A Related Work

MARL has been used for solving multi-robot problems Alon and Zhou [2020], Khan et al. [2019], Mitchell et al. [2020], Park et al. [2020], Stone et al. [2005], Strickland et al. [2019], Tang [2019] and hierarchy has also been introduced into multi-robot scenarios Chen [2014], Luo et al. [2018], Oliehoek and Visser [2006], Omidshafiei et al. [2017b], Zhang et al. [2017], but hierarchical MARL is still novel for multi-robot systems Nachum et al. [2019], Wang et al. [2020b], Wu et al. [2021a], Xiao et al. [2019].

One line of hierarchical MARL is still focusing on learning primitive-action-based policy for each agent, while leveraging a hierarchical structure to achieve knowledge transfer Yang et al. [2021], credit assignment Ahilan and Dayan [2019] and low-level policy factorization over agents Vezhnevets et al. [2020]. In these works, as the decision-making over agents is still limited at a single low-level, none of them has been evaluated in large-scale realistic domains. Instead, by having macro-actions, our methods equip agents with the potential capability of exploiting abstracted skills, sub-task allocation and problem decomposition via hierarchical decision-making, which is critical for scaling up to real-world multi-robot tasks.

Another line of the research allow agents to learn both a high-level policy and a low-level policy, but the methods either force agents to perform a high-level choice at every time step de Witt et al. [2019], Han et al. [2019] or require all agents' high-level decisions have the same time duration Nachum et al. [2019], Wang et al. [2020b, 2021b], Xu et al. [2021], Yang et al. [2020b], where agents are actually synchronized at both levels. In contrast, our frameworks are more general and applicable to real-world multi-robot systems because they allow agents to asynchronously execute at a high-level without synchronization or waiting for all agents to terminate.

Recently, some asynchronous hierarchical approaches have been developed. Xiao et al. [2019] and Wu et al. [2021a] extend DQN Mnih et al. [2015] to learn macro-action-value functions and spatial-action-value maps for agents respectively. Our work, however, focuses on policy gradient algorithms that have different theoretical properties than value-based approaches (e.g., our methods are more scalable in the action space). Both classes of methods can co-exit and fit well with different sets of tasks. Menda et al. [2019] frame multi-agent asynchronous decision-making problems as event-driven processes with one assumption on the acceptable of losing the ability to capture low-level interaction between agents within an event duration and the other on homogeneous agents, but our frameworks rely on the time-driven simulator used for general multi-agent and single-agent RL problems and do not have the above assumptions. Chakravorty et al. [2019] adapt a single-agent *option-critic* framework Bacon et al. [2017] to multi-agent domains to learn all components (e.g., low-level policy, high-level abstraction, high-level policy) from scratch, but learning at both levels is difficult and the proposed method does not perform well even in small TeamGrid Maxime and Julien [2020] scenarios. More important to note is that none of the existing works provides a principled way for directly optimizing parameterized macro-action-based policies via asynchronous policy gradients to solve general multi-agent problems with macro-actions, and our work in this paper seeks to fill this gap.

B Macro-Action-Based Policy Gradient Theorem

As POMDPs can always be transformed to history-based MDPs, we can directly adapt the general Bellman equation for the state values of a hierarchical policy [Sutton et al., 1999] to a macro-action-based POMDP by replacing the state s with a history h as follows (for keeping the notaion simple, we use τ to represent the number of timesteps taken by the corresponding macro-action m, and we use h to represent macro-observation-action history):

$$V^{\Psi}(h) = \sum_{m} \Psi(m|h) Q^{\Psi}(h,m) \tag{8}$$

$$Q^{\Psi}(h,m) = r^{c}(h,m) + \sum_{h'} P(h'|h,m) V^{\Psi}(h') \tag{9}$$

where,

$$r^{c}(h,m) = \mathbb{E}_{\tau \sim \beta_m, s_{t_m}|h}\Big[\sum_{t=t_m}^{t_m+\tau-1} \gamma^{t} r_t\Big] \tag{10}$$

$$P(h'|h,m) = P(z'|h,m) = \sum_{\tau=1}^{\infty} \gamma^{\tau} P(z',\tau|h,m) \tag{11}$$

$$= \sum_{\tau=1}^{\infty} \gamma^{\tau} P(\tau|h,m) P(z'|h,m,\tau) \tag{12}$$

$$= \sum_{\tau=1}^{\infty} \gamma^{\tau} P(\tau|h,m) P(z'|h,m,\tau) \tag{13}$$

$$= \mathbb{E}_{\tau \sim \beta_m}\Big[\gamma^{\tau} \mathbb{E}_{s|h}\big[\mathbb{E}_{s'|s,m,\tau}[P(z'|m,s')]\big]\Big] \tag{14}$$

Next, we follow the proof of the policy gradient theorem [Sutton et al., 2000]:

$$\nabla_{\theta} V^{\Psi_\theta}(h) = \nabla_{\theta}\left[\sum_{m} \Psi_{\theta}(m|h) Q^{\Psi_\theta}(h,m)\right] \tag{15}$$

$$= \sum_{m}\Big[\nabla_{\theta}\Psi_{\theta}(m|h) Q^{\Psi_\theta}(h,m) + \Psi_{\theta}(m|h)\nabla_{\theta} Q^{\Psi_\theta}(h,m)\Big] \tag{16}$$

$$= \sum_{m}\Big[\nabla_{\theta}\Psi_{\theta}(m|h) Q^{\Psi_\theta}(h,m) + \Psi_{\theta}(m|h)\nabla_{\theta}\big(r^{c}(h,m) + \sum_{h'} P(h'|h,m) V^{\Psi_\theta}(h')\big)\Big] \tag{17}$$

$$= \sum_{m}\Big[\nabla_{\theta}\Psi_{\theta}(m|h) Q^{\Psi_\theta}(h,m) + \Psi_{\theta}(m|h)\sum_{h'} P(h'|h,m)\nabla_{\theta} V^{\Psi_\theta}(h')\big)\Big] \tag{18}$$

$$= \sum_{\hat{h}\in H}\sum_{k=0}^{\infty} P(h \to \hat{h}, k, \Psi_\theta) \sum_{m} \nabla_{\theta}\Psi_{\theta}(m|\hat{h}) Q^{\Psi_\theta}(\hat{h},m) \quad \text{(after repeated unrolling)} \tag{19}$$

Then, we can have:

$$\nabla_{\theta} J(\theta) = \nabla_{\theta} V^{\Psi_\theta}(h_0) \tag{20}$$

$$= \sum_{h\in H}\sum_{k=0}^{\infty} P(h_0 \to h, k, \Psi_\theta) \sum_{m} \nabla_{\theta}\Psi_{\theta}(m|h) Q^{\Psi_\theta}(h,m) \tag{21}$$

$$= \sum_{h} \rho^{\Psi_\theta}(h) \sum_{m} \nabla_{\theta}\Psi_{\theta}(m|h) Q^{\Psi_\theta}(h,m) \tag{22}$$

$$= \sum_{h} \rho^{\Psi_\theta}(h) \sum_{m} \Psi_{\theta}(m|h)\nabla_{\theta}\log\Psi_{\theta}(m|h) Q^{\Psi_\theta}(h,m) \tag{23}$$

$$= \mathbb{E}_{h\sim\rho^{\Psi_\theta}, m\sim\Psi_\theta}\Big[\nabla_{\theta}\log\Psi_{\theta}(m|h) Q^{\Psi_\theta}(h,m)\Big] \tag{24}$$

C Asynchronous Acotr-Critic Algorithms

In this section, we present the pesudo code of each proposed macro-action-based actor-critic algorithm with an example to show how the sequential experiences are squeezed for training the critic and the actor. We describe all methods in the on-policy learning manner while off-policy learning can be achieved by applying importance sampling weights and not resetting the buffer.

Macro-Action-Based Independent Actor-Critic (Mac-IAC):

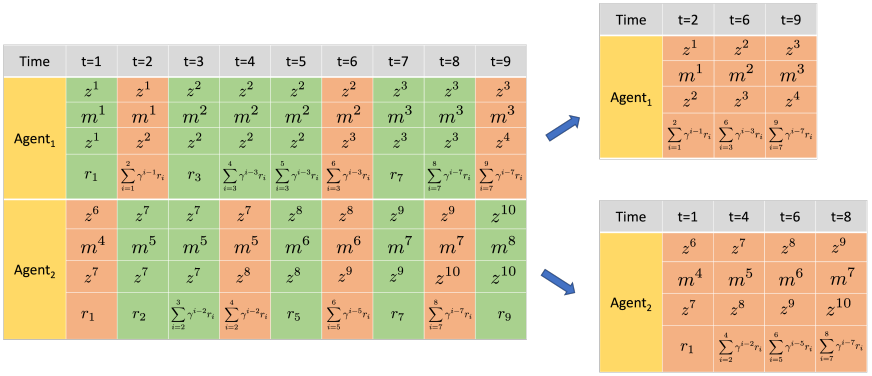


Figure 6: An example of the trajectory squeezing process in Mac-IAC. We collect each agent's high-level transition tuple at every primitive-step. Each agent is allowed to obtain a new macro-observation if and only if the current macro-action terminates, otherwise, the next macro-observation is set as same as the previous one. Each agent separately squeezes its sequential experiences by picking out the transitions when its macro-action terminates (red cells). Each agent independently train the critic and the policy using the squeezed trajectory.

Algorithm 1 Mac-IAC

Initialize a decentralized policy network for each agent i: Ψ_{θ_i}
Initialize decentralized critic networks for each agent i: $V_{\mathbf{w}_i}^{\Psi_{\theta_i}}$, $V_{\mathbf{w}_i^-}^{\Psi_{\theta_i}}$
Initialize a buffer $\mathcal{D}$
for *episode* = 1 to M **do**
    $t = 0$
    Reset env
    while not reaching a terminal state **and** $t < \mathbb{H}$ **do**
        $t \leftarrow t + 1$
        for each agent i **do**
            if the macro-action m_i is terminated **then**
                $m_i \sim \Psi_{\theta_i}(\cdot \mid h_i; \epsilon)$
            else
                Continue running current macro-action m_i
        for each agent i **do**
            Get cumulative reward r_i^c, next macro-observation z_i'
            Collect $\langle z_i, m_i, z_i', r_i^c \rangle$ into the buffer $\mathcal{D}$
    if *episode* mod $I_{\text{train}} = 0$ **then**
        for each agent i **do**
            Squeeze agent i's trajectories in the buffer $\mathcal{D}$
            Perform a gradient decent step on $L(\mathbf{w}_i) = \big(y - V_{\mathbf{w}_i}^{\Psi_{\theta_i}}(h_i)\big)_{\mathcal{D}}^2$, where $y = r_i^c + \gamma^{\tau_{m_i}} V_{\mathbf{w}_i^-}^{\Psi_{\theta_i}}(h_i')$
            Perform a gradient ascent on:
            $\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\Psi}, \vec{\theta}}\Big[\nabla_{\theta_i} \log \Psi_{\theta_i}(m_i|h_i)\big(r_i^c + \gamma^{\tau_{m_i}} V_{\mathbf{w}_i^-}^{\Psi_{\theta_i}}(h_i') - V_{\mathbf{w}_i}^{\Psi_{\theta_i}}(h_i)\big)\Big]$
        Reset buffer $\mathcal{D}$
    if *episode* mod $I_{\text{TargetUpdate}} = 0$ **then**
        for each agent i **do**
            Update the critic target network $\mathbf{w}_i^- \leftarrow \mathbf{w}_i$

Macro-Action-Based Centralized Actor-Critic (Mac-CAC):

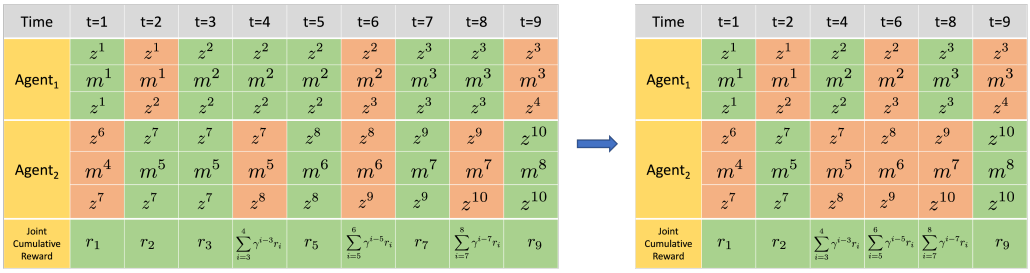


Figure 7: An example of the trajectory squeezing process in Mac-CAC. Joint sequential experiences are squeezed by picking out joint transition tuples when the joint macro-action terminates, in that, any agent's macro-action termination (marked in red) ends the joint macro-action at the timestep. For example, at $t = 1$, agents execute a joint macro-action $\vec{m} = \langle m^1, m^4 \rangle$ for one timestep; at $t = 2$, the joint macro-action becomes $\langle m^1, m^5 \rangle$ as Agent_2 finished m^4 at last step and chooses a new macro-action m^5; Agent_1 finished its macro-action m_1 at $t = 2$ and selects a new macro-action m^2 at $t = 3$ so that the joint macro-action switches to $\langle m^2, m^5 \rangle$ which keeps running until the 4th timestep. Therefore, the first two joint macro-actions have two single-step reward respectively, and reward of joint macro-action $\langle m^2, m^5 \rangle$ is an accumulative reward over two consecutive timesteps.

Algorithm 2 Mac-CAC

Initialize a centralized policy network: Ψ_θ
Initialize centralized critic networks: $V_{\mathbf{w}}^{\Psi_\theta}$, $V_{\mathbf{w}^-}^{\Psi_\theta}$
Initialize a centralized buffer $\mathcal{D} \leftarrow$ Mac-JERTs,
for *episode* = 1 to M **do**
    $t = 0$
    Reset env
    while not reaching a terminal state **and** $t < \mathbb{H}$ **do**
        $t \leftarrow t + 1$
        if the joint macro-action $\vec{m}$ is terminated **then**
            $\vec{m} \sim \Psi_\theta(\cdot \mid \vec{h}, \vec{m}^{\text{undone}}; \epsilon)$
        else
            Continue running current joint macro-action $\vec{m}$
        Get a joint cumulative reward $\vec{r}^c$, next joint macro-observation $\vec{z}'$
        Collect $\langle \vec{z}, \vec{m}, \vec{z}', \vec{r}^c \rangle$ into the buffer $\mathcal{D}$
    if *episode* mod $I_{\text{train}} = 0$ **then**
        Squeeze joint macro-level trajectories in the buffer $\mathcal{D}$ according to joint macro-action terminations
        Perform a gradient decent step on $L(\mathbf{w}) = \big(y - V_{\mathbf{w}}^{\Psi_\theta}(\vec{h})\big)^2_{\mathcal{D}}$, where $y = \vec{r}^c + \gamma^{\vec{\tau}_{\vec{m}}} V_{\mathbf{w}^-}^{\Psi_\theta}(\vec{h}')$
        Perform a gradient ascent on $\nabla_\theta J(\theta) = \mathbb{E}_{\Psi_\theta}\Big[\nabla_\theta \log \Psi_\theta(\vec{m} \mid \vec{h})\big(\vec{r}^c + \gamma^{\vec{\tau}_{\vec{m}}} V_{\mathbf{w}^-}^{\Psi_\theta}(\vec{h}') - V_{\mathbf{w}}^{\Psi_\theta}(\vec{h})\big)\Big]$
        Reset buffer $\mathcal{D}$
    if *episode* mod $I_{\text{TargetUpdate}} = 0$ **then**
        Update the critic target network $\mathbf{w}^- \leftarrow \mathbf{w}$

where, $\vec{m}^{\text{undone}}$ is the sub-joint-macro-action over the agents who have not terminated their macro-actions and will continue running.

Naive Mac-IACC:

In the pseudo code of Naive Mac-IACC presented below, we assume the accessible centralized information $\mathbf{x}$ is joint macro-observation-action history in the centralized critic.

Figure 8: An example of the trajectory squeezing process in Navie Mac-IACC.The joint trajectory is first squeezed depending on joint macro-action termination for training the centralized critic (line 18-19 in Algorithm 3). Then, the trajectory is further squeezed for each agent depending on each agent's own macro-action termination for training the decentralized policy (line 20-23 in Algorithm 3).

