the realm of computer games [21], serving as integral planning and decision-making tools [12] for the effective modeling and control of autonomous agents. A BT consists of nodes that perform specific actions or conditions. These nodes are categorized into three main types: control nodes, action nodes, and condition nodes. The control nodes control the flow of execution in the BT, which is divided into three categories, i.e., *Sequence*, *Fallback*, and *Parallel*. The sequence node returns a Success when all its child nodes succeed. The fallback node returns a Failure when all of its children fail. In parallel nodes, there are M child nodes, and each iteration requires the execution of all nodes. If a node returns Failure or if all nodes return Success, the parent node then returns Failure or Success. The action node is always a leaf node, employed to execute specific actions associated with the node, and it returns the execution outcome to the parent node based on the status of the action execution. The condition node corresponds to the if-else structure in programming languages, serving to assess whether the current environmental state satisfies the specified logical conditions. If the condition test is True, the condition node returns Success to the parent node. If it is False, it returns Failure.

Periodically, the execution of a BT starts from its root node, a process facilitated by the generation of a tick signal that traverses the tree branches in accordance with the distinct characteristics of each node type [22]. A node is deemed eligible for execution solely upon receipt of the tick signal. Upon execution, a child node promptly communicates its status to the parent, status either Running if its execution is ongoing, Success if its objective has been attained, or Failure if unsuccessful. The tick signal allows a BT to respond and adapt to changes in the environment or the agent's state in real-time. By regularly updating the tree's nodes, the agent can make informed decisions and perform appropriate actions based on the current state.

Multi-agent reinforcement learning

A multi-agent game can be modeled as a multi-agent extension of the Markov Decision Process [23]. It is represented by the tuple $\langle \mathcal{N}, \mathcal{S}, \mathcal{A}, R, P \rangle$, where $\mathcal{N} = \{1, 2, \ldots, N\}$ is the set of agents. $\mathcal{S} = \left\{S^t\right\}_{t=0}^{T}$ is the environment state where S^t is the state at time t and T is the maximum time steps. The terminal state S^T represents the final state when the stopping condition is satisfied. $\mathcal{A} = \{A_i\}_{i=0}^{N}$ represents the action space for all agents. $R : \mathcal{S} \times \mathcal{A} \rightarrow \mathcal{R}$ is the reward function. $\mathcal{P} = \{P_{ss'}^{a} \mid s, s' \in \mathcal{S}, a \in \mathcal{A}\}$ is the state transition function,

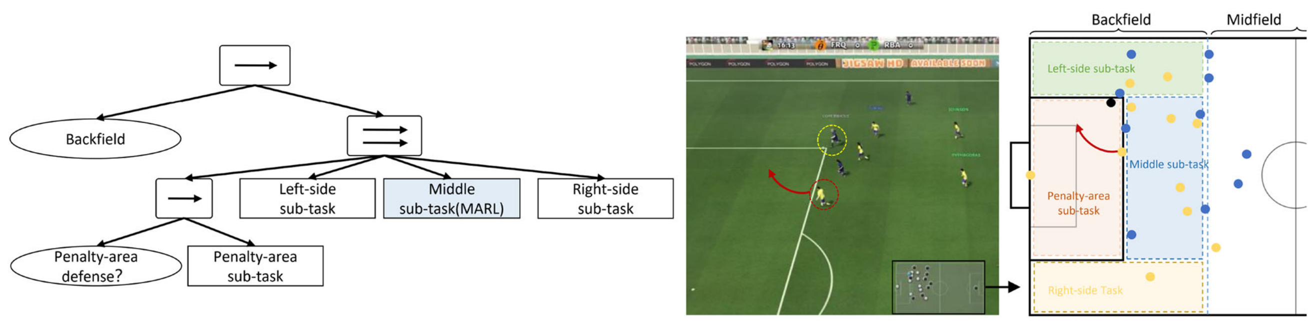


Fig. 1 An example illustrating the combination of BT with MARL

and $P^a_{ss'}$ gives the probability from state s to state s' if the action a is taken. To compute the optimal policy, the state value function $V^{\pi}(s)$ under policy π is introduced, which is calculated by accumulated rewards with the discount factor $\lambda \in [0, 1]$, see (1), where $\mathbb{E}$ is the expectation:

$$V_{\pi}(s) = \mathbb{E}_{\pi}\left\{\sum_{k=0}^{\infty} \lambda^k r^{t+k+1} \middle| S^t = s\right\} \tag{1}$$

Based on $V_{\pi}(s)$, the state-action value $Q_{\pi}(s, a)$ is defined as (2)

$$\begin{aligned} Q_{\pi}(s, a) &= \mathbb{E}_{\pi}\left\{\sum_{k=0}^{\infty} \lambda^k r^{t+k+1} \middle| S^t = s, A^t = a\right\} \\ &= r^{t+1} + \lambda \sum_{s'} P^a_{ss'} V^{\pi}(s') \end{aligned} \tag{2}$$