Algorithm 3 Naive Mac-IACC

1: Initialize a decentralized policy network for each agent i: Ψ_{θ_i}
2: Initialize centralized critic networks: $V_{\mathbf{w}}^{\vec{\Psi}_{\vec{\theta}}}$, $V_{\mathbf{w}^-}^{\vec{\Psi}_{\vec{\theta}}}$
3: Initialize a decentralized buffer $\mathcal{D} \leftarrow$ Mac-JERTs,
4: **for** *episode* = 1 to M **do**
5: $t = 0$
6: Reset env
7: **while** not reaching a terminal state **and** $t < \mathbb{H}$ **do**
8: $t \leftarrow t + 1$
9: **for** each agent i **do**
10: **if** the macro-action m_i is terminated **then**
11: $m_i \sim \Psi_{\theta_i}(\cdot \mid h_i; \epsilon)$
12: **else**
13: Continue running current macro-action m_i
14: Get a reward $\vec{r}^c$ accumulated based on current joint macro-action termination
15: Get next joint macro-observations $\vec{z}'$
16: Collect $\langle \vec{z}, \vec{m}, \vec{z}', \vec{r}^c \rangle$ into the buffer $\mathcal{D}$
17: **if** *episode* mod $I_{\text{train}} = 0$ **then**
18: Squeeze joint macro-level trajectories in the buffer $\mathcal{D}$ according to joint macro-action terminations
19: Perform a gradient decent step on $L(\mathbf{w}) = \left(y - V_{\mathbf{w}}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h})\right)^2_{\mathcal{D}}$, where $y = \vec{r}^c + \gamma^{\vec{\tau}_{\vec{m}}} V_{\mathbf{w}^-}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h}')$
20: **for** each agent i **do**
21: Squeeze agent i's trajectories in the buffer $\mathcal{D}$ according to its own macro-action terminations
22: Perform a gradient ascent on:
23: $\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\Psi}_{\vec{\theta}}}\left[\nabla_{\theta_i} \log \Psi_{\theta_i}(m_i|h_i)\left(\vec{r}^c + \gamma^{\vec{\tau}_{\vec{m}}} V_{\mathbf{w}^-}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h}') - V_{\mathbf{w}}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h})\right)\right]$
24: Reset buffer $\mathcal{D}$
25: **if** *episode* mod $I_{\text{TargetUpdate}} = 0$ **then**
26: Update the critic target network $\mathbf{w}^- \leftarrow \mathbf{w}$

Macro-Action-Based Independent Actor with Individual Centralized Critic (Mac-IAICC):

In the pseudo code of Mac-IAICC presented below, we assume the accessible centralized information $\mathbf{x}$ is joint macro-observation-action history in the centralized critic.

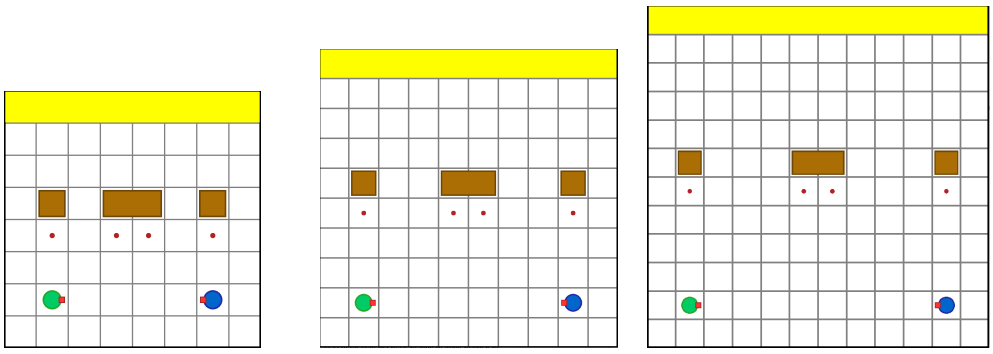
Figure 9: An example of the trajectory squeezing process in Mac-IAICC: each agent learns an individual centralized critic for the decentralized policy optimization. In order to achieve a better use of centralized information, the recurrent layer in each critic's neural network should receive all the valid joint macro-observation-action information (when any agent terminates its macro-action (line 20-22) and obtain a new joint macro-observation). However, the critic's TD updates and the policy's updates still rely on each agent's individual macro-action termination and the accumulative reward at the corresponding timestep (line 23-26). Hence, the trajectory squeezing process for training each critic still depends on joint-macro-action termination but only retaining the accumulative rewards w.r.t. the corresponding agent's macro-action termination for computing the TD loss (the middle part in the above picture). Then, each agent's trajectory is further squeezed depending on its macro-action termination to update the decentralized policy.

Algorithm 4 Mac-IAICC

Initialize a decentralized policy network for each agent i: Ψ_{θ_i}
Initialize centralized critic networks for each agent i: $V_{\mathbf{w}_i}^{\vec{\Psi}_{\vec{\theta}}}$, $V_{\mathbf{w}_i^-}^{\vec{\Psi}_{\vec{\theta}}}$
Initialize a decentralized buffer $\mathcal{D}$
for *episode* = 1 to M **do**
    $t = 0$
    Reset env
    while not reaching a terminal state **and** $t < \mathbb{H}$ **do**
        $t \leftarrow t + 1$
        for each agent i **do**
            if the macro-action m_i is terminated **then**
                $m_i \sim \Psi_{\theta_i}(\cdot \mid h_i; \epsilon)$
            else
                Continue running current macro-action m_i
        for each agent i **do**
            Get a reward r_i^c accumulated based on agent i's macro-action termination
        Get next joint macro-observations $\vec{z}'$
        Collect $\langle \vec{z}, \vec{m}, \vec{z}', \{r_1^c, \ldots, r_n^c\} \rangle$ into the buffer $\mathcal{D}$
    if *episode* mod $I_{\text{train}} = 0$ **then**
        for each agent i **do**
            Squeeze trajectories in the buffer $\mathcal{D}$ according to joint macro-action terminations
            Compute the TD-error of each timestep in the squeezed experiences:
            $L(\mathbf{w}_i) = \big(y - V_{\mathbf{w}_i}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h})\big)^2_{\mathcal{D}}$, where $y = r_i^c + \gamma^{\tau_{m_i}} V_{\mathbf{w}_i^-}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h}')$
            Perform a gradient descent only over the TD-errors when agent i's macro-action is terminated
            Squeeze agent i's trajectories in the buffer $\mathcal{D}$ according to its own macro-action terminations
            Perform a gradient ascent on:
            $\nabla_{\theta_i} J(\theta_i) = \mathbb{E}_{\vec{\Psi}_{\vec{\theta}}}\Big[\nabla_{\theta_i} \log \Psi_{\theta_i}(m_i|h_i)\big(r_i^c + \gamma^{\tau_{m_i}} V_{\mathbf{w}_i^-}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h}') - V_{\mathbf{w}_i}^{\vec{\Psi}_{\vec{\theta}}}(\vec{h})\big)\Big]$
        Reset buffer $\mathcal{D}$
    if *episode* mod $I_{\text{TargetUpdate}} = 0$ **then**
        for each agent i **do**
            Update the critic target network $\mathbf{w}_i^- \leftarrow \mathbf{w}_i$

D Domain Descriptions and Results

D.1 Box Pushing



(a) 8x8 (b) 10x10 (c) 12x12

Figure 10: Experimental environments.

Goal. The objective of the two robots is to learn collaboratively push the middle big box to the goal area at the top rather than pushing a small box on each own.

State. The global state information consists of the position and orientation of each robot and each box's position in a grid world.

Primitive-Action Space. *move forward*, *turn-left*, *turn-right* and *stay*.

Macro-Action Space.

• One-step macro-actions: ***Turn-left***, ***Turn-right***, and ***Stay***.

• Multi-step macro-actions: ***Move-to-small-box(i)*** that navigates the robot to the red spot below the corresponding small box and terminate with robot facing the box; ***Move-to-big-box(i)*** that navigates the robot to a red spot below the big box and terminate with robot facing the big box; ***Push*** that operates the robot to keep moving forward and terminate while arriving the world's boundary, touching the big box along or pushing a small box to the goal.

Observation Space. In both the primitive-observation and macro-observation, each robot is only allowed to capture one of five states of the cell in front of it: *empty*, *teammate*, *boundary*, *small box*, *big box*.

Dynamics. The transition in this task is deterministic. Boxes can only be moved towards the north when the robot faces the box and moves forward. The small box can be moved by a single robot while the big box require two robots to move it together.

Rewards. The team receives $+300$ for pushing big box to the goal area and $+20$ for pushing a small box to the goal area. A penalty -10 is issued when any robot hits the boundary or pushes the big box on its own.

Episode Termination. Each episode terminates when any box is pushed to the goal area, or when 100 timesteps has elapsed.

Results.

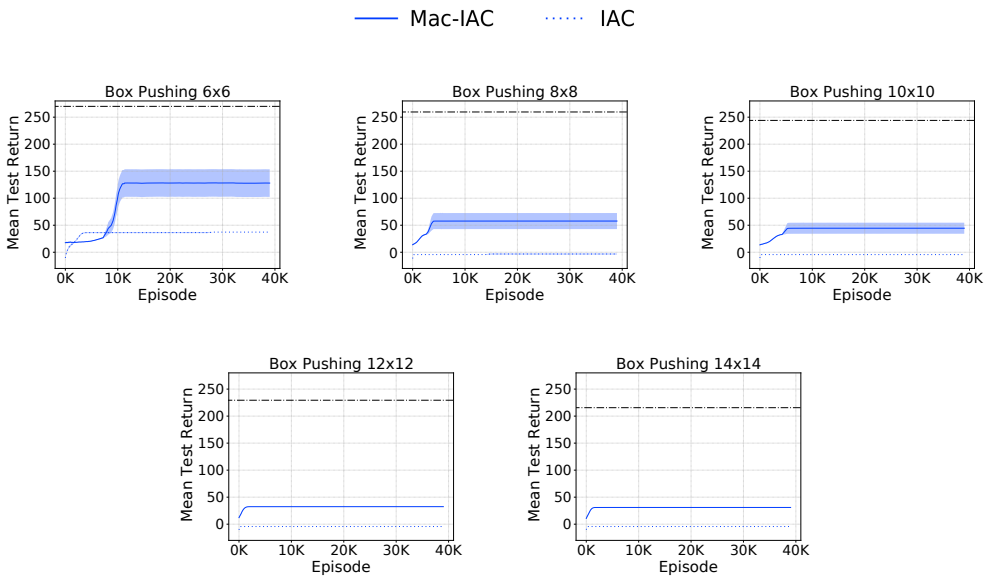


Figure 11: Decentralized learning with macro-actions vs primitive-actions.

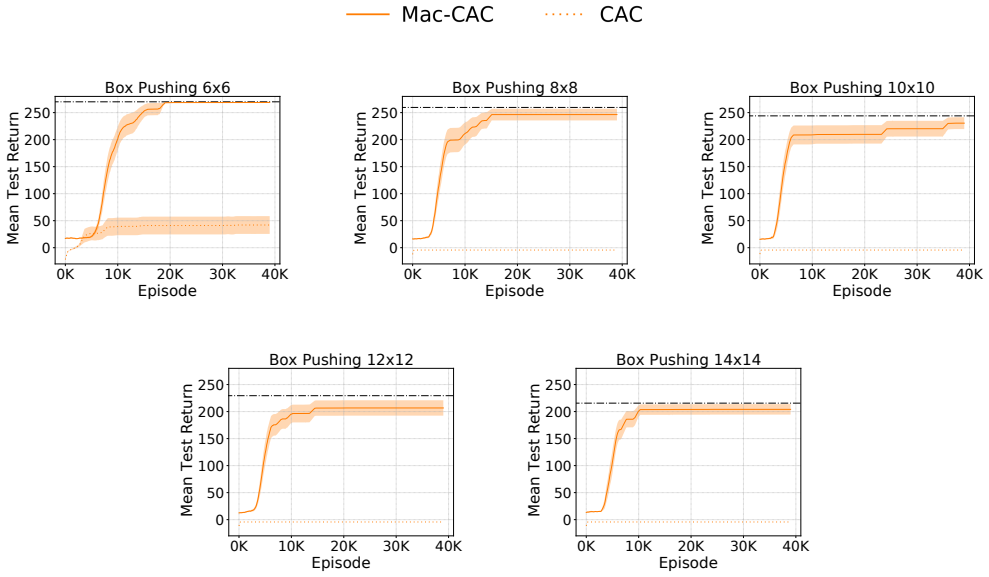


Figure 12: Centralized learning with macro-actions vs primitive-actions.

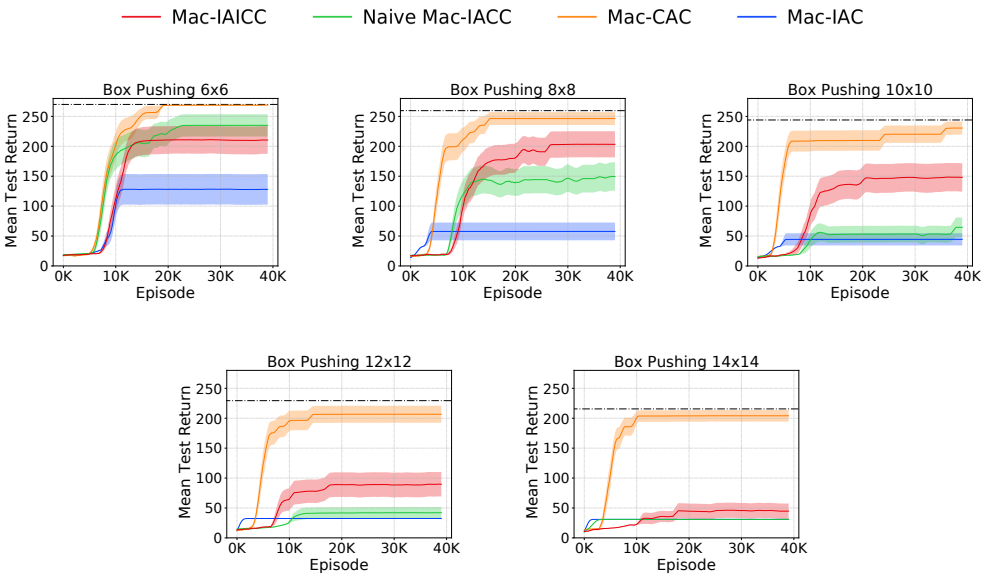


Figure 13: Comparison of macro-action-based multi-agent actor-critic methods.

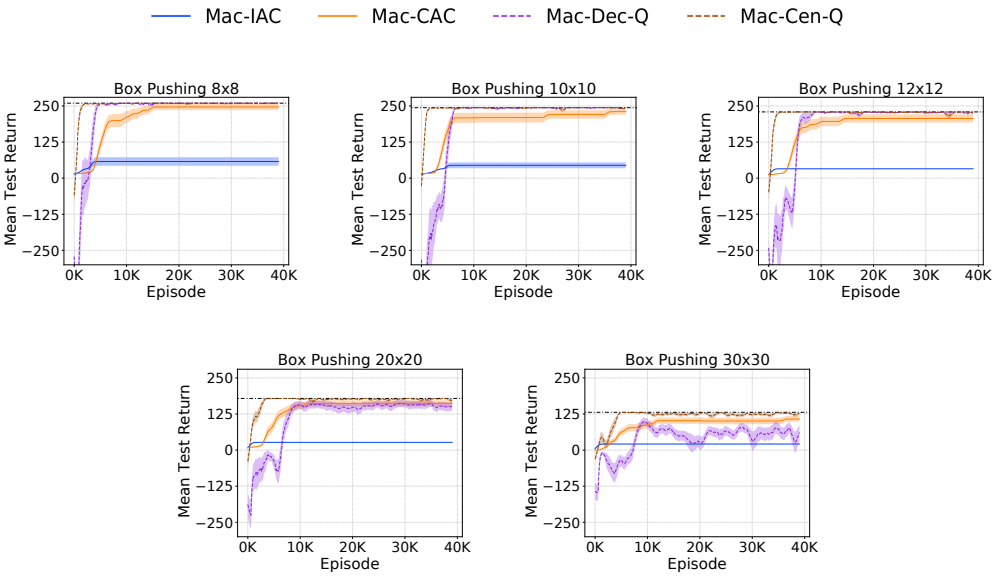
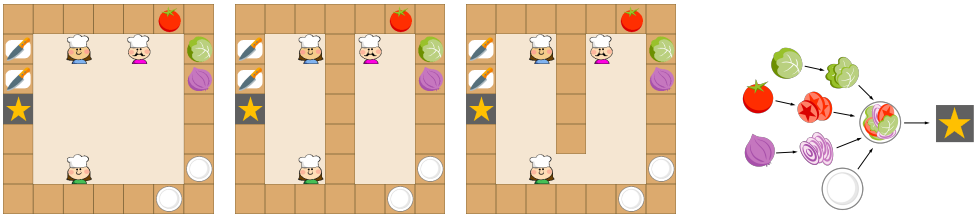


Figure 14: Comparison of macro-action-based actor-critic methods and value-based methods

D.2 Overcooked



(a) Overcooked-A (b) Overcooked-B (c) Overcooked-C (d) Lettuce-Onion-Tomato salad recipe

Figure 15: Experimental environments.

Goal. Three agents need to learn cooperating with each other to prepare a Tomato-Lettuce-Onion salad and deliver it to the 'star' counter cell as soon as possible. The challenge is that the recipe of making a tomato-lettuce-onion salad is unknown to agents. Agents have to learn the correct procedure in terms of picking up raw vegetables, chopping, and merging in a plate before delivering.

State Space. The environment is a 7×7 grid world involving three agents, one tomato, one lettuce, one onion, two plates, two cutting boards and one delivery cell. The global state information consists of the positions of each agent and above items, and the status of each vegetable: chopped, unchopped, or the progress under chopping.

Primitive-Action Space. Each agent has five primitive-actions: *up*, *down*, *left*, *right* and *stay*. Agents can move around and achieve picking, placing, chopping and delivering by standing next to the corresponding cell and moving against it (e.g., in Fig. 15a, the pink agent can *move right* and then *move up* to pick up the tomato).

Macro-Action Space. Here, we first describe the main function of each macro-action and then list the corresponding termination conditions.

- Five one-step macro-actions that are the same as the primitive ones;
- ***Chop***, cuts a raw vegetable into pieces (taking three time steps) when the agent stands next to a cutting board and an unchopped vegetable is on the board, otherwise it does nothing; and it terminates when:
  - The vegetable on the cutting board has been chopped into pieces;
  - The agent is not next to a cutting board;
  - There is no unchopped vegetable on the cutting board;
  - The agent holds something in hand.
- ***Get-Lettuce***, ***Get-Tomato***, and ***Get-Onion***, navigate the agent to the latest observed position of the vegetable, and pick the vegetable up if it is there; otherwise, the agent moves to check the initial position of the vegetable. The corresponding termination conditions are listed below:
  - The agent successfully picks up a chopped or unchopped vegetable;
  - The agent observes the target vegetable is held by another agent or itself;
  - The agent is holding something else in hand;
  - The agent's path to the vegetable is blocked by another agent;
  - The agent does not find the vegetable either at the latest observed location or the initial location;
  - The agent attempts to enter the same cell with another agent, but has a lower priority than another agent.
- ***Get-Plate-1/2***, navigates the agent to the latest observed position of the plate, and picks the vegetable up if it is there; otherwise, the agent moves to check the initial position of the vegetable. The corresponding termination conditions are listed below:
  - The agent successfully picks up a plate;
  - The agent observes the target plate is held by another agent or itself;

  - The agent is holding something else in hand;
  - The agent's path to the plate is blocked by another agent;
  - The agent does not find the plate either at the latest observed location or at the initial location;
  - The agent attempts to enter the same cell with another agent but has a lower priority than another agent.
- ***Go-Cut-Board-1/2***, navigates the agent to the corresponding cutting board with the following termination conditions:
  - The agent stops in front of the corresponding cutting board, and places an in-hand item on it if the cutting board is not occupied;
  - If any other agent is using the target cutting board, the agent stops next to the teammate;
  - The agent attempts to enter the same cell with another agent but has a lower priority than another agent.
- ***Go-Counter*** (only available in Overcook-B, Fig. 1c), navigates the agent to the center cell in the middle of the map when the cell is not occupied, otherwise it moves to an adjacent cell. If the agent is holding an object the object will be placed. If an object is in the cell, the object will be picked up.
- ***Deliver***, navigates the agent to the 'star' cell for delivering with several possible termination conditions:
  - The agent places the in-hand item on the cell if it is holding any item;
  - If any other agent is standing in front of the 'star' cell, the agent stops next to the teammate;
  - The agent attempts to enter the same cell with another agent, but has a lower priority than another agent.

Observation Space: The macro-observation space for each agent is the same as the primitive observation space. Agents are only allowed to observe the *positions* and *status* of the entities within a 5×5 view centered on the agent. The initial position of all the items are known to agents.

Dynamics: The transition in this task is deterministic. If an agent delivers any wrong item, the item will be reset to its initial position. From the low-level perspective, to chop a vegetable into pieces on a cutting board, the agent needs to stand next to the cutting board and executes *left* three times. Only the chopped vegetable can be put on a plate.

Reward: $+10$ for chopping a vegetable, $+200$ terminal reward for delivering a tomato-lettuce-onion salad, -5 for delivering any wrong entity, and -0.1 for every timestep.

Episode Termination: Each episode terminates either when agents successfully deliver a tomato-lettuce-onion salad or reaching the maximal time steps, 200.

Results

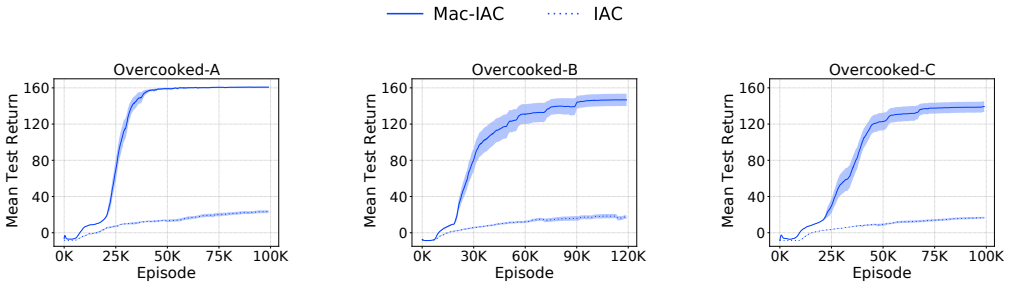


Figure 16: Decentralized learning with macro-actions vs primitive-actions.

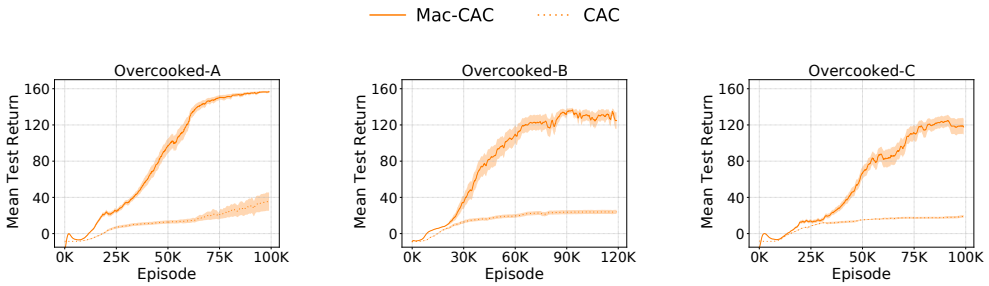


Figure 17: Centralized learning with macro-actions vs primitive-actions.

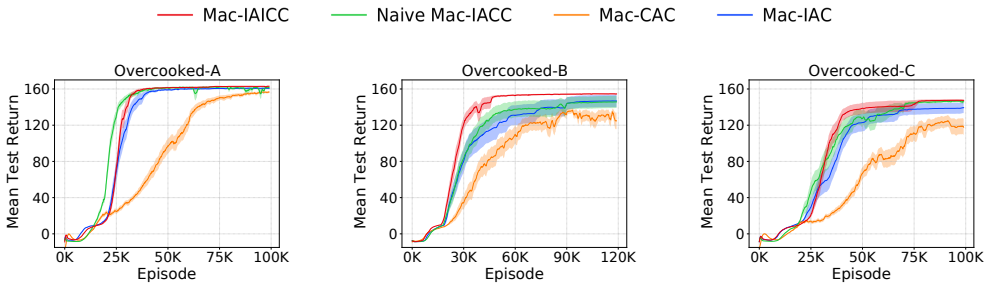


Figure 18: Comparison of macro-action-based multi-agent actor-critic methods.

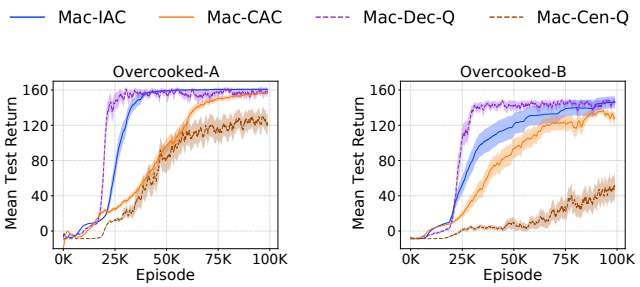
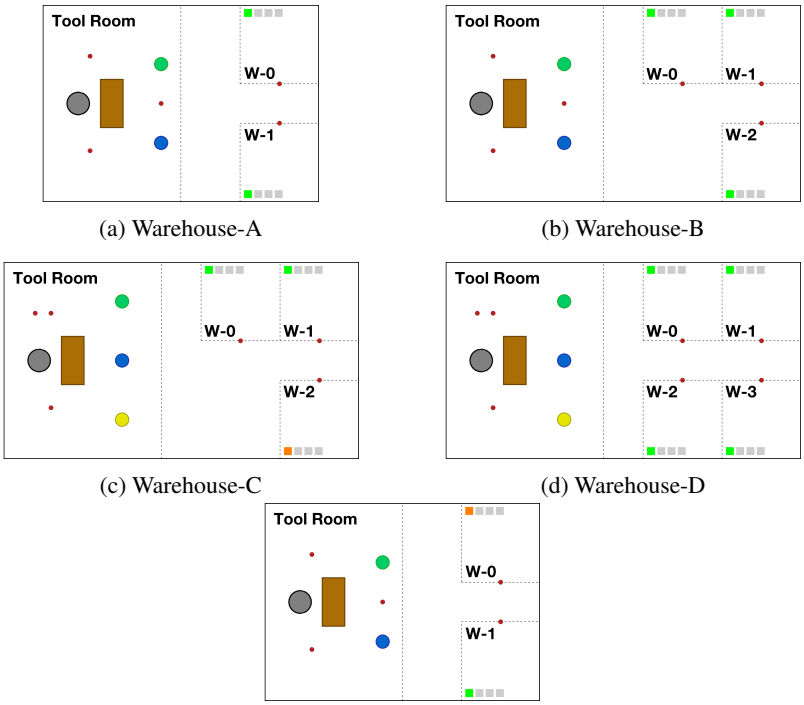


Figure 19: Comparisons of macro-action-based actor-critic methods and value-based methods.

D.3 Warehouse Tool Delivery



(a) Warehouse-A

(b) Warehouse-B

(c) Warehouse-C

(d) Warehouse-D

(e) Warehouse-E

Figure 20: Experimental environments.

In this Warehouse Tool Delivery domain, we consider five different scenarios shown in Fig. 20. To further examine the scalability of our methods and the effectiveness of Mac-IAICC on handling more noisy asynchronous terminations over robots, we consider many variants in terms of both the number of robots and the number of humans as well as having faster human(orange) in the environment.

Goal. Under all scenarios, in each workshop, a human is working on an assembly task involving 4 subtasks to be finished (each subtask takes amount of primitive time steps). At the beginning, each human has already got the tool for the first subtask and immediately starts. In order to continue, the human needs a particular tool for each following subtask. In the scenarios, humans either work in the same speed (Fig. 20a, 20b, 20d) or have one of them working faster (the orange one in Fig.20c and 20e). A team of robots includes a robot arm (gray) with the duty of finding tools for each human on the table (brown) and passing them to mobile robots (green, blue and yellow) who are responsible for delivering tools to the humans. The objective of the robots is to assist the humans to finish their assembly tasks as soon as possible by finding and delivering the correct tools in the proper order. To make this problem more challenging, the correct tools needed by each human are unknown to robots, which has to be learned during training in order to perform timely delivery without letting humans wait.

State. The environment is either a 5×7 (Fig. 20a and 20e) or a 5×9 (Fig. 20b - 20d) continuous space. A global state consists of the 2D position of each mobile robots, the execution status of the arm robot's current macro-action (e.g how munch steps are left for completing the macro-action, but in real-world, this should be the angle and speed of each arm's joint), the subtask each human is working with a percentage indicating the progress of the subtask, and the position of each tools (either on the brown table or carried by a mobile robot). The initial state of every episode is deterministic as shown in Fig. 20, where humans always start from the first step.

Macro-Action Space.

The available macro-actions for each mobile robot include:

• ***Go-W(i)***, navigates to the red waypoint at the corresponding workshop;

• ***Go-TR***, navigates to the red waypoint (covered by the blue robot in Fig. 20c and 20d) at the right side of the tool room;

• ***Get-Tool***, navigates to a pre-allocated waypoint besides the arm robot and waits over there until either 10 timesteps have passed or receiving a tool from the gray robot.

The available macro-actions for the arm robot include:

• ***Search-Tool(i)***, takes 6 timesteps to find tool i and place it in a staging area (containing at most two tools) when the area is not fully occupied, otherwise freezes the robot for the same amount of time;

• ***Pass-to-M(i)***, takes 4 timesteps to pass the first found tool to a mobile robot from the staging area;

• ***Wait-M***, takes 1 timestep to wait for mobile robots coming.

Macro-Observation Space.

The arm robot's macro-observation include the information about *the type* of each tool in the staging area and *which mobile robot* is waiting beside.

Each mobile robot always observes its own *position* and *the type* of each tool carried by itself, while observes *the number* of tools in the staging area or *the subtask* a human working on only when locating at the tool room or the workshop respectively.

Dynamics. Transitions are deterministic. Each mobile robot moves in a fixed velocity 0.8 and is only allowed to receive tools from the arm robot rather than from humans. Note that each human is only allowed to possess the tool for the next subtask from a mobile robot when the robot locates at the corresponding workshop and carries the correct tool. Humans are not allowed to pass tool back to mobile robots. There are enough tools for humans on the table in tool room, such that the number of each type of tool exactly matches with the number of humans in the environment. Human cannot start the next subtask without obtaining the correct tool. Humans' dynamics on their tasks are shown in Table 1.

Table 1: The number of time steps taken by each human on each subtask in scenarios.

Scenarios	Warehouse-A	Warehouse-B	Warehouse-C	Warehouse-D	Warehouse-E
Human-0	$[27, 20, 20, 20]$	$[40, 40, 40, 40]$	$[38, 38, 38, 38]$	$[40, 40, 40, 40]$	$[18, 15, 15, 15]$
Human-1	$[27, 20, 20, 20]$	$[40, 40, 40, 40]$	$[38, 38, 38, 38]$	$[40, 40, 40, 40]$	$[48, 18, 15, 15]$
Human-2	N/A	$[40, 40, 40, 40]$	$[27, 27, 27, 27]$	$[40, 40, 40, 40]$	N/A
Human-3	N/A	N/A	N/A	$[40, 40, 40, 40]$	N/A

Rewards. The team receives a $+100$ reward when a correct tool is delivered to a human in time while getting an extra -20 penalty for a delayed delivery such that the human has paused over there. A -10 reward occurs when the gray robot does ***Pass-to-M(i)*** but the mobile robot i is not next to it, and a -1 reward is issued every time step.

Episode Termination. Each episode terminates when all humans obtained all the correct tools for all subtasks, otherwise, the episode will run until the maximal time steps (200 for Warehouse-A and E, 250 for Warehouse-B and C, 300 for Warehous-D).

Results

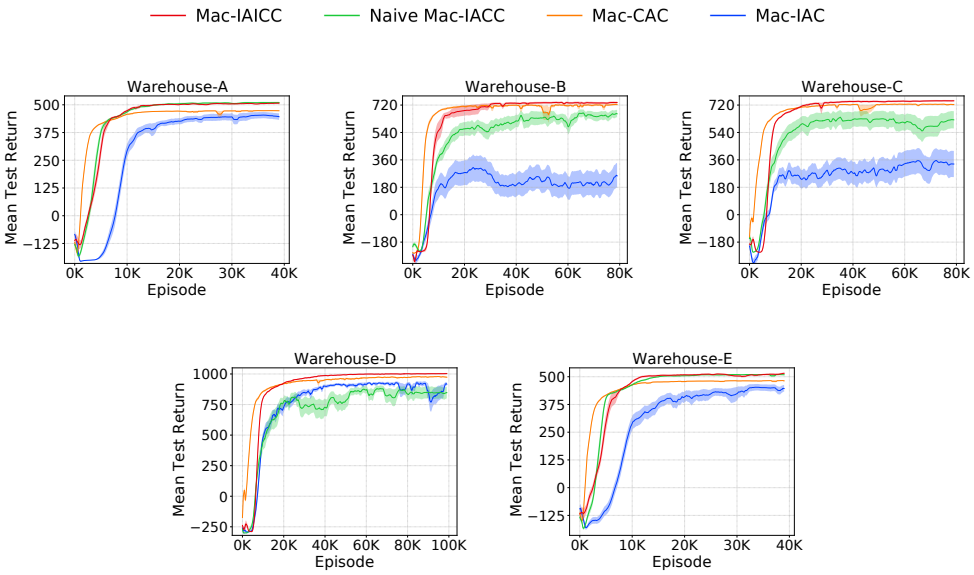


Figure 21: Comparison of macro-action-based multi-agent actor-critic methods.