The optimal policy is obtained by iteratively updating the Q-value function, for which a popular one is that of Q-learning, see (3). α is the updating step:

$$Q(s, a) = Q(s, a) + \alpha[r + \gamma \max_{a} Q(s', a) - Q(s, a)] \tag{3}$$

For multi-agent games, a popular solution approach is the value-decomposition based MARL algorithms [16, 23, 24]. Each agent i is associated with a neural network, which is used to compute the individual value Q_i based on its observation. During the training process, the global state-action value Q_{tot} is computed as a function of Q_i for each agent. At present time, different algorithms are proposed to study the relation between Q_{tot} and Q_i. For example, VDN [24] expresses Q_{tot} as a sum of Q_i, i.e., $Q_{\text{tot}} = \sum_i Q_i$. QMIX [16] uses a continuous monotonic function in form of a mixing network to express this relation, i.e., $Q_{\text{tot}} = f(Q_1, Q_2, \ldots, Q_n)$. For value-decomposition based method, a general principle Individual-Global-Max (IGM) [25] should be satisfied, which is shown as (4). It guarantees that a global argmax performed on Q_{tot} yields the same result as a set of individual argmax operations performed on each Q_i:

$$\begin{aligned} &\arg\max_{\boldsymbol{a}\in\mathcal{A}} Q_{\text{tot}}(\boldsymbol{o}, \boldsymbol{a}) \\ &= \left(\arg\max_{a_1\in\mathcal{A}} Q_1(o_1, a_1), \ldots, \arg\max_{a_n\in\mathcal{A}} Q_n(o_n, a_n)\right) \end{aligned} \tag{4}$$

Integrating reinforcement learning with BTs allows for the potential adoption of adaptive strategies by the BTs. BTs combined with reinforcement learning is not an entirely new concept, there have been some works that integrate reinforcement learning into BTs to relieve manual programming BTs efforts. References [11, 26] generate BTs through reinforcement learning, while the common approach is to embed reinforcement learning as a learning node into BTs to improve the adaptability of a predefined BT [10, 27]. However, the current research on BTs combined with reinforcement learning focuses on simple single-agent tasks, i.e., BTs combined with single-agent reinforcement learning. And the learning process is independent of BTs, i.e., the training is conducted in a separate sub-scenario. Once a learning model is obtained, it is embedded into the tree. However, for complex multi-agent game tasks, such a separate sub-scenario is normally hard to be constructed. It is better to train the model during the run of the behavior trees. We present such a training procedure in this paper.

Methodology

In this section, we present a framework for combining BTs with MARL algorithms, which is named as MARL-BT, and provide a detailed description for the training procedure of MARL-BT. In addition, we introduce unexpected interruptions that can occur during MARL-BT training, and present solutions for such problems.

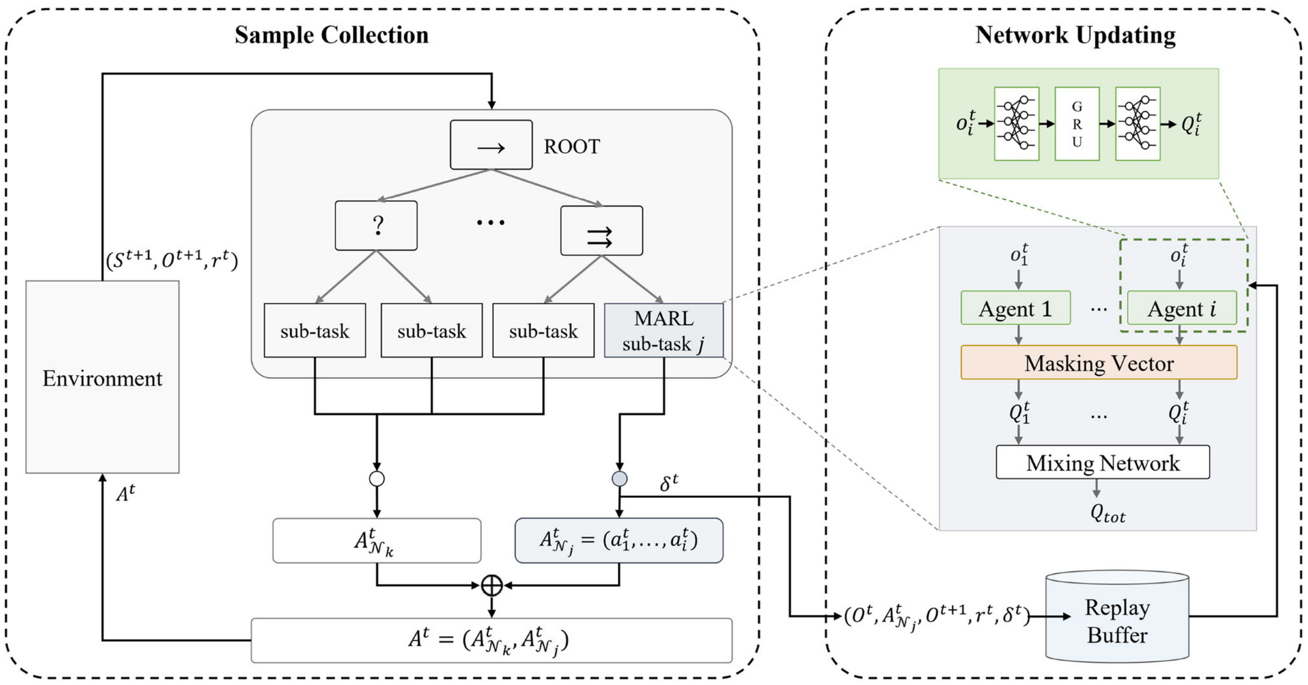


Fig. 2 The framework of MARL-BT

MARL-BT architecture

A BT is designed to decompose a complex task into multiple sub-tasks with different goals. The number of sub-tasks (action nodes) in BTs is denoted by J. The set of all agents is denoted by $\mathcal{N} = \mathcal{N}_1 \bigcup \mathcal{N}_2 \ldots \mathcal{N}_J$. Each sub-task j is denoted by $\langle \mathcal{N}_j, \mathcal{T}_j, g_j, p_j, c_j \rangle$, where $\mathcal{N}_j \in \mathcal{N}$ is the set of agents that are assigned to sub-task j, $\mathcal{T}_j$ is the set of time steps, p_j is the priority value and g_j is the goal of sub-task j. c_j is a binary value, for which 1 means sub-task j is activated while 0 means not. All sub-tasks are executed following the control flow of BTs, i.e., Sequence, Fallback, or Parallel. The whole framework MARL-BT is shown as Fig. 2. It consists of two main parts, the left part involves sample collection, and the right part concerns network updating. The sample collection is responsible for interacting with the environment and generating samples, which are then saved into the replay buffer. The core module of sample collection is the BT with a MARL sub-task. The network structure of the MARL sub-task is shown in the right part, which is updated by samples in the replay buffer.