Ablation Study

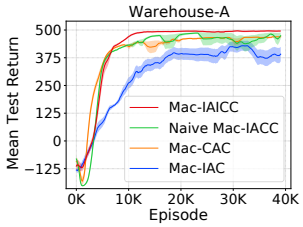


Figure 22: Results of an ablation study.

Scenarios	Ablation Experiment
Human-0	[18, 18, 18, 18]
Human-1	[18, 18, 18, 18]
Human-2	N/A
Human-3	N/A

Table 2: The number of time steps taken by each human in the ablation study.

We also conducted an ablation experiment in Warehouse-A, where two humans still operate at the same speed on their tasks but faster than the original setting. Such a change makes agents' learning more difficult, because the probability of having a delayed delivery for each tool grows, especially when agents are exploring. Agents likely receives more penalty during training. Fig. 22 shows the learning quality of Naive Mac-IACC degrades markedly and becomes much less stable with higher variance than its performance in the original domain configuration (shown in Fig. 21). In contrast, Mac-IAICC remains its high-quality performance, which reveals its robustness to noisy penalty signals and further proves the advantage of separately training a centralized critic depending on each agent's own macro-action terminations. Both Mac-CAC and Mac-IAC still cannot rival Mac-IAICC.

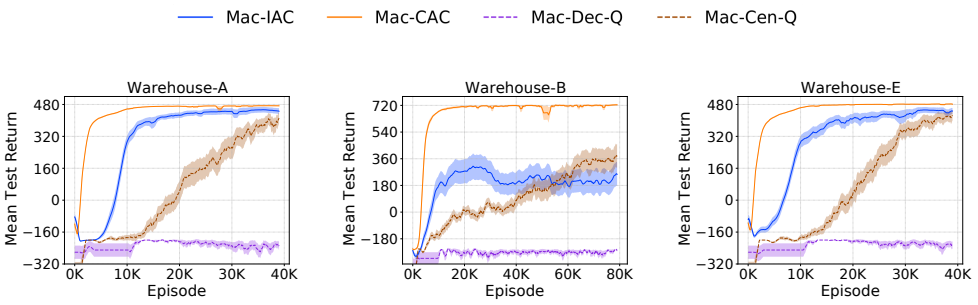


Figure 23: Comparison of macro-action-based actor-critic methods and value-based methods.

E Training Details

Our results are generated by running on a cluster of computer nodes under "CentOS Linux" operating system. We use the CPUs including "Dual Intel Xeon E5-2650", "Dual Intel Xeon E5-2680 v2", "Dual Intel Xeon E5-2690 v3".

E.1 Network Architecture

For all domains, all methods apply the same neural network architecture for both actor & critic network and Q-network. Each of them consists of two fully connected (FC) layers with Leaky-Relu activation function, one GRU layer [Cho et al., 2014] and one more FC layer followed by an output layer. The number of neurons in each layer for Decentralized(Dec) or Centralized(Cen) actor, critic or Q-network are shown in Table 3. Empirical experiments show that centralized actor and critic usually need more neurons to deal with larger joint macro-observation and macro-action spaces.

Table 3: Number of neurons on each layer in networks for all methods in domains

Domain	Box Pushing		Overcooked		Warehouse	
Actor & Critic & Q-network	Dec	Cen	Dec	Cen	Dec	Cen
MLP-1	32	32	32	128	32	32
MLP-2	32	32	32	128	32	32
GRU	32	64	32	64	32	64
MLP-3	32	32	32	64	32	32

E.2 Hyper-Parameters for macro-action-based actor-critic methods

In following subsections, we first list the hyper-parameter candidates used for tuning each method via grid search in the corresponding domain, and then show the hyper-parameter table with the parameters used by each method achieving the best performance. We choose the best performance of each method depending on its final converged value as the first priority and the sample efficiency as the second.

• Box Pushing:

Table 4: Hyper-parameter candidates for grid search tuning.

Learning rate pair (actor,critic)	(1e-3,3e-3), (1e-3,1e-3) (5e-4,3e-3), (5e-4,1e-3) (5e-4,5e-4), (3e-4,3e-3)
Episodes per train	8, 16, 32
Target-net update freq (episode)	32, 64, 128
N-step TD	0, 3, 5

Table 5: Hyper-parameter candidates for grid search tuning.

Learning rate pair (actor,critic)	(1e-3,3e-3), (1e-3,1e-3) (5e-4,3e-3), (5e-4,1e-3) (5e-4,5e-4), (3e-4,3e-3)
Episodes per train	48
Target-net update freq (episode)	48, 96, 144
N-step TD	0, 3, 5

Table 6: Hyper-parameters used for methods in Box Pushing 6×6.

Parameter	IAC	CAC	Mac-IAC	Mac-CAC	Mac-NIACC	Mac-IAICC
Training Episodes	40K	40K	40K	40K	40K	40K
Actor Learning rate	0.001	0.0005	0.0005	0.0003	0.0005	0.0003
Critic Learning rate	0.003	0.0005	0.001	0.003	0.001	0.003
Episodes per train	8	8	48	48	48	48
Target-net update freq (episode)	32	64	48	144	144	96
N-step TD	5	5	5	5	0	0
ϵ_{start}	1	1	1	1	1	1
ϵ_{end}	0.01	0.01	0.01	0.01	0.01	0.01
ϵ_{decay} (episode)	4K	4K	4K	4K	4K	4K

Table 7: Hyper-parameters used for methods in Box Pushing 8×8.

Parameter	IAC	CAC	Mac-IAC	Mac-CAC	Mac-NIACC	Mac-IAICC
Training Episodes	40K	40K	40K	40K	40K	40K
Actor Learning rate	0.001	0.001	0.001	0.0005	0.0005	0.0003
Critic Learning rate	0.003	0.003	0.003	0.003	0.001	0.003
Episodes per train	8	8	16	48	48	48
Target-net update freq (episode)	32	32	32	48	144	144
N-step TD	3	0	5	3	0	0
ϵ_{start}	1	1	1	1	1	1
ϵ_{end}	0.01	0.01	0.01	0.01	0.01	0.01
ϵ_{decay} (episode)	4K	4K	4K	4K	4K	4K

Table 8: Hyper-parameters used for methods in Box Pushing 10×10.

Parameter	IAC	CAC	Mac-IAC	Mac-CAC	Mac-NIACC	Mac-IAICC
Training Episodes	40K	40K	40K	40K	40K	40K
Actor Learning rate	0.001	0.001	0.001	0.001	0.0005	0.0003
Critic Learning rate	0.003	0.003	0.001	0.003	0.001	0.003
Episodes per train	8	8	32	48	48	32
Target-net update freq (episode)	64	32	32	96	144	64
N-step TD	0	0	5	3	0	0
ϵ_{start}	1	1	1	1	1	1
ϵ_{end}	0.01	0.01	0.01	0.01	0.01	0.01
ϵ_{decay} (episode)	6K	6K	6K	6K	6K	6K

Table 9: Hyper-parameters used for methods in Box Pushing 12×12.

Parameter	IAC	CAC	Mac-IAC	Mac-CAC	Mac-NIACC	Mac-IAICC
Training Episodes	40K	40K	40K	40K	40K	40K
Actor Learning rate	0.001	0.001	0.001	0.0005	0.0005	0.0003
Critic Learning rate	0.003	0.003	0.003	0.0005	0.001	0.003
Episodes per train	8	8	8	32	48	32
Target-net update freq (episode)	128	128	64	64	96	128
N-step TD	0	0	5	3	0	0
ϵ_{start}	1	1	1	1	1	1
ϵ_{end}	0.01	0.01	0.01	0.01	0.01	0.01
ϵ_{decay} (episode)	6K	6K	6K	6K	6K	6K

Table 10: Hyper-parameters used for methods in Box Pushing 14×14.

Parameter	IAC	CAC	Mac-IAC	Mac-CAC	Mac-NIACC	Mac-IAICC
Training Episodes	40K	40K	40K	40K	40K	40K
Actor Learning rate	0.001	0.001	0.001	0.001	0.001	0.0003
Critic Learning rate	0.003	0.003	0.003	0.001	0.003	0.003
Episodes per train	8	8	8	48	16	32
Target-net update freq (episode)	128	64	32	96	32	64
N-step TD	0	0	3	3	5	0
ϵ_{start}	1	1	1	1	1	1
ϵ_{end}	0.01	0.01	0.01	0.01	0.01	0.01
ϵ_{decay} (episode)	8K	8K	8K	8K	8K	8K

• Overcooked:

Table 11: Hyper-parameter candidates for grid search tuning.

Learning rate pair (actor,critic)	(1e-4, 3e-3) (3e-4,3e-3)
Episodes per train	4
Target-net update freq (episode)	8, 16, 32
N-step TD	3, 5

Table 12: Hyper-parameter candidates for grid search tuning.

Learning rate pair (actor,critic)	(1e-4, 3e-3) (3e-4,3e-3)
Episodes per train	8, 16
Target-net update freq (episode)	16, 32, 64
N-step TD	3, 5

Table 13: Hyper-parameters used for methods in Overcooked-A.

Parameter	IAC	CAC	Mac-IAC	Mac-CAC	Mac-NIACC	Mac-IAICC
Training Episodes	100K	100K	100K	100K	100K	100K
Actor Learning rate	0.0003	0.0003	0.0003	0.0001	0.0003	0.0003
Critic Learning rate	0.003	0.003	0.003	0.003	0.003	0.003
Episodes per train	4	8	4	8	4	8
Target-net update freq (episode)	8	16	8	32	16	32
N-step TD	5	5	5	5	5	5
ϵ_{start}	1	1	1	1	1	1
ϵ_{end}	0.05	0.05	0.05	0.05	0.05	0.05
ϵ_{decay} (episode)	20K	20K	20K	20K	20K	20K

Table 14: Hyper-parameters used for methods in Overcooked-B.

Parameter	IAC	CAC	Mac-IAC	Mac-CAC	Mac-NIACC	Mac-IAICC
Training Episodes	120K	120K	120K	120K	120K	120K
Actor Learning rate	0.0003	0.0003	0.0003	0.0001	0.0003	0.0003
Critic Learning rate	0.003	0.003	0.003	0.003	0.003	0.003
Episodes per train	4	4	4	4	8	4
Target-net update freq (episode)	8	16	8	16	16	32
N-step TD	5	5	5	3	5	5
ϵ_{start}	1	1	1	1	1	1
ϵ_{end}	0.05	0.05	0.05	0.05	0.05	0.05
ϵ_{decay} (episode)	20K	20K	20K	20K	20K	20K

Table 15: Hyper-parameters used for methods in Overcooked-C.

Parameter	IAC	CAC	Mac-IAC	Mac-CAC	Mac-NIACC	Mac-IAICC
Training Episodes	100K	100K	100K	100K	100K	100K
Actor Learning rate	0.0003	0.0003	0.0003	0.0001	0.0003	0.0003
Critic Learning rate	0.003	0.003	0.003	0.003	0.003	0.003
Episodes per train	8	8	8	8	8	8
Target-net update freq (episode)	32	32	32	32	16	32
N-step TD	5	5	5	3	5	5
ϵ_{start}	1	1	1	1	1	1
ϵ_{end}	0.05	0.05	0.05	0.05	0.05	0.05
ϵ_{decay} (episode)	20K	20K	20K	20K	20K	20K

• Warehouse Tool Delivery:

Table 16: Hyper-parameter candidates for grid search tuning.

Learning rate pair (actor,critic)	(1e-3,1e-3), (5e-4,1e-3) (5e-4,5e-4) (3e-4,3e-3)
Episodes per train	4, 8
Target-net update freq (episode)	8, 16, 32, 64
N-step TD	0, 3, 5

Table 17: Hyper-parameter candidates for grid search tuning.

Learning rate pair (actor,critic)	(1e-3,1e-3), (5e-4,1e-3) (5e-4,5e-4) (3e-4,3e-3)
Episodes per train	16
Target-net update freq (episode)	16, 32, 64
N-step TD	0, 3, 5

Table 18: Hyper-parameters used for methods in Warehouse-A.

Parameter	Mac-IAC	Mac-CAC	Mac-NIACC	Mac-IAICC
Training Episodes	40K	40K	40K	40K
Actor Learning rate	0.0003	0.0003	0.0003	0.0005
Critic Learning rate	0.003	0.003	0.003	0.0005
Episodes per train	4	4	4	4
Target-net update freq (episode)	32	32	32	32
N-step TD	5	5	3	5
ϵ_{start}	1	1	1	1
ϵ_{end}	0.01	0.01	0.01	0.01
ϵ_{decay} (episode)	10K	10K	10K	10K

Table 19: Hyper-parameters used for methods in Warehouse-A for ablation.

Parameter	Mac-IAC	Mac-CAC	Mac-NIACC	Mac-IAICC
Training Episodes	40K	40K	40K	40K
Actor Learning rate	0.0005	0.0005	0.0003	0.0005
Critic Learning rate	0.0005	0.001	0.003	0.0005
Episodes per train	16	8	8	4
Target-net update freq (episode)	16	64	64	64
N-step TD	5	5	5	5
ϵ_{start}	1	1	1	1
ϵ_{end}	0.05	0.05	0.05	0.05
ϵ_{decay} (episode)	10K	10K	10K	10K

Table 20: Hyper-parameters used for methods in Warehouse-B.

Parameter	Mac-IAC	Mac-CAC	Mac-NIACC	Mac-IAICC
Training Episodes	40K	40K	40K	40K
Actor Learning rate	0.0005	0.0005	0.0003	0.0003
Critic Learning rate	0.0005	0.001	0.003	0.003
Episodes per train	8	4	16	4
Target-net update freq (episode)	64	64	64	32
N-step TD	5	5	5	5
ϵ_{start}	1	1	1	1
ϵ_{end}	0.01	0.01	0.01	0.01
ϵ_{decay} (episode)	10K	10K	10K	10K

Table 21: Hyper-parameters used for methods in Warehouse-C.

Parameter	Mac-IAC	Mac-CAC	Mac-NIACC	Mac-IAICC
Training Episodes	80K	80K	80K	80K
Actor Learning rate	0.0005	0.0003	0.0003	0.0003
Critic Learning rate	0.001	0.003	0.003	0.003
Episodes per train	8	8	8	8
Target-net update freq (episode)	64	64	64	64
N-step TD	5	5	5	5
ϵ_{start}	1	1	1	1
ϵ_{end}	0.01	0.01	0.01	0.01
ϵ_{decay} (episode)	10K	10K	10K	10K

Table 22: Hyper-parameters used for methods in Warehouse-D.

Parameter	Mac-IAC	Mac-CAC	Mac-NIACC	Mac-IAICC
Training Episodes	80K	80K	80K	80K
Actor Learning rate	0.0003	0.0003	0.0005	0.0003
Critic Learning rate	0.003	0.003	0.005	0.003
Episodes per train	4	8	4	8
Target-net update freq (episode)	16	64	32	64
N-step TD	5	5	5	5
ϵ_{start}	1	1	1	1
ϵ_{end}	0.01	0.01	0.01	0.01
ϵ_{decay} (episode)	10K	10K	10K	10K

Table 23: Hyper-parameters used for methods in Warehouse-E.

Parameter	Mac-IAC	Mac-CAC	Mac-NIACC	Mac-IAICC
Training Episodes	100K	100K	100K	100K
Actor Learning rate	0.0005	0.0003	0.0003	0.0005
Critic Learning rate	0.001	0.003	0.003	0.0005
Episodes per train	4	4	4	4
Target-net update freq (episode)	32	16	32	32
N-step TD	5	5	5	5
ϵ_{start}	1	1	1	1
ϵ_{end}	0.05	0.05	0.05	0.05
ϵ_{decay} (episode)	10K	10K	10K	10K

E.3 Hyper-Parameters for macro-action-based value-based methods

• Box Pushing:

Table 24: Hyper-parameter candidates for grid search tuning.

Learning rate	5e-4, 1e-3
batch size	32, 64, 128

Table 25: Hyper-parameters used in Box Pushing 8×8.

Parameter	Mac-Dec-Q	Mac-Cen-Q
Training Episodes	40K	40K
Learning rate	0.001	0.001
Batch size	64	64
Replay-buffer size (step)	100K	100K
Train freq (step)	10	10
Trace length	10	10
Target-net update freq (step)	5K	5K
ϵ_{start}	1	1
ϵ_{end}	0.05	0.05
ϵ_{decay} (episode)	4K	4K

Table 26: Hyper-parameters used in Box Pushing 10×10.

Parameter	Mac-Dec-Q	Mac-Cen-Q
Training Episodes	40K	40K
Learning rate	0.001	0.001
Batch size	32	128
Replay-buffer size (step)	100K	100K
Train freq (step)	14	14
Trace length	14	14
Target-net update freq (step)	5K	5K
ϵ_{start}	1	1
ϵ_{end}	0.05	0.05
ϵ_{decay} (episode)	6K	6K

Table 27: Hyper-parameters used in Box Pushing 12×12.

Parameter	Mac-Dec-Q	Mac-Cen-Q
Training Episodes	40K	40K
Learning rate	0.001	0.001
Batch size	32	128
Replay-buffer size (step)	100K	100K
Train freq (step)	20	20
Trace length	20	20
Target-net update freq (step)	5K	5K
ϵ_{start}	1	1
ϵ_{end}	0.05	0.05
ϵ_{decay} (episode)	6K	6K

Table 28: Hyper-parameters used in Box Pushing 20×20.

Parameter	Mac-Dec-Q	Mac-Cen-Q
Training Episodes	40K	40K
Learning rate	0.001	0.001
Batch size	32	64
Replay-buffer size (step)	100K	100K
Train freq (step)	35	35
Trace length	35	35
Target-net update freq (step)	5K	5K
ϵ_{start}	1	1
ϵ_{end}	0.05	0.05
ϵ_{decay} (episode)	8K	8K

Table 29: Hyper-parameters used in Box Pushing 30×30.

Parameter	Mac-Dec-Q	Mac-Cen-Q
Training Episodes	40K	40K
Learning rate	0.0005	0.001
Batch size	32	32
Replay-buffer size (step)	100K	100K
Train freq (step)	45	45
Trace length	45	45
Target-net update freq (step)	5K	5K
ϵ_{start}	1	1
ϵ_{end}	0.05	0.05
ϵ_{decay} (episode)	8K	8K

• Overcooked:

Table 30: Hyper-parameter candidates for grid search tuning.

Learning rate	3e-5, 5e-5, 1e-4, 3e-4, 5e-4
batch size	32, 64
Train freq (step)	64, 128
Replay-buffer size (episode)	500, 1K, 2K, 3K

Table 31: Hyper-parameters used in Overcooked-A.

Parameter	Mac-Dec-Q	Mac-Cen-Q
Training Episodes	100K	100K
Learning rate	0.0005	0.00003
Batch size	64	64
Replay-buffer size (episode)	1K	1K
Train freq (step)	64	64
Target-net update freq (step)	5K	5K
ϵ_{start}	1	1
ϵ_{end}	0.05	0.05
ϵ_{decay} (episode)	20K	20K

Table 32: Hyper-parameters used in Overcooked-B.

Parameter	Mac-Dec-Q	Mac-Cen-Q
Training Episodes	100K	100K
Learning rate	0.0005	0.0001
Batch size	32	32
Replay-buffer size (episode)	3K	500
Train freq (step)	64	64
Target-net update freq (step)	5K	5K
ϵ_{start}	1	1
ϵ_{end}	0.05	0.05
ϵ_{decay} (episode)	20K	20K

• Warehouse Tool Delivery:

Table 33: Hyper-parameter candidates for grid search tuning.

Learning rate	5e-5, 1e-4
batch size	32, 64
Train freq (step)	64, 128
Replay-buffer size (episode)	1K, 2K

Table 34: Hyper-parameters used in Warehouse-A.

Parameter	Mac-Dec-Q	Mac-Cen-Q
Training Episodes	40K	40K
Learning rate	0.0001	0.0001
Batch size	64	64
Replay-buffer size (episode)	2K	2K
Train freq (step)	128	128
Target-net update freq (step)	5K	5K
ϵ_{start}	1	1
ϵ_{end}	0.05	0.05
ϵ_{decay} (episode)	10K	10K

Table 35: Hyper-parameters used in Warehouse-B.

Parameter	Mac-Dec-Q	Mac-Cen-Q
Training Episodes	40K	40K
Learning rate	0.00005	0.00005
Batch size	64	64
Replay-buffer size (episode)	2K	2K
Train freq (step)	128	128
Target-net update freq (step)	5K	5K
ϵ_{start}	1	1
ϵ_{end}	0.05	0.05
ϵ_{decay} (episode)	10K	10K

Table 36: Hyper-parameters used in Warehouse-E.

Parameter	Mac-Dec-Q	Mac-Cen-Q
Training Episodes	100K	100K
Learning rate	0.0001	0.0001
Batch size	64	64
Replay-buffer size (episode)	2K	2K
Train freq (step)	128	128
Target-net update freq (step)	5K	5K
ϵ_{start}	1	1
ϵ_{end}	0.05	0.05
ϵ_{decay} (episode)	10K	10K

F Hardware Experiments

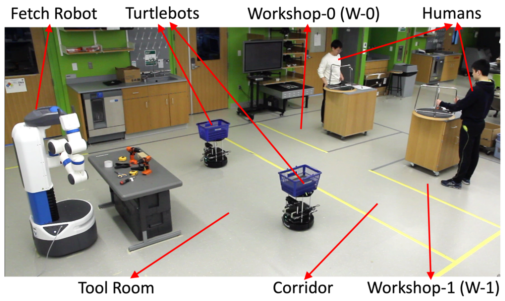


Figure 24: Overview of Warehouse-A hardware domain.

While simulation results validate that the proposed Mac-IAICC approach achieves the best performance for learning decentralized policies in various macro-action-based domains, we also extend scenario A of the Warehouse Tool Delivery task to a hardware domain. Fig. 24 provides an overview of the real-world experimental setup. An open area is divided into regions, a tool room, a corridor, and two workshops, to resemble the configuration shown in Fig. 1e. This mission involves one Fetch Robot Wise et al. [2016] and two Turtlebots Koubaa et al. [2016] to cooperatively find and deliver three YCB tools Calli et al. [2015], in the order: a tape measure, a clamp and an electric drill, required by each human in order to assemble an IKEA table.

The Turtlebot's navigation macro-actions were executed by using the ROS navigation stack Marder-Eppstein et al. [2010]. For Fetch's manipulation macro-actions, we combined PCL bindings for Python Gualtieri et al. [2018], MoveIt Coleman et al. and the OpenRave simulator Diankov and Kuffner [2008] with an OMPL Şucan et al. [2012] plugin to achieve picking and placing of tools. The information about the number of tools in staging areas and each human's working status was tracked and broadcast by ROS services but were only observable in the tool room and the corresponding workshop area respectively (to simulate possible visual information).

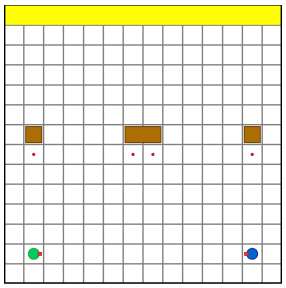
For the visualization of the real-robot experiment, please check the video in our supplementary.

G Behavior Visualization in Simulation

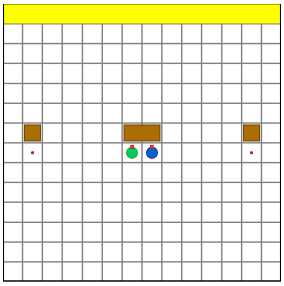
In this section, we display the decentralized behaviors learned by using Mac-IAICC under all considered domains.

G.1 Box Pushing

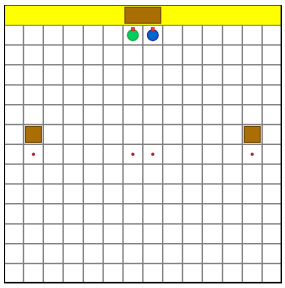
We show the behaviors learned under the grid world size 14×14 in Fig. 25. Although the averaged performance of the training is not near-optimal (Fig. 13), several runs can learn the optimal behavior.



(a) Green robot executes ***Move-to-big-box(1)*** to move to the left waypoint below the big box while the blue robot runs ***Move-to-big-box(2)*** to move to the right waypoint below the big box.



(b) After completing the previous macro-actions, robots choose ***Push*** to move the big box towards the goal together.

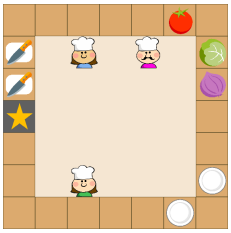


(c) Robots finish the task by pushing the big box to the goal area.

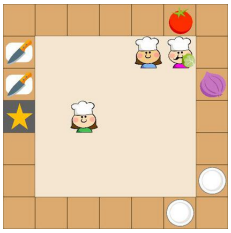
Figure 25: Visualization of the optimal macro-action-based behaviors learned using Mac-IAICC in the Box Pushing domain under a 14×14 grid world.

G.2 Overcooked

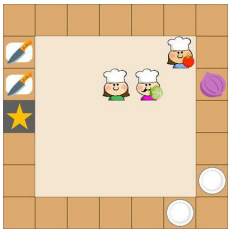
Map A: In this map, our method learns an efficient collaboration such as three agents separately get three different vegetables, and then go to the cutting board and chop them respectively. Especially, the pink agent leans to take away the chopped lettuce in order to make room for the incoming green agent to chop the onion (Fig. 27h - 27l). Details are shown below.



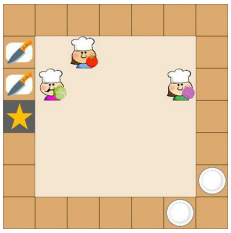
(a) The blue agent executes ***Get-Lettuce***. The pink agent executes ***Get-Tomato***. The green agent executes ***Get-Onion***.



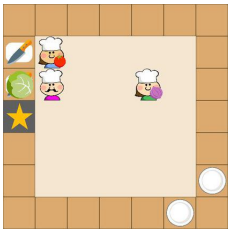
(b) After getting the lettuce, the pink agent executes ***Go-Cut-Board-2***.



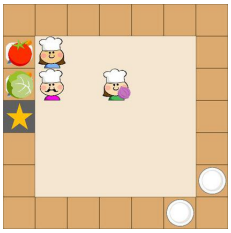
(c) After getting the tomato, the blue agent executes ***Go-Cut-Board-1***.



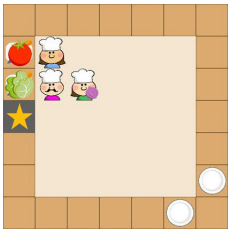
(d) After getting the onion, the green agent executes ***Go-Cut-Board-2***.



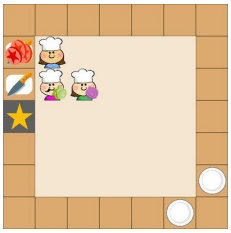
(e) After placing the lettuce on the cutting board, the pink agent executes ***Chop***.



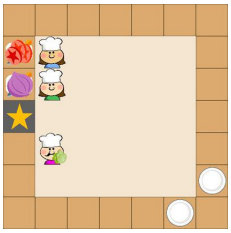
(f) After placing the tomato on the cutting board, the blue agent executes ***Chop***.



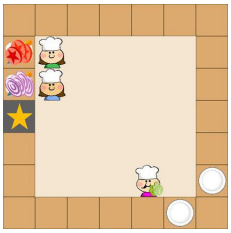
(g) After finishing chopping the lettuce, the pink agent executes ***Get-Lettuce*** to pick it up.



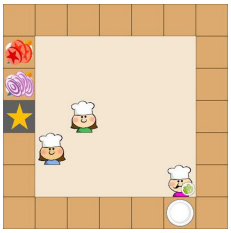
(h) With the lettuce in hand, the pink agent executes ***Get-Plate-1***.



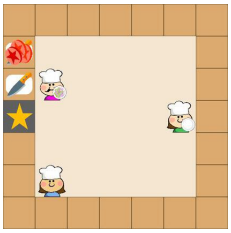
(i) After placing the onion on the cutting board, the green agent executes ***Chop***.



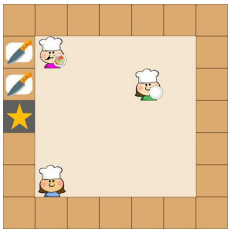
(j) The green agent executes ***Get-Plate-2***, and the blue agent keep running ***Move-Down*** to make room for the pink agent to merge the chopped vegetables later on.



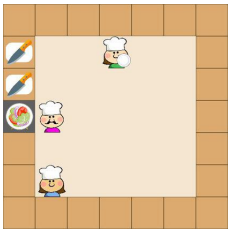
(k) The pink agent reaches the plate and it is going to put the lettuce on the plate.



(l) After putting the lettuce on the plate, the pink agent merges the onion in the plate by executing executes ***Get-Onion***.



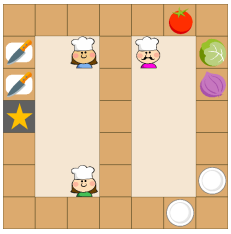
(m) The pink agent gets the chopped tomato into the plate by executing *Get-Tomato*.



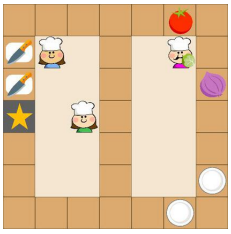
(n) The pink agent successfully delivers the tomato-lettuce-onion salad by running *Deliver*.

Figure 27: Visualization of running decentralized policies learned by Mac-IAICC in Overcooked-A.

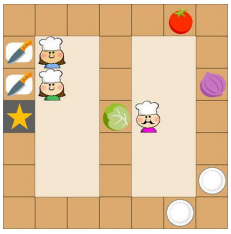
Map B: In this map, the decentralized policies trained by our method learns the collaboration such that the pink agent focuses on transporting items from right to left, while the other two agents cooperatively prepare the salad.



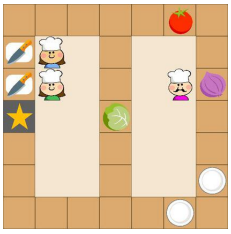
(a) The blue agent executes ***Go-Cut-board-1***. The green agent executes ***Go-Cut-board-2***. The pink agent executes ***Get-Lettuce***.



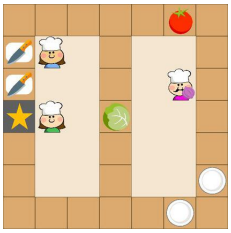
(b) After getting the lettuce, the pink agent executes ***Go-Counter***.



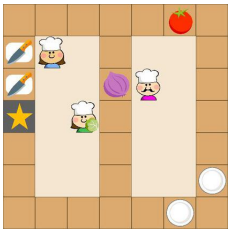
(c) After putting the lettuce on the counter, the pink agent executes ***Get-Onion***.



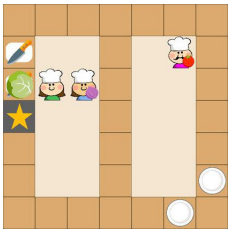
(d) The green agent executes ***Get-Lettuce***.



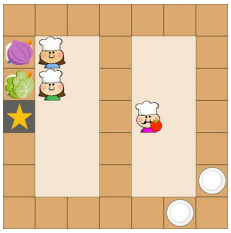
(e) After getting the onion, the pink agent executes ***Go-Counter***.



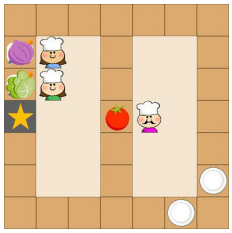
(f) After getting the lettuce, the green agent executes ***Go-Cut-Board-2***. Meanwhile the pink agent puts the onion on the counter, and then executes ***Get-Tomato***. The blue agent executes ***Get-Onion***.



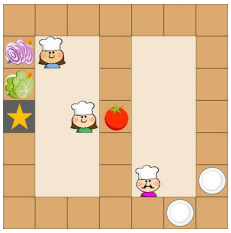
(g) After putting the lettuce on the cutting board, the green executes ***Chop***. Blue agent executes ***Go-Cut-Board-1*** with onion in hand.



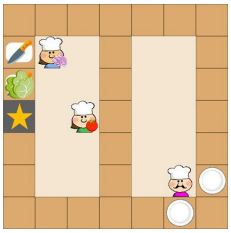
(h) The blue agent executes ***Chop*** to cut the onion to pieces.



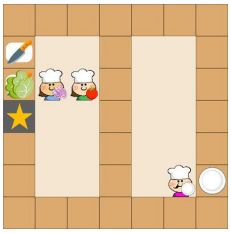
(i) After putting the tomato on the counter, the pink agent executes ***Get-Plate-2***. The green agent executes ***Get-Tomato***.



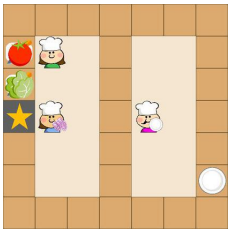
(j) The blue agent finishes chopping the onion, and then picks it up by executing ***Get-Onion*** again.



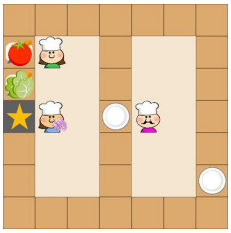
(k) The blue agent keeps executing ***Moving-Down*** to make room for the green agent to chop tomato later on. Meanwhile the green agent moves towards the upper cutting board by executing ***Go-Cut-Board-1***.