The entire sampling and learning procedure of the framework MARL-BT is delineated in Algorithm 1. The input is a BT with a MARL node, and the output is the trained MARL model parameterized by θ. The sample collection runs in accordance with the operational mechanism of BTs, initiating the execution process of the BT. Upon a MARL sub-task j receives the tick signal (Step 9), the MARL node agent action set $A^t_{\mathcal{N}_j}$ and action masking vector δ^t are obtained

Algorithm 1 Training Procedure of MARL-BT

```
Input: BT with a MARL node defined by $\langle \mathcal{N}_j, \mathcal{T}_j, g_j \rangle$
Output: Optimized MARL neural network parameters $\theta$
1: Initialize replay buffer $D$ and network parameters $\theta$ in MARL method;
2: for m = 1,2,...,M do
3:     Initialize an empty episode batch;
4:     for t = 1,2,..., $\mathcal{T}$ do
5:         Execute the BT with tick signal
6:         for ticked non-MARL node $k \in \bar{\mathcal{J}}$ do
7:             Compute actions $A^t_{\mathcal{N}_k} = \{a^t_i, i \in \mathcal{N}_k\}$
8:         end for
9:         if the MARL node $j \in \hat{\mathcal{J}}$ is ticked then
10:            Collect observations $O^t = \{o^t_i, i \in \mathcal{N}_j\}$;
11:            $A^t_{\mathcal{N}_j}, \delta^t$ = Algorithm2($\mathcal{N}_k, \mathcal{N}_j, O^t$), introduced in section "Learning with unexpected interruptions";
12:            Execute actions $A^t = \{A^t_{\mathcal{N}_k}, A^t_{\mathcal{N}_j}\}$;
13:            Get new observations $O^{t+1}$ and reward $r^t$;
14:            Insert $(O^t, A^t_{\mathcal{N}_j}, O^{t+1}, r^t, \delta^t)$ in the batch;
15:            if The sub-task goal $g_j$ is finished, or $t \geq \mathcal{T}_j$ then
16:                Store the batch into the buffer $D$;
17:                Reset the batch to be empty;
18:            end if
19:        end if
20:    end for
21: end for
22: if UPDATE then
23:    Randomly sample batches from $D$;
24:    Calculate the loss according to (6);
25:    Perform gradient descent for neural networks.
26: end if
27: return $\theta$
```

by Algorithm 2. $A^t_{\mathcal{N}_j}$ along with actions generated by other active sub-tasks $A^t_{\mathcal{N}_k}$ is passed to the environment. At the next time slot $t+1$, a reward r^t, global state S^{t+1} and observations O^{t+1} are obtained through the environment. A sample $(O^t, A^t_{\mathcal{N}_j}, O^{t+1}, r^t, \delta^t)$ is then acquired and stored in the replay buffer.

Finally, we utilize the keyword 'UPDATE' to regulate the frequency of updating the neural networks (Steps 22–26). Specifically, the network structure comprises $|\mathcal{N}_j|$ agent networks and a mixing network, following the idea of QMIX [16]. Each time we take samples from the replay buffer, which are sent to agent networks. The Q-values are computed for all agents, which are then disposed by the designed masking mechanism to handle unexpected interruptions introduced in section "Learning with unexpected interruptions". Then a mixing network is used to compute the total Q-value Q_{tot}, and the loss for the whole neural networks is computed as (5) and (6). The loss computation follows the way of DQN with double network [28]. θ' represents parameters of target networks and θ represents parameters of current networks:

$$Q_{\text{tot}}(O^t, A^t_{\mathcal{N}_j}; \theta) = f(Q_i((o^t_i, a^t_i; \theta), i \in \mathcal{N}_j)) \tag{5}$$

$$\text{loss} = \frac{1}{|\mathcal{T}_j|}\sum_{t=1}^{|\mathcal{T}_j|}(r^t + \gamma \max_A Q_{\text{tot}}(O^{t+1}, A^{t+1}_{\mathcal{N}_j}; \theta') - Q_{\text{tot}}(O^t, A^t_{\mathcal{N}_j}; \theta)) \tag{6}$$

Learning with unexpected interruptions

In this section, we introduce unexpected interruptions that happened in the framework MARL-BT. There are multiple sub-tasks organized by control flow nodes of BTs. Many sub-tasks run in overlapped time periods. In the case that they control some common agents, different priorities will make sure that no conflict will happen among sub-tasks. For the sub-task realized by MARL method, a higher sub-task that controls common agents may happen in BTs. It leads that some agents controlled by MARL method may be scheduled by other sub-tasks with the higher priority during its decision period. These agents may also come back to the sub-task with MARL method again when sub-tasks with higher priorities release the control of them. The actions of agents for MARL method generated by sub-task with higher priority are defined as unexpected interruptions. Such interruptions will affect the learning efficiency of MARL methods. The reason is that unexpected interruptions will be taken into account in the computation of Q-value for each agent. This will amplify the bias of the learned value function, leading to high prediction errors which affect the updating of Q-values of agents. The errors will be accumulated with the increasing number of unexpected interruptions, which will hinder finding a favorable strategy.