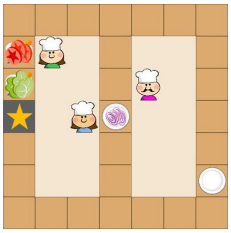
(l) After getting the plate, the pink agent executes ***Go-Counter***.



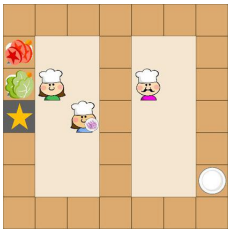
(m) After putting the tomato on the cutting board, the green agent executes ***Chop***.



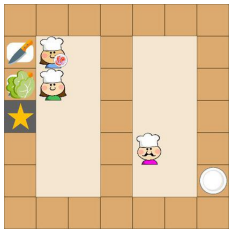
(n) The pink agent puts the plate on the counter. The blue agent executes ***Go-Counter*** to get the plate.



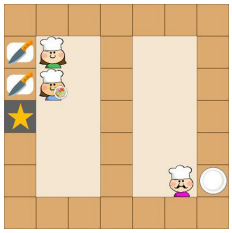
(o) The green agent finishes chopping the tomato, while the blue agent puts chopped onion on the plate.



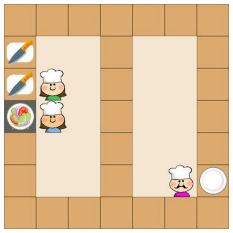
(p) The green agent executes *Go-Cut-Board-2* to make room for the blue agent to merge the tomato in to the plate.



(q) The blue agent executes ***Get-Lettuce***. The green agent executes ***Go-Cut-Board-1***.



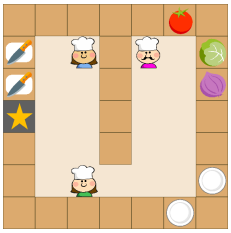
(r) After putting the lettuce on the plate, the blue agent executes ***Deliver***.



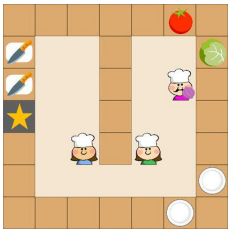
(s) The blue agent successfully delivers the tomato-lettuce-onion salad.

Figure 29: Visualization of running decentralized policies learned by Mac-IAICC in Overcooked-B.

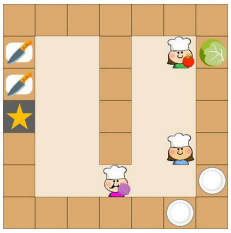
Map C: In this map, the best strategy is that the pink agent should take the advantage of the middle counters to pass vegetables to the other agent. Our method learns a sub-optimal policy such that the blue agent still crosses the narrow passage to get the vegetable at the right side of the map.



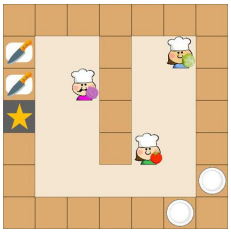
(a) The blue agent executes ***Get-Lettuce***. The pink agent executes ***Get-Onion***. The green agent executes ***Get-Tomato***.



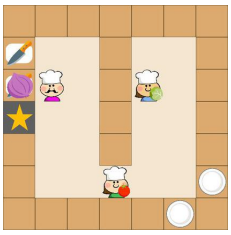
(b) After getting the onion, the pink agent executes ***Go-Cut-Board-2***.



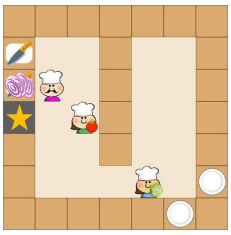
(c) The green agent gets the tomato, and it executes ***Go-Cut-Board-2***.



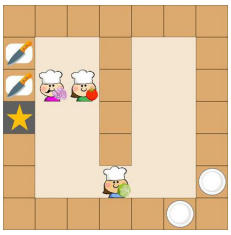
(d) The blue agent gets the lettuce, and it executes ***Go-Cut-Board-1***.



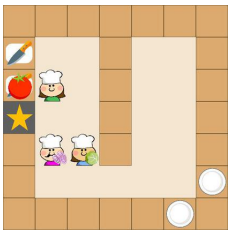
(e) After putting the onion on the cutting board, the pink agent executes ***Chop***.



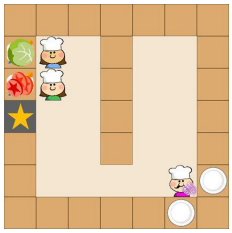
(f) The pink agent finishes chopping the onion, and then it executes ***Get-Onion*** to pick it up.



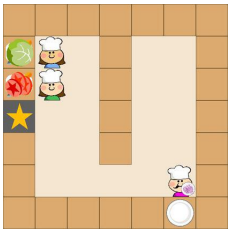
(g) After picking up the onion, the pink agent executes ***Get-Plate-1***.



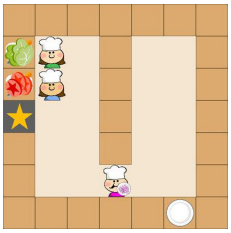
(h) After putting the tomato on the cutting board, the green agent executes ***Chop***.



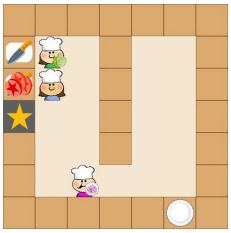
(i) After putting the lettuce on the cutting board, the blue agent executes ***Chop***.



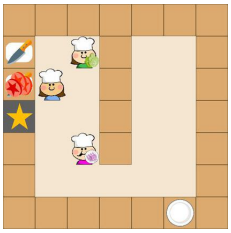
(j) The pink agent puts the onion on the plate, and then executes ***Go-Cut-Board-2***. The blue agent executes ***Go-Cut-Board-2***. The green agent executes ***Go-Cut-Board-1***.



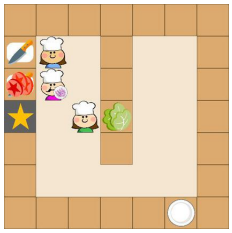
(k) The green agent executes ***Get-Lettuce***.



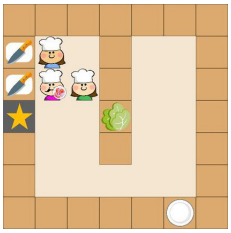
(l) After picking up the lettuce, the green agent executes ***Go-Counter***.



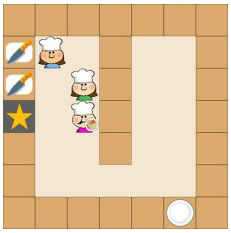
(m) The blue agent executes ***Go-Cut-Board-1*** to make room for the pink agent.



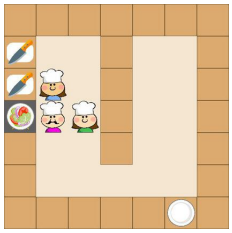
(n) The green agent puts the lettuce on the counter



(o) After merging the tomato into the plate, the pink agent executes ***Get-Lettuce***. Meanwhile the green agent steps away to make room for the pink agent.



(p) After getting the lettuce, the pink agent executes ***Deliver***.

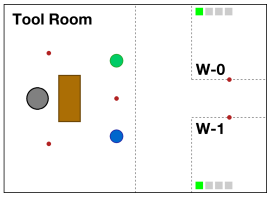


(q) The pink agent successfully delivers the tomato-lettuce-onion salad.

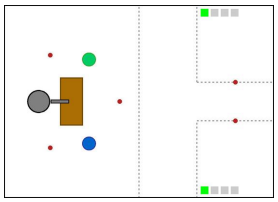
Figure 31: Visualization of running decentralized policies learned by Mac-IAICC in Overcooked-C.

G.3 Warehouse Tool Delivery

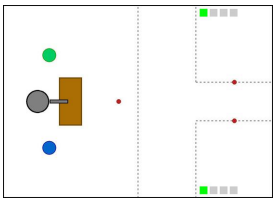
Warehouse A:



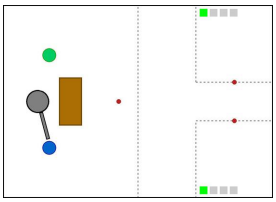
(a) Initial State.



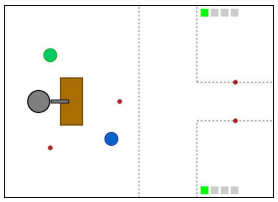
(b) Mobile robots moves towards the table by running ***Get-Tool***, and arm robot runs ***Search-Tool(0)*** to find Tool-0.



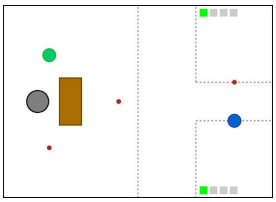
(c) Mobile robots wait there and arm robot keeps looking for Tool-0.



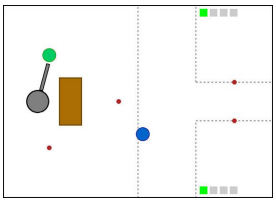
(d) Arm robot executes ***Pass-to-M(1)*** to pass Tool-0 to the blue robot.



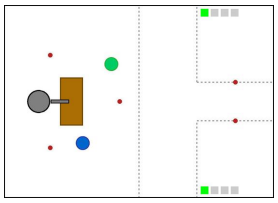
(e) Arm robot executes ***Search-Tool(0)*** to find Tool-0, and blue robot moves to workshop-1 by executing ***Go-W(1)***.



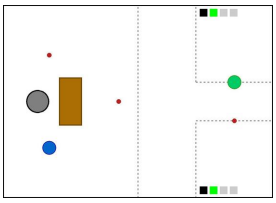
(f) Blue robot successfully delivers Tool-0 to workshop-1.



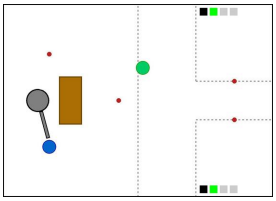
(g) Blue robot runs ***Get-Tool*** to go back table, and arm robot executes ***Pass-to-M(0)*** to pass Tool-0 to green robot.



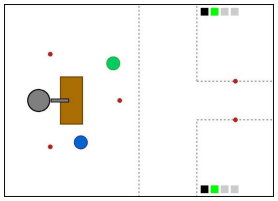
(h) Green robot executes ***Go-W(0)*** and arm robot runs ***Search-Tool(1)***.



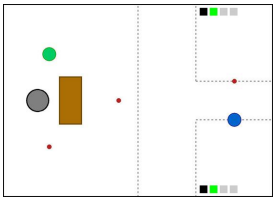
(i) Green robot successfully delivers Tool-0 to workshop-0. Human-0 and human-1 finish subtask-0 and start to do subtask-1 with delivered Tool-0.



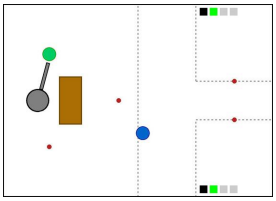
(j) Green robot runs ***Get-Tool*** to go back table, and arm robot executes ***Pass-to-M(1)*** to pass a Tool-1 to blue robot.



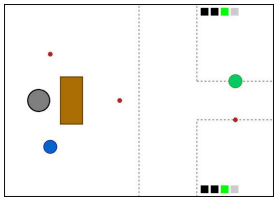
(k) Blue robot executes ***Go-W(1)*** and arm robot runs ***Search-Tool(1)***.



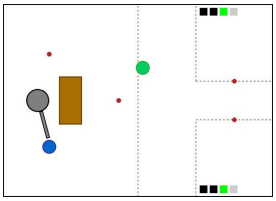
(l) Blue robot successfully delivers a Tool-1 to workshop-1.



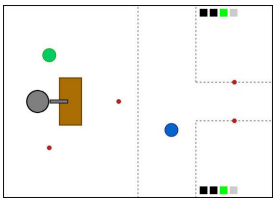
(m) Arm robot executes ***Pass-to-M(0)*** to pass Tool-1 to green robot. Blue robot runs ***Get-Tool*** to go back table.



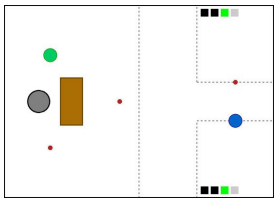
(n) Green robot successfully delivers Tool-1 to workshop-0. Human-0 and human-1 finish subtask-1 and start to do subtask-2 with delivered Tool-1.



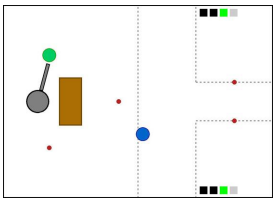
(o) Arm robot executes ***Pass-to-M(1)*** to pass Tool-2 to blue robot. Green robot runs ***Get-Tool*** to go back table.



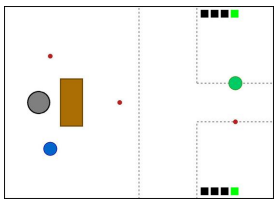
(p) Blue robot executes ***Go-W(1)***. Arm robot runs ***Search-Tool(2)***.



(q) Blue robot successfully delivers Tool-2 to human-0.

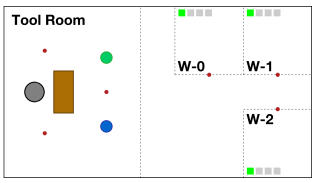


(r) Arm robot executes ***Pass-to-M(0)*** to pass Tool-2 to green robot. Blue robot runs ***Get-Tool*** to go back table.

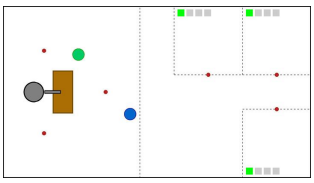


(s) Green robot directly goes to workshop-0 by running ***Go-W(0)*** and finishes the last tool delivery for human-0. The entire task is done.

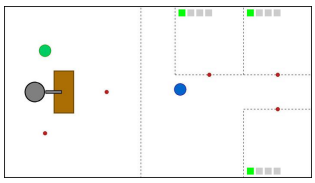
Warehouse-B:.



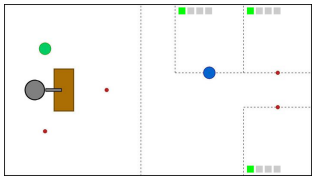
(a) Initial State.



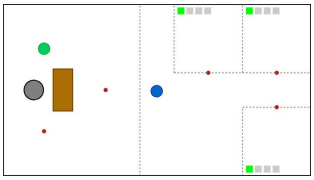
(b) Green robot moves towards the table by running ***Get-Tool***. Blue robot moves to workshop-0 by executing ***Go-W(0)***. Arm robot runs ***Search-Tool(0)*** to find Tool-0.



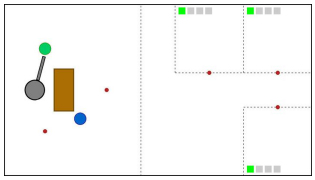
(c) Green robot waits there and arm robot keeps looking for Tool-0.



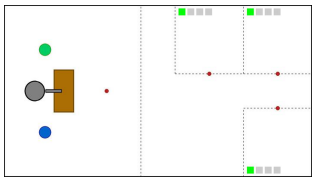
(d) Blue robot reaches workshop-0.



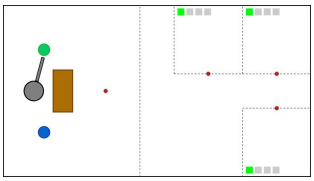
(e) Blue robot runs ***Get-Tool*** to go back table.



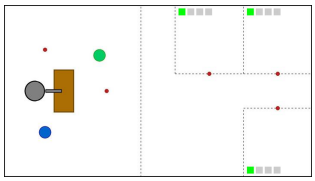
(f) Arm robot executes ***Pass-to-M(0)*** to pass Tool-0 to green robot.



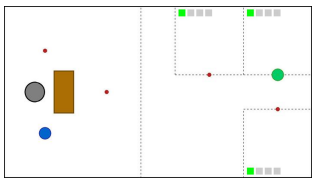
(g) Arm robot runs ***Search-Tool(0)*** to find the 2nd Tool-0.



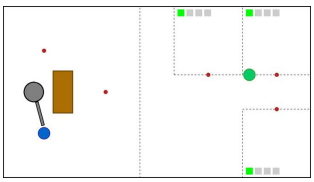
(h) Arm robot executes ***Pass-to-M(0)*** to pass the 2nd Tool-0 to green robot.



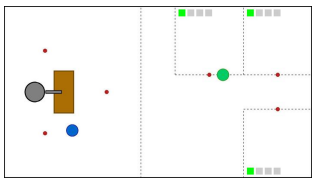
(i) Arm robot runs ***Search-Tool(0)*** to find the the 3rd Tool-0. Green robot moves to workshop-1 by executing ***Go-W(1)***.



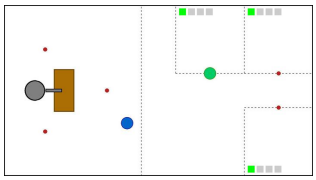
(j) Green robot successfully delivers Tool-0 to workshop-1.



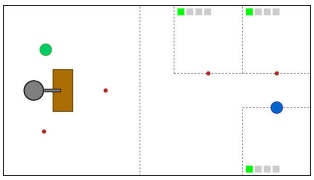
(k) Arm robot executes ***Pass-to-M(1)*** to pass the 3rd Tool-0 to blue robot. Green robot moves to workshop-0 by executing ***Go-W(0)***.



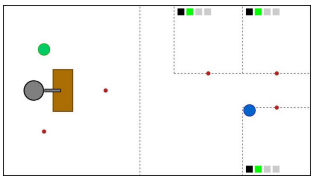
(l) Arm robot runs ***Search-Tool(1)*** to find Tool-1. Blue robot moves to workshop-2 by executing ***Go-W(2)***.



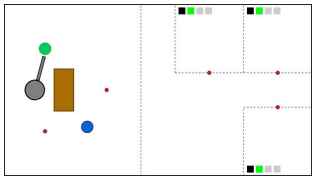
(m) Green robot successfully delivers a Tool-0 to workshop-0.



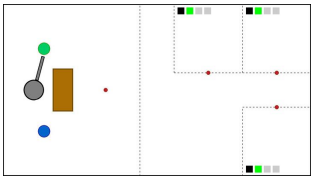
(n) Blue robot successfully delivers a Tool-0 to workshop-2. Arm robot runs ***Search-Tool(1)*** to find another Tool-1.



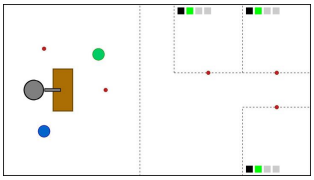
(o) Blue robot runs ***Get-Tool*** to go back table. All humans finish subtask-0 and start to do subtask-1.



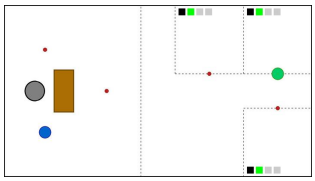
(p)Arm robot executes ***Pass-to-M(0)*** to pass a Tool-1 to green robot.



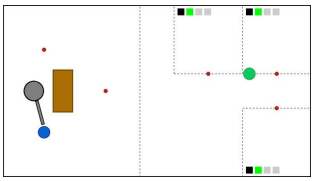
(q) Arm robot executes ***Pass-to-M(0)*** to pass another Tool-1 to green robot.



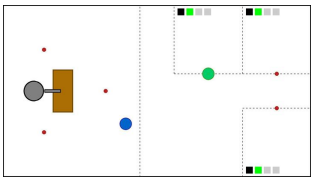
(r) Green robot moves to workshop-1 by executing ***Go-W(1)***. Arm robot runs ***Search-Tool(1)*** to find the 3rd Tool-1.



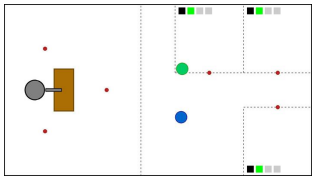
(s) Green robot successfully delivers a Tool-0 to workshop-0.



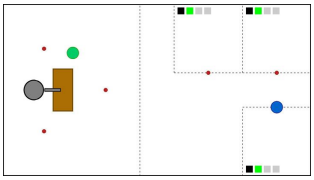
(t) Green robot moves to workshop-0 by executing ***Go-W(0)***. Arm robot executes ***Pass-to-M(1)*** to pass the 3rd Tool-1 to blue robot.



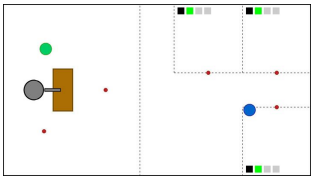
(u) Arm robot runs ***Search-Tool(2)*** to find Tool-2. Green robot successfully delivers a Tool-1 to workshop-0. Blue robot moves to workshop-2 by executing ***Go-W(2)***.



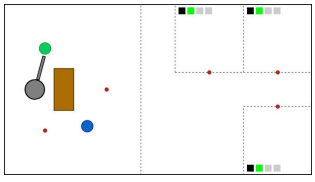
(v) Green robot runs ***Get-Tool*** to go back table.



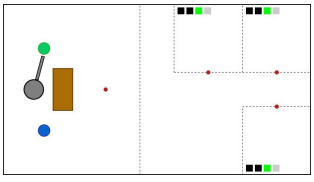
(w) Arm robot runs ***Search-Tool(2)*** to find another Tool-2. Blue robot successfully delivers a Tool-1 to workshop-2.



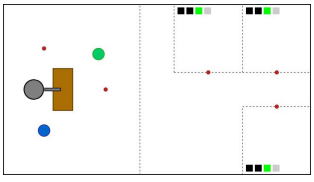
(x) Blue robot runs ***Get-Tool*** to go back table.



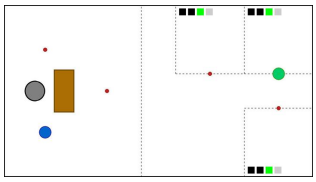
(y) Arm robot executes ***Pass-to-M(0)*** to pass a Tool-2 to green robot.



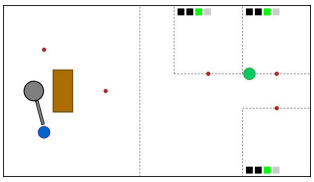
(z) Arm robot executes ***Pass-to-M(0)*** to pass another Tool-2 to green robot. All humans finish subtask-1 and start to do subtask-2.



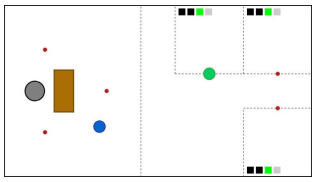
(A) Arm robot runs ***Search-Tool(2)*** to find the 3rd Tool-2. Green robot moves to workshop-1 by executing ***Go-W(1)***.



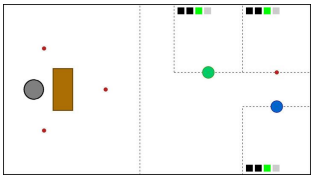
(B) Green robot successfully delivers a Tool-2 to workshop-1.



(C) Arm robot executes ***Pass-to-M(1)*** to pass the 3rd Tool-2 to blue robot.

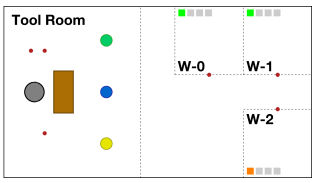


(D) Green robot successfully delivers a Tool-2 to workshop-0. Blue robot moves to workshop-2 by executing ***Go-W(2)***.

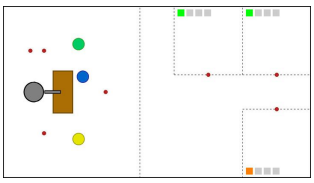


(E) Blue robot successfully delivers a Tool-2 to workshop-2. Humans have received all tools, and for robots, the task is done.

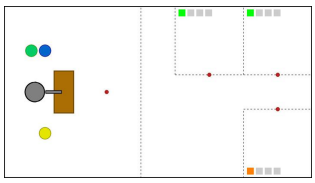
Warehouse-C:



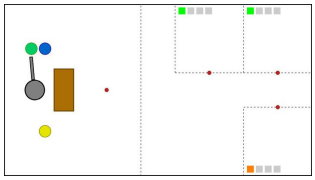
(a) Initial State.



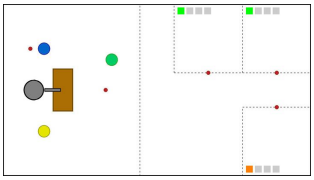
(b) Mobile robots move towards the table by running ***Get-Tool***. Arm robot runs ***Search-Tool(0)*** to find the 1st Tool-0.



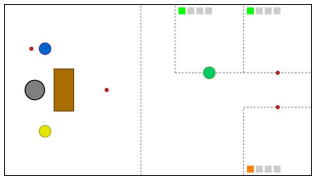
(c) Mobile robots wait there and arm robot keeps looking for the 1st Tool-0.



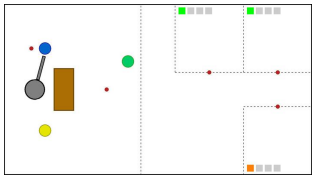
(d) Arm robot executes ***Pass-to-M(0)*** to pass a Tool-0 to green robot.



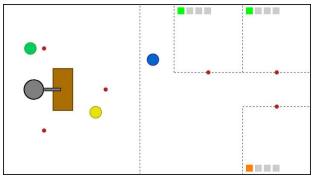
(e) Arm robot runs ***Search-Tool(0)*** to find the 2nd Tool-0. Green robot moves to workshop-0 by executing ***Go-W(0)***.



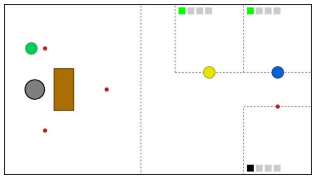
(f) Green robot successfully delivers the a Tool-0 to workshop-0.



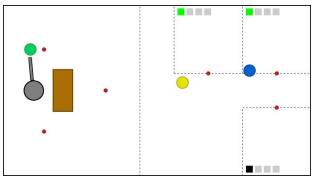
(g) Green robot runs ***Get-Tool*** to go back table. Arm robot executes ***Pass-to-M(1)*** to pass a Tool-0 to blue robot.



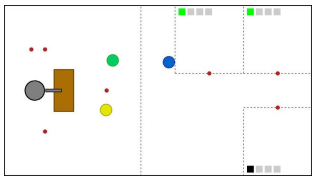
(h) Arm robot runs ***Search-Tool(0)*** to find the 3rd Tool-0. Blue robot moves to workshop-1 by executing ***Go-W(1)***. Yellow robot moves to workshop-0 by executing ***Go-W(0)***.



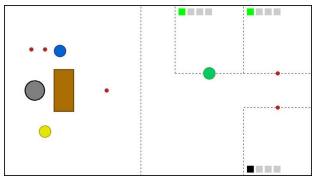
(i) Blue robot successfully delivers the a Tool-0 to workshop-1. Yellow robot reaches workshop-0 and observes that human-0 has got Tool-0. Human-2 finishes subtask-0 and waits for Tool-0.



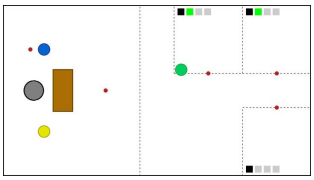
(j) Arm robot executes ***Pass-to-M(0)*** to pass a Tool-0 to green robot. Yellow and blue robots run ***Get-Tool*** to go back table.



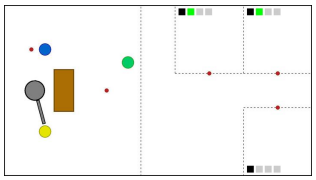
(k) Arm robot runs ***Search-Tool(1)*** to find the 1st Tool-1. Green robot moves to workshop-0 by executing ***Go-W(0)***.



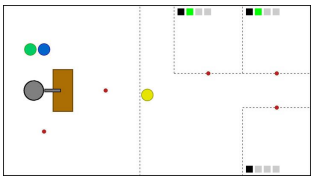
(l) Green robot reaches workshop-0 and observes that human-0 does not need Tool-0.



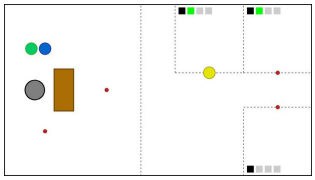
(m) Green robot runs ***Get-Tool*** to go back table.



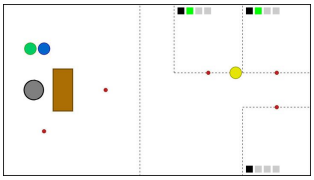
(n) Arm robot executes ***Pass-to-M(2)*** to pass a Tool-1 to yellow robot.



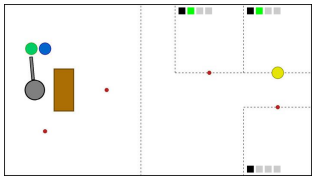
(o) Arm robot runs ***Search-Tool(1)*** to find the 2nd Tool-1. Yellow robot moves to workshop-0 by executing ***Go-W(0)***.



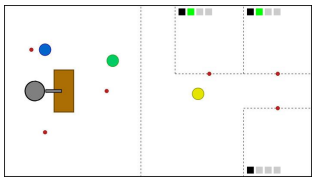
(p) Yellow robot successfully delivers the a Tool-1 to workshop-0.



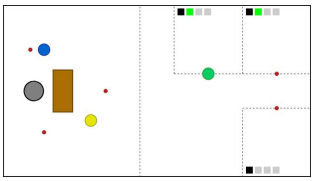
(q) Yellow robot moves to workshop-1 by executing ***Go-W(1)***.



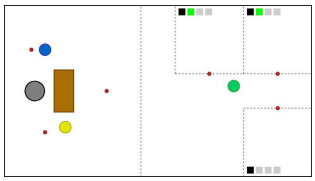
(r) Arm robot executes ***Pass-to-M(0)*** to pass a Tool-1 to green robot.



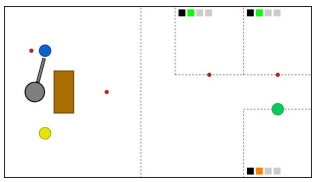
(s) Arm robot runs ***Search-Tool(1)*** to find 3st Tool-1. Yellow robot runs ***Get-Tool*** to go back table. Green robot moves to workshop-0 by executing ***Go-W(0)***.



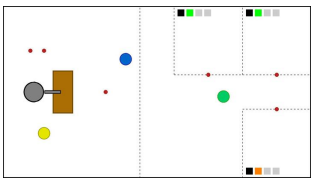
(t) Green robot successfully delivers a Tool-1 to workshop-0.



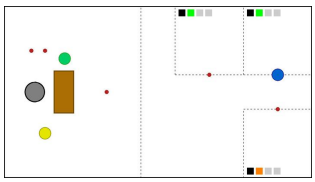
(u) Green robot moves to workshop-2 by executing ***Go-W(2)***.



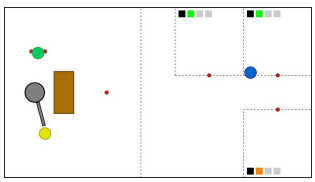
(v) Green robot successfully delivers the a Tool-0 to workshop-2. Human-2 finishes subtask-0 and starts to do subtask-1. Arm robot executes ***Pass-to-M(1)*** to pass a Tool-1 to blue robot.



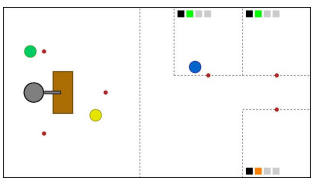
(w) Green robot runs ***Get-Tool*** to go back table. Arm robot runs ***Search-Tool(2)*** to find the 1st Tool-2. Blue robot moves to workshop-1 by executing ***Go-W(1)***.



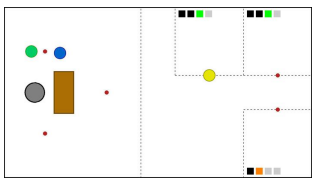
(x) Blue robot successfully delivers a Tool-1 to workshop-1.



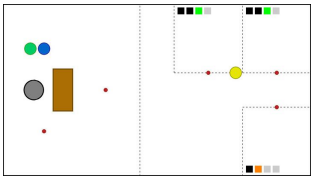
(y) Blue robot runs ***Get-Tool*** to go back table. Arm robot executes ***Pass-to-M(2)*** to pass a Tool-2 to yellow robot.



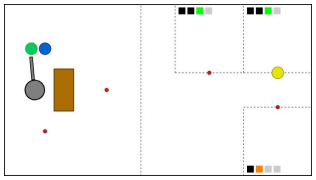
(z) Arm robot runs ***Search-Tool(2)*** to find the 2nd Tool-2. Yellow robot moves to workshop-1 by executing ***Go-W(1)***.



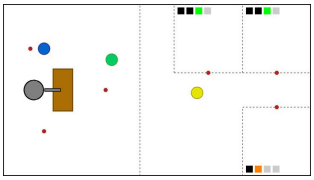
(A) Yellow robot successfully delivers a Tool-2 to workshop-0. Human-0 and human-1 finish subtask-1 and start to do subtask-2.



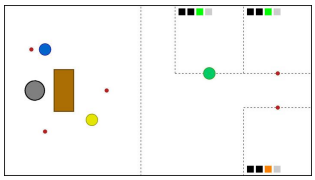
(B) Yellow robot moves to workshop-1 by executing ***Go-W(1)***.