Algorithm 2 MARL Node Action and Masking Vector Calculation

```
Input: Non-MARL node agent set $\mathcal{N}_k$, MARL node agent set $\mathcal{N}_j$ and
    its observation set $O^t$ at the current time $t$
Output: Current MARL node action set $A^t_{\mathcal{N}_j}$ and masking vector $\delta^t$
 1: Initialized $\epsilon$-greedy strategy;
 2: for agent $i \in \mathcal{N}_j$ do
 3:   if a random probability $p_{rand} < \epsilon$ then
 4:     $a^t_i = \arg\max_{a_i \in A} Q_i(o^t_i, a^t_i)$;
 5:   else
 6:     $a^t_i = random(|A|)$;
 7:   end if
 8:   if there exist node $k \in \bar{\mathcal{J}}$ such that agent $i \in \mathcal{N}_k$ then
 9:     Compute masking vector $\delta^t_i$ according to Eq. (7);
10:   end if
11: end for
12: return $A^t_{\mathcal{N}_j} = \{a^t_i, i \in \mathcal{N}_j\}$, $\delta^t = \{\delta^t_i, i \in \mathcal{N}_j\}$
```

Here we introduce a action masking mechanism to deal with unexpected interruptions. In the BTs, the set of indexes of all sub-tasks is denoted by $\mathcal{J} = \{\hat{\mathcal{J}} \bigcup \bar{\mathcal{J}}\}$. $\hat{\mathcal{J}}$ represents the set of sub-tasks realized by MARL methods while $\bar{\mathcal{J}}$ represents the set of sub-tasks realized by rules. For a MARL node $j \in \hat{\mathcal{J}}$, considering that agents in MARL nodes may also be included in other nodes, we compute the agent action set $A^t_{\mathcal{N}_j}$ and action masking vector δ^t by Algorithm 2. In Step 2, each agent $i \in \mathcal{N}_j$ computes its action a^t_i based on its observation o^t_i. To facilitate action exploration, we employ the ϵ-greedy strategy. Meanwhile, the action masking vector according to Eq. (7). This means that $\delta^t_i = 0$ if there exist an activated node k ($c_k = 1$) with higher priority p_k generate actions for a common agent i, i.e., $i \in \mathcal{N}_j \bigcup \mathcal{N}_k$. With the action masking vector, the transition should be expressed as $(O^t, A^t_{\mathcal{N}_j}, O^{t+1}, r^t, \delta^t)$. To improve the learning performance, we consider to change the reward for such samples, if $r^t = 0$, then set $r^t = r^t + \varepsilon$, and ε is small positive value. Then the Q_{tot} is computed as (8) and the loss is computed still as (6)

$$\delta^t_i = \begin{cases} 0, & i \in \mathcal{N}_j \bigcup \mathcal{N}_k, c_j = c_k = 1, \text{ and } p_k > p_j \\ 1, & \text{Otherwise} \end{cases} \tag{7}$$

$$Q_{\text{tot}}(O^t, A^t_{\mathcal{N}_j}; \theta) = f(Q_i(o^t_i, a^t_i; \theta) * \delta^t_i, i \in \mathcal{N}_j) \tag{8}$$

Experiments

In this section, we verify the effectiveness of the proposed MARL-BT framework through extensive experiments. We firstly describe the environment and experimental settings.

With a given BT, we show the performance improvement brought by MARL method, and the action masking technique for the learning efficiency.

Experiment settings

We conduct experiments in a challenging task, i.e., Google Research Football (GRF) [29], as shown in Fig. 3a. The game requires balancing short-term control tasks such as passing, dribbling, and shooting with long-term strategic planning. To evaluate the MARL-BT framework, we divided the football ground into three zones: Backfield, Midfield, and Front-field, with different tasks assigned to each area. For example, defense tasks, organizing tasks, and attack tasks frequently occur when the ball is in the Backfield, Midfield, and Front-field, respectively, as shown in Fig. 3b. Each agent in the game has a discrete action space of dimension 19, including moving in eight directions, sliding, shooting, and other actions. The observation contains information about the positions and movement directions of blue agents, red agents, the ball, and other elements.

We take the most votes code *gfootball-with-memory-patterns*[1] in the kaggle GRF game competition [30], and reform it as a BT which is used as the baseline. We name it as *Baseline-BT* which is used subsequently. Figure 4 shows the structure of the Baseline-BT used in our experiments. decomposes the full game task into three independent sub-tasks based on the ball's position on the field: Backfield, Midfield, and Front-field. In Backfield sub-task, four sub-tasks are further decomposed, which corresponding to the players' defensive positions on the penalty, left-side, middle and right-side of field. In addition, The penalty-area defense sub-task is given the highest priority, and as a result, players from other tasks may be reassigned to this sub-task. The Midfield and Front-field sub-tasks include emergency marking tasks to prevent ball interception and perform counterattacks. Moreover, based on ball ownership, the Midfield sub-task is decomposed into an organizing task and a defensive task, while the Front-field sub-task is decomposed into an attacking task and a defensive task. Then we embed different MARL algorithms, i.e., VDN and QMIX, to the BT with the proposed framework, which is named as *VDN-BT* and *QMIX-BT*, respectively, as indicated by the red dashed box. Finally, we use the action mask technique for the two methods, and name them as *MASK-VDN-BT* and *MASK-QMIX-BT*, respectively.

All the experiments are conducted on a computer with i7-11700F CPU, RTX3060, and 32 G RAM. We set the discount factor $\gamma = 0.99$. The optimization is conducted using Adam with a learning rate 5×10^{-4}. ε is set to 0.1.

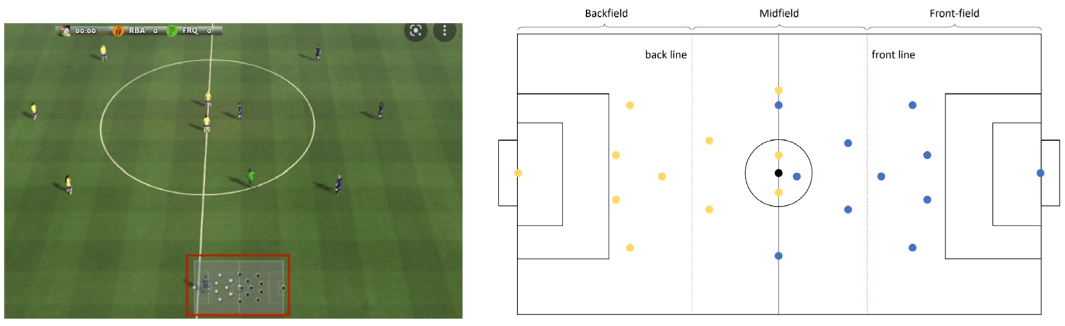
[1] https://www.kaggle.com/code/yegorbiryukov/gfootball-with-memory-patterns.

Comparison of the training performance

In this section, we compare the MARL learning performance with the baseline for different scenarios. We take two popular MARL methods, i.e., VDN and QMIX, to replace respective parts of BTs. For the reward design of the attack sub-task in Front-field, we take the reward setting as in CHECKPOINT provided by GRF Engine. For the organize task in Midfield, we give +1 when the ball is near to the front line while -1 when the ball is near to the back line. For the middle sub-task in Backfield, we give -1 when the ball is kicked into the goal while +1 when the ball is near to the back line.

The training curves are shown in Fig. 5, in which the vertical axis represents the sum of rewards over an episode and the horizontal axis represents the training episodes. For all curves, we can see that the reward increases gradually with the increasing number of episodes. At the beginning, the reward of Baseline-BT is better than VDN-BT and QMIX-BT, due to the learning algorithm needs time to explore the unknown environment. However, the performance of VDN-BT and QMIX-BT gradually improves as training progresses, and ultimately outperform Baseline-BT. Figure 5a, b show the performance of VDN and QMIX for the middle task in Backfield. After about 0.5×10^4 episodes, the reward of VDN-BT starts to be better than Baseline-BT, and after about 1.4×10^5 episodes, the reward value tends to converge. Compared to Baseline-BT, VDN-BT improved the reward by 1.4 times, while QMIX-BT improved it by 2.6 times. Figure 5c, d show the performance of the two MARL methods for the organize sub-task in Midfield. The two MARL methods improve the reward by 1.7 times and 2.4 times compared to Baseline-BT. For the attack sub-task in Front-field, i.e., Fig. 5e, f, the reward is improved by 0.3 times and 1.3 times, respectively. This shows that MARL methods could effectively improve the performance of BT in games.

For the masking mechanism, the performance of MARL methods is improved significantly in all cases. In Fig. 5a, b, the reward of VDN-BT is improved by 2 times with MASK-VDN-BT, while that of QMIX-BT is improved around 1.2 times with MASK-QMIX-BT. The performance improvements for Fig. 5c, d are 0.6 times and 0.8 times, respectively. For the attack sub-task in Fig. 5e, f, the masking mechanism improves the performance of VDN and QMIX by 0.3 times and 0.2 times, respectively. Another important point to note is that the reward growth of MASK-VDN-BT and MASK-QMIX-BT is more significant in Backfield and Front-field than that in Midfield. This is because in Backfield, the opponent often intercepts the ball, activating the assisting task more frequently. Similarly, for the attack sub-task in Front-field, the opponent frequently intercepts the ball, leading to a higher frequency of the marking task. Overall, these results highlight the importance of carefully evaluating different



(a) Multi-agent Full Game Scenarios. (b) Visualization of the Super Mini Map.

Fig. 3 **a** A snapshot of the *11_vs_11_competition* football game. **b** Initial positions of agents. Yellow points represent our agents while blue represent opponents. The black point represents the ball

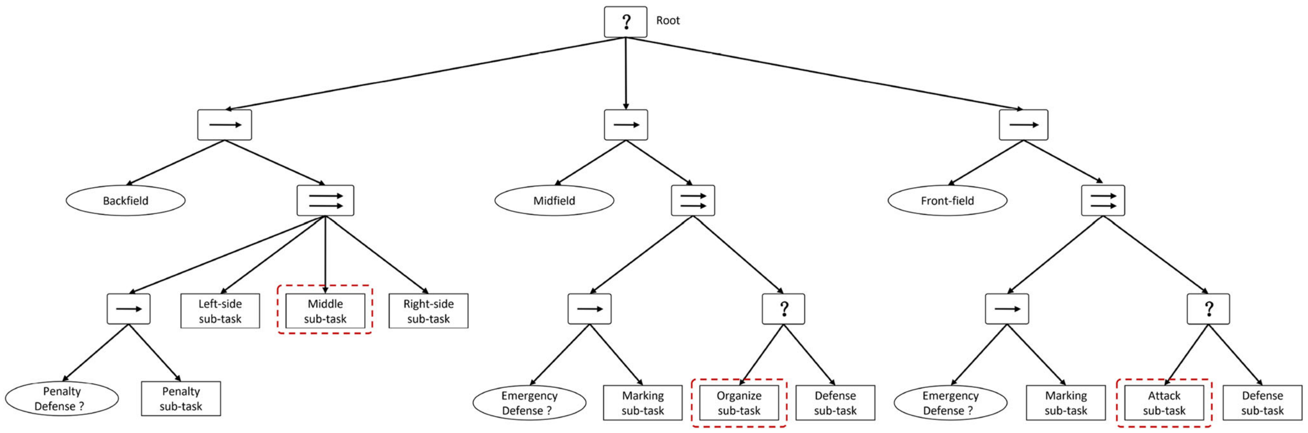


Fig. 4 The structure of Baseline-BT in our experiments

Table 1 The possession, win and loss rate of MARL methods

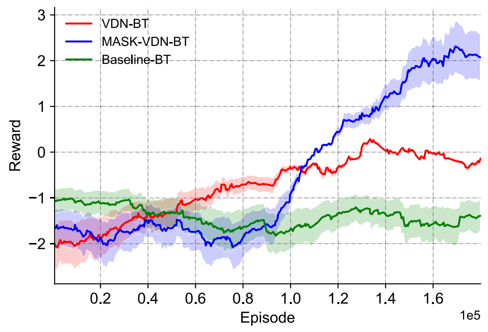
Ground zone	Target (%)	Baseline-BT (%)	VDN-BT (%)	MASK-VDN-BT (%)	QMIX-BT (%)	MASK-QMIX-BT (%)
Backfield	Possession rate	36.121	38.154	39.124	41.431	42.658
	Win rate	55.031	55.123	56.221	56.43	56.987
	Loss rate	**34.237**	**24.633**	**23.534**	**22.102**	**20.592**
Midfield	Possession rate	**36.121**	**41.244**	**42.512**	**42.824**	**43.641**
	Win rate	55.031	56.09	56.383	57.154	58.384
	Loss rate	34.237	32.832	32.432	31.034	30.591
Front-field	Possession rate	36.121	39.951	40.316	42.144	42.820
	Win rate	**55.031**	**59.031**	**60.970**	**62.20**	**63.940**
	Loss rate	34.237	32.792	31.114	30.166	29.236

learning algorithms embedding different scenarios, and considering masking mechanisms to improve performance.

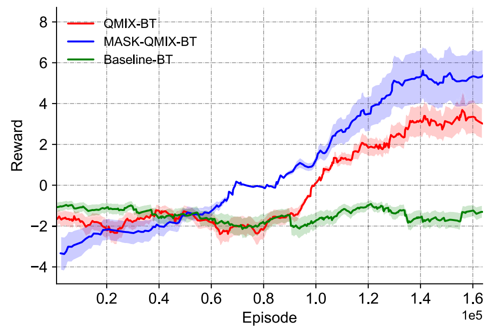
Comparison of the evaluating performance

We further investigate the impact of VDN-BT, QMIX-BT, MASK-VDN-BT, and MASK-QMIX-BT on the overall performance of the full game, we conducted a comprehensive evaluation using three key criteria: possession rate, win rate, and loss rate. The possession rate reflects the number of times that our agents were able to control of the ball, while the win rate and loss rate indicate the number of matches that our agents won and lost, respectively. By considering these three criteria, we are able to obtain a holistic understanding of the effectiveness of the MARL-BT.

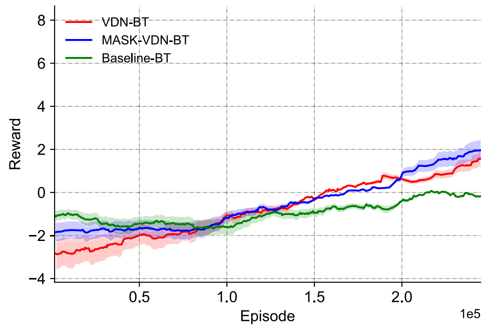
The results for all methods under three tasks are shown in Table 1, where bold values identify the data with the most significant changes in evaluation criteria in Compared to



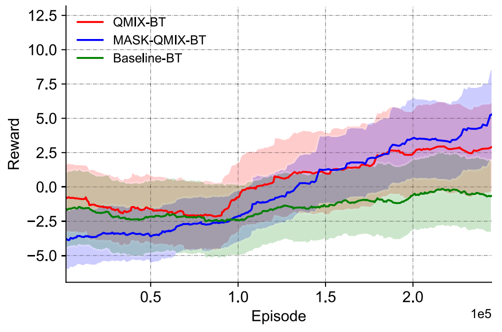
(a) VDN for Backfield.



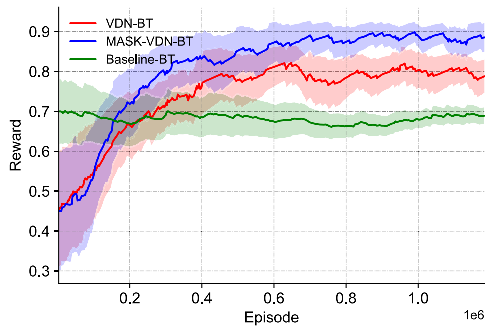
(b) QMIX for Backfield.



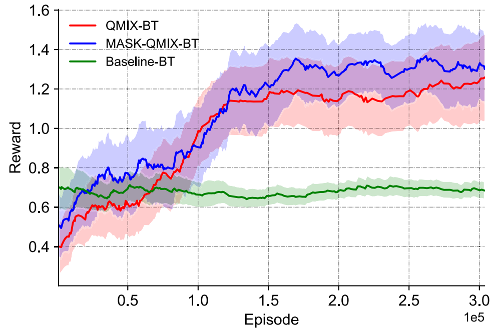
(c) VDN for Midfield.



(d) QMIX for Midfield.



(e) VDN for Front-field.

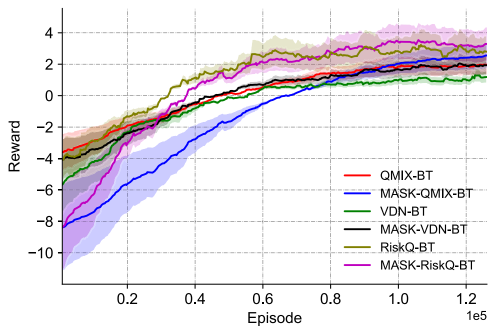


(f) QMIX for Front-field.

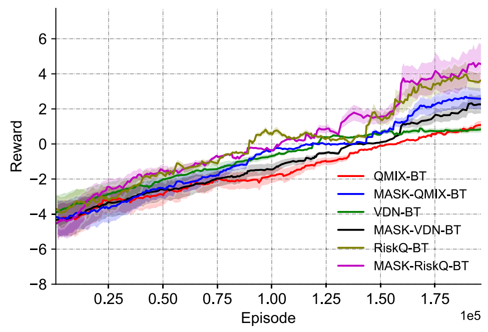
Fig. 5 Comparisons of training performance for different sub-tasks

Baseline-BT, we found that VDN-BT improves the possession rate and win rate by around 3.7% and 1.7% on average, respectively. The QMIX-BT improves that by around 5.9% and 3.6%. For the loss rate, VDN-BT and QMIX-BT decrease 4.2% and 6.5% on average. Further, the masking mechanism brings additional improvements, i.e., MASK-VDN-BT gets 4.5% and 2.8% improvements in the possession rate compared to Baseline-BT, while MASK-QMIX-BT obtains around 6.7% and 4.6% improvements. This further verifies that the masking mechanism can successfully deal with unexpected interruptions in the training procedure of MARL-BT.

More specifically, for the middle sub-task in Backfield, the loss rate is reduced by 12.7% and 13.6% while the win rate is increased only by 3% and 6% with MASK-VDN-BT and MASK-QMIX-BT. The reason is that the main responsibility for players in Backfield is to prevent the opponent from scoring goals. Similarly, for the organize sub-task in Midfield, the most significant improvement is the ball possession rate. MASK-VDN-BT and MASK-QMIX-BT obtain around 6.3% and 7.5% improvements in the ball possession rate compared to Baseline-BT. The win rate is increased only 1.4% and 3.3%, The loss rate is decreased by 1.8% and 3.5%.



(a) 3 players.



(b) 5 players

Fig. 6 The training curves with different numbers of players

For the attack sub-task in Front-field, we clearly find that the win rate is improved most significantly. This illustrates that introducing MARL methods with the masking mechanism can significantly improve the performance of BT. MASK-VDN-BT and MASK-QMIX-BT improve the win rate by around 5.8% and 8.9%, respectively, which is bigger than that for the loss rate and for the ball possession rate.

Ablation study

In this section, we conduct experiments to examine the number of players controlled by the MARL method and the impact of different performance MARL algorithms on the learning performance of the MARL-BT framework. We consider the organize task in the Midfield, in which the MARL method controls 3 players and 5 players, respectively. The results for embedding different MARL methods into BTs with and without the action masking technique are shown in Fig. 6. First, the MARL method and the action masking technique could improve the performance of BTs no matter how the number of players controlled by the MARL method changes. Second, we observe a positive correlation between the improvement in learning performance of the MARL-BT framework and algorithmic performance. In the experiments, the latest RiskQ [31] is embedded into BTs, i.e., RiskQ-BT. RiskQ-BT exhibits superior performance compared to QMIX-BT and VDN-BT, with the masked mechanism of MASK-RiskQ-BT further enhancing the performance of RiskQ-BT.

Conclusion

In this paper, we propose the framework MARL-BT, which combines BTs with MARL methods. It inherits the ability of BTs for decomposing complex tasks to sub-tasks, and good performance of learning-based methods on well-defined small problems. Different from previous works which construct a separate sub-scenario, or train over the whole game, we present a procedure that the MARL method is trained following the running mechanism of BTs, which works only as a segment over the whole game. Meanwhile, we point out a special phenomenon which is the unexpected interruption, that exists in MARL-BT. It happens between the learning-based sub-task with lower priority and the rule-based sub-task with higher priority. We propose a action masking technique to remove the effects of unexpected interruptions for the learning of the MARL method. We conduct experiments on the GRF game, and the results show that the performance of BTs are significantly improved by the proposed framework, i.e., getting an 11.507% improvement for certain scenarios. The action masking technique could greatly improve the performance of the learning method, i.e., the final reward is improved around 100% times for a sub-task. We hope that our approach could provide valuable guidance for combining BTs with MARL to solve real-world large-scale problems. As the future work, it would be interesting to study more sophisticated method other than the masking mechanism to deal with unexpected interruptions.

Acknowledgements This work was financially supported by the National Natural Science Foundation of China Youth Science Foundation under Grants Nos. 61902425, 62102444, 62102446.

Data availability All data generated or analyses during this study are included in this published article.

Declarations

Conflict of interest The authors have no relevant financial or non-financial interests to disclose.

References

1. Weber BG, Mateas M, Jhala A (2011) Building human-level AI for real-time strategy games. In: 2011 AAAI Fall symposium series
2. Robertson G, Watson I (2015) Building behavior trees from observations in real-time strategy games. In: 2015 International symposium on innovations in intelligent systems and applications (INISTA), pp 1–7. https://doi.org/10.1109/INISTA.2015.7276774
3. Goudarzi H, Hine D, Richards A (2019) Mission automation for drone inspection in congested environments. In: 2019 Workshop on research, education and development of unmanned aerial systems (RED UAS). IEEE, pp 305–314
4. Olsson M (2016) Behavior trees for decision-making in autonomous driving. https://api.semanticscholar.org/CorpusID:112621565
5. Kuckling J, Ligot A, Bozhinoski D, Birattari M (2018) Behavior trees as a control architecture in the automatic modular design of robot swarms. In: International conference on swarm intelligence. Springer, pp 30–43
6. Sprague CI, Özkahraman Ö, Munafò A, Marlow R, Phillips AB, Ögren P (2018) Improving the modularity of AUV control systems using behaviour trees. In: 2018 IEEE/OES autonomous underwater vehicle workshop (AUV), pp 1–6
7. Macenski S, Mart'in FJP, White R, Clavero JG (2020) The marathon 2: a navigation system. In: 2020 IEEE/RSJ international conference on intelligent robots and systems (IROS), pp 2718–2725
8. Zhang Q, Xu K, Jiao P, Yin Q (2018) Behavior modeling for autonomous agents based on modified evolving behavior trees. In: 2018 IEEE 7th data driven control and learning systems conference (DDCLS). IEEE, pp 1140–1145
9. Sagredo-Olivenza I, Gómez-Martín PP, Gómez-Martín MA, González-Calero PA (2017) Trained behavior trees: programming by demonstration to support AI game designers. IEEE Trans Games 11(1):5–14
10. Fu Y, Qin L, Yin Q (2016) A reinforcement learning behavior tree framework for game AI. In: 2016 International conference on economics, social science, arts, education and management engineering. Atlantis Press, pp 573–579
11. Dey R, Child C (2013) QL-BT: enhancing behaviour tree design and implementation with q-learning. In: 2013 IEEE conference on computational intelligence in games (CIG). IEEE, pp 1–8
12. Pereira RdP, Engel PM (2015) A framework for constrained and adaptive behavior-based agents. arXiv preprint arXiv:1506.02312
13. Kartasev M (2019) Integrating reinforcement learning into behavior trees by hierarchical composition
14. Zhang Q, Sun L, Jiao P, Yin Q (2017) Combining behavior trees with maxq learning to facilitate cgfs behavior modeling. In: 2017 4th International conference on systems and informatics (ICSAI). IEEE, pp 525–531
15. Lowe R, Wu YI, Tamar A, Harb J, Pieter Abbeel O, Mordatch I (2017) Multi-agent actor-critic for mixed cooperative-competitive environments. Adv Neural Inf Process Syst 30:271
16. Rashid T, Samvelyan M, Schroeder C, Farquhar G, Foerster J, Whiteson S (2018) Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In: International conference on machine learning. PMLR, pp 4295–4304
17. Yu C, Velu A, Vinitsky E, Wang Y, Bayen AM, Wu Y (2021) The surprising effectiveness of MAPPO in cooperative, multi-agent games. CoRR. arXiv:2103.01955
18. Zhao J, Zhao Y, Wang W, Yang M, Hu X, Zhou W, Hao J, Li H (2022) Coach-assisted multi-agent reinforcement learning framework for unexpected crashed agents. arXiv preprint arXiv:2203.08454
19. Wen M, Kuba JG, Lin R, Zhang W, Wen Y, Wang J, Yang Y (2022) Multi-agent reinforcement learning is a sequence modeling problem. arXiv preprint arXiv:2205.14953
20. Li L, Wang L, Li Y, Sheng J (2021) Mixed deep reinforcement learning-behavior tree for intelligent agents design. ICAART 1:113–124
21. Isla D (2005) GDC 2005 proceeding: handling complexity in the halo 2 AI. Retrieved Oct 21, 2009
22. Iovino M, Scukins E, Styrud J, Ögren P, Smith C (2022) A survey of behavior trees in robotics and AI. Robot Auton Syst 154:104096
23. Yang Y, Hao J, Liao B, Shao K, Chen G, Liu W, Tang H (2020) Qatten: a general framework for cooperative multiagent reinforcement learning. arXiv preprint arXiv:2002.03939
24. Sunehag P, Lever G, Gruslys A, Czarnecki WM, Zambaldi V, Jaderberg M, Lanctot M, Sonnerat N, Leibo JZ, Tuyls K et al (2017) Value-decomposition networks for cooperative multi-agent learning. arXiv preprint arXiv:1706.05296
25. Son K, Kim D, Kang WJ, Hostallero DE, Yi Y (2019) QTRAN: learning to factorize with transformation for cooperative multi-agent reinforcement learning. In: International conference on machine learning. PMLR, pp 5887–5896
26. Tomai E, Salazar R, Flores R (2013) Simulating aggregate player behavior with learning behavior trees. In: Proceedings of the 22nd annual conference on behavior representation in modeling and simulation
27. Zhu X (2019) Behavior tree design of intelligent behavior of non-player character (NPC) based on unity3d. J Intell Fuzzy Syst 37(5):6071–6079
28. Mnih V, Kavukcuoglu K, Silver D, Graves A, Antonoglou I, Wierstra D, Riedmiller M (2013) Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602
29. Kurach K, Raichuk A, Stańczyk P, Zając M, Bachem O, Espeholt L, Riquelme C, Vincent D, Michalski M, Bousquet O et al (2020) Google research football: a novel reinforcement learning environment. In: Proceedings of the AAAI conference on artificial intelligence, vol. 34, pp 4501–4510
30. Google Research (2020). https://www.kaggle.com/competitions/google-football/code
31. Shen S, Ma C, Li C, Liu W, Fu Y, Mei S, Liu X, Wang C (2023) RiskQ: risk-sensitive multi-agent reinforcement learning value factorization