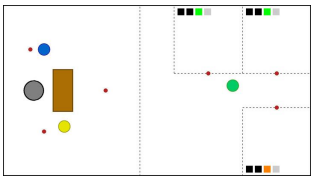
(C) Arm robot executes ***Pass-to-M(0)*** to pass a Tool-2 to green robot. Yellow robot reaches workshop-1 but it does not have any tool.



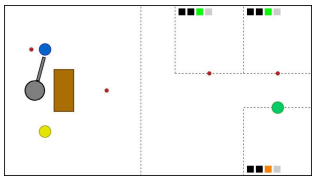
(D) Arm robot runs ***Search-Tool(2)*** to find the 3rd Tool-2. Green robot moves to workshop-0 by executing ***Go-W(0)***. Yellow robot runs ***Get-Tool*** to go back table.



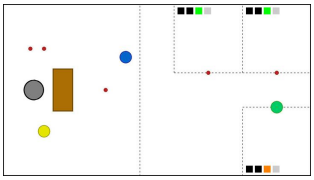
(E) Green robot reaches workshop-0 and observes that human-0 does not need Tool-2. Human-2 finishes subtask-1 and starts to do subtask-2.



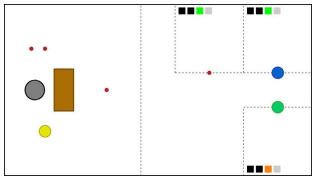
(F) Green robot moves to workshop-2 by executing ***Go-W(2)***.



(G) Green robot successfully delivers a Tool-2 to workshop-2. Arm robot executes ***Pass-to-M(1)*** to pass a Tool-2 to blue robot.

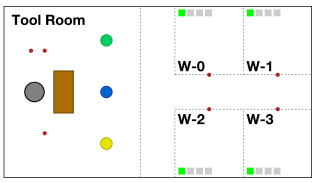


(H) Blue robot moves to workshop-1 by executing ***Go-W(1)***.

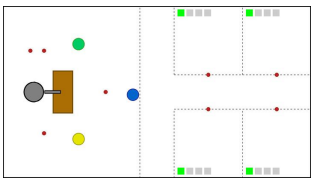


(I) Blue robot successfully delivers a Tool-2 to workshop-1. Humans have received all tools, and for robots, the task is done.

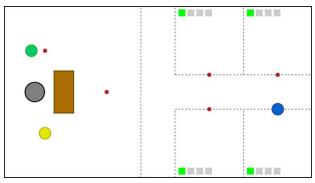
Warehouse-D:



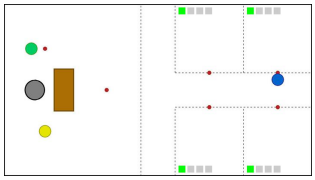
(a) Initial State.



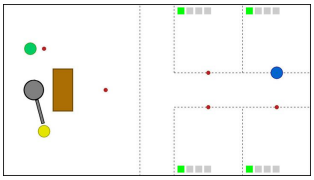
(b) Green and yellow robots move towards the table by running ***Get-Tool***. Blue robot moves to workshop-3 by executing ***Go-W(3)***. Arm robot runs ***Search-Tool(0)*** to find the 1st Tool-0.



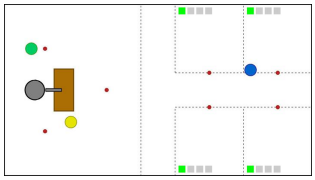
(c) Blue robot reaches workshop-3.



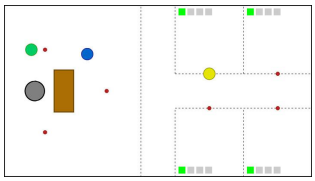
(d) Blue robot moves to workshop-1 by executing ***Go-W(1)***.



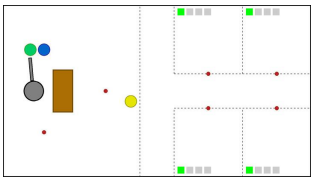
(e) Blue robot reaches workshop-1. Arm robot executes ***Pass-to-M(2)*** to pass a Tool-0 to yellow robot.



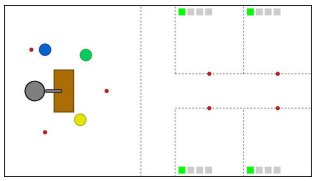
(f) Arm robot runs ***Search-Tool(0)*** to find the 2nd Tool-0. Blue robot runs ***Get-Tool*** to go back table. Yellow robot moves to workshop-1 by executing ***Go-W(1)***.



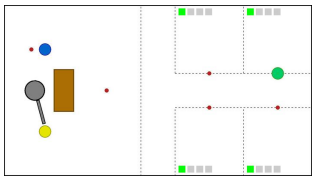
(g) Yellow robot successfully delivers a Tool-0 to workshop-0.



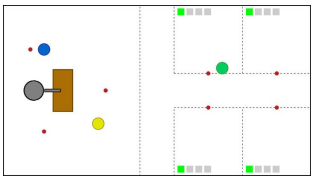
(h) Arm robot executes ***Pass-to-M(0)*** to pass a Tool-0 to green robot. Yellow robot runs ***Get-Tool*** to go back table.



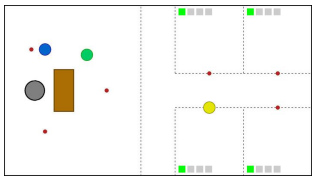
(i) Arm robot runs ***Search-Tool(0)*** to find the 3rd Tool-0. Green robot moves to workshop-1 by executing ***Go-W(1)***.



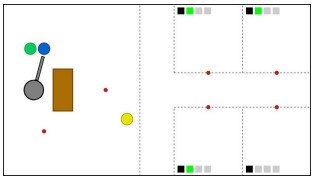
(j) Green robot successfully delivers the a Tool-0 to workshop-1. Arm robot executes ***Pass-to-M(2)*** to pass a Tool-0 to yellow robot.



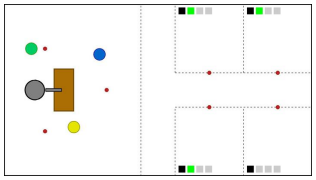
(k) Arm robot runs ***Search-Tool(1)*** to find the 1st Tool-1. Yellow robot moves to workshop-2 by executing ***Go-W(2)***. Green robot runs ***Get-Tool*** to go back table.



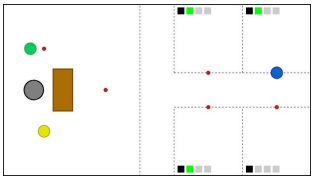
(l) Yellow robot successfully delivers a Tool-0 to workshop-2.



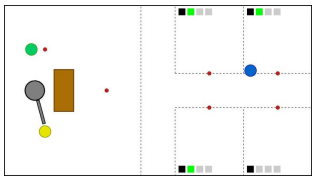
(m) Yellow robot runs ***Get-Tool*** to go back table. Arm robot executes ***Pass-to-M(1)*** to pass the a Tool-1 to blue robot. Human-0, human-1 and human-2 finish subtask-0 and start to do subtask-1.



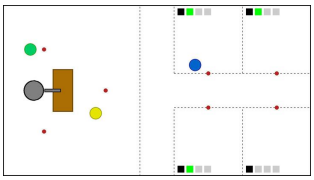
(n) Arm robot runs ***Search-Tool(1)*** to find the 2nd Tool-1. Blue robot moves to workshop-1 by executing ***Go-W(1)***.



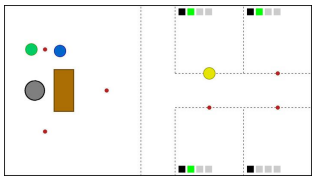
(o) Blue robot successfully delivers a Tool-1 to workshop-1.



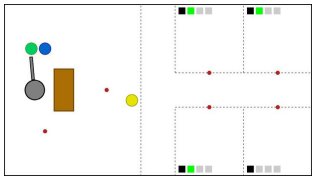
(p)Arm robot executes ***Pass-to-M(2)*** to pass a Tool-1 to yellow robot. Blue robot runs ***Get-Tool*** to go back table.



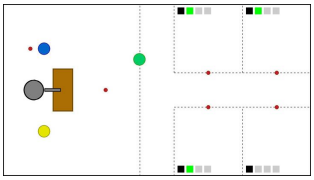
(q) Arm robot runs ***Search-Tool(0)*** to find 4th Tool-0. Yellow robot moves to workshop-0 by executing ***Go-W(0)***.



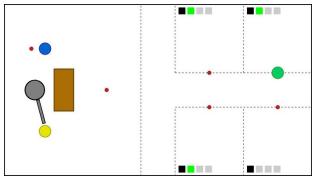
(r) Yellow robot successfully delivers a Tool-1 to workshop-0.



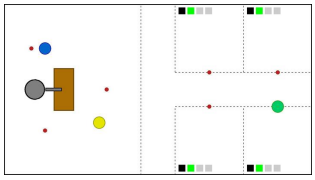
(s) Yellow robot runs ***Get-Tool*** to go back table. Arm robot executes ***Pass-to-M(0)*** to pass a Tool-0 to green robot.



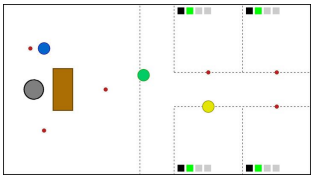
(t) Arm robot runs ***Search-Tool(1)*** to find the 3rd Tool-1. Green robot moves to workshop-1 by executing ***Go-W(1)***.



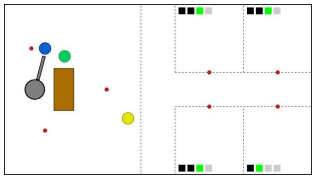
(u) Green robot reaches workshop-1 and observes that human-1 does not need Tool-0 and it moves to workshop-3 by executing ***Go-W(3)***. Arm robot executes ***Pass-to-M(2)*** to pass a Tool-1 to yellow robot.



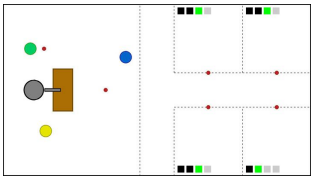
(v) Arm robot runs ***Search-Tool(1)*** to find the 4th Tool-1. Green robot successfully delivers a Tool-0 to workshop-3. Human-3 finishes subtask-0 and starts to do subtask-1. Yellow robot moves to workshop-2 by executing ***Go-W(2)***.



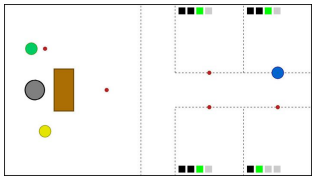
(w) Yellow robot successfully delivers a Tool-1 to workshop-2. Green robot runs ***Get-Tool*** to go back table.



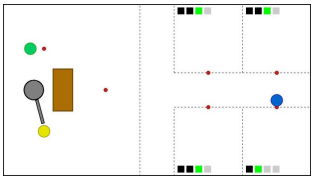
(x) Arm robot executes ***Pass-to-M(1)*** to pass a Tool-1 to blue robot. Yellow robot runs ***Get-Tool*** to go back table.



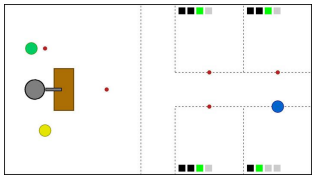
(y) Arm robot runs ***Search-Tool(2)*** to find the 1st Tool-2. Blue robot moves to workshop-1 by executing ***Go-W(1)***.



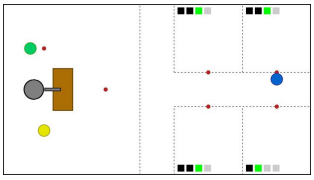
(z) Blue robot reaches workshop-1 and observes that human-1 does not need Tool-1.



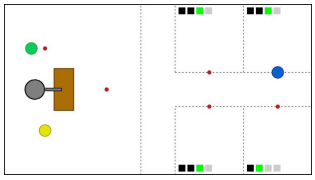
(A) Arm robot executes ***Pass-to-M(2)*** to pass a Tool-2 to yellow robot. Blue robot moves to workshop-3 by executing ***Go-W(3)***.



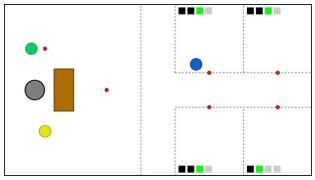
(B) Arm robot runs ***Search-Tool(2)*** to find the 2nd Tool-2. Blue robot successfully delivers a Tool-1 to workshop-3.



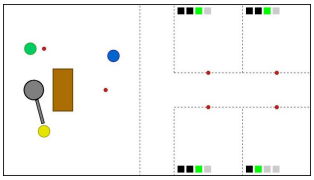
(C) Blue robot moves to workshop-1 by executing ***Go-W(1)***.



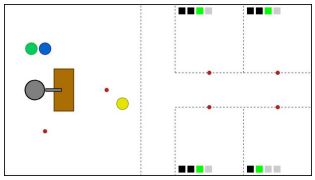
(D) Blue robot reaches workshop-1.



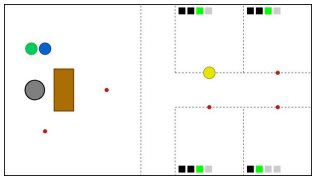
(E) Blue robot runs ***Get-Tool*** to go back table.



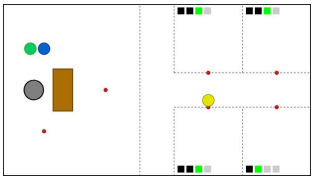
(F) Arm robot executes ***Pass-to-M(2)*** to pass the a Tool-2 to yellow robot.



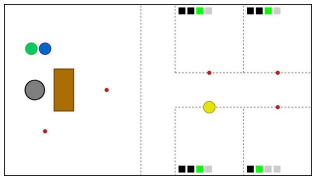
(G) Arm robot runs ***Search-Tool(2)*** to find the 3rd Tool-2. Yellow robot moves to workshop-1 by executing ***Go-W(1)***.



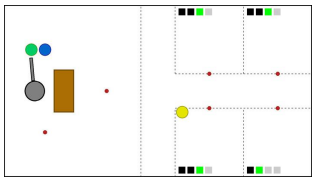
(H) Yellow robot reaches workshop-0 and observes that human-0 has got a Tool-2.



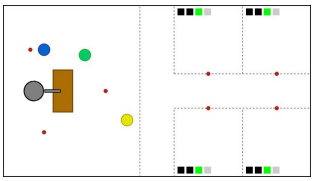
(I) Yellow robot moves to workshop-2 by executing ***Go-W(2)***.



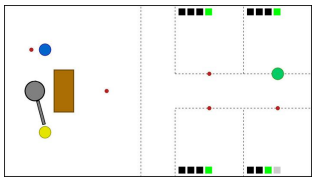
(J) Yellow robot successfully delivers a Tool-2 to workshop-2.



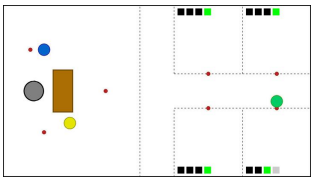
(K) Arm robot executes ***Pass-to-M(0)*** to pass a Tool-2 to green robot. Yellow robot runs ***Get-Tool*** to go back table.



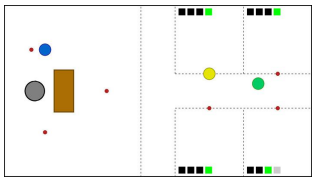
(L) Arm robot runs ***Search-Tool(2)*** to find the 4th Tool-2. Green robot moves to workshop-1 by executing ***Go-W(1)***.



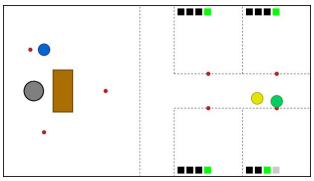
(M) Arm robot executes ***Pass-to-M(2)*** to pass a Tool-2 to yellow robot. Green robot successfully delivers a Tool-2 to workshop-1. Human-0, human-1 and human-2 finish subtask-2 and starts to do subtask-3.



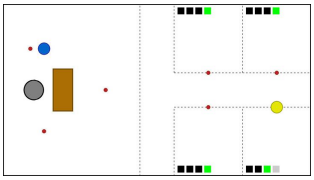
(N) Yellow robot moves to workshop-0 by executing ***Go-W(0)***. Green robot moves to workshop-3 by executing ***Go-W(3)***.



(O) Yellow robot reaches workshop-0 and observes that human-0 does not need Tool-2.

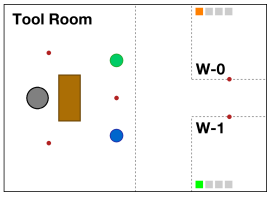


(P) Yellow and green robot move to workshop-3 by executing ***Go-W(3)***.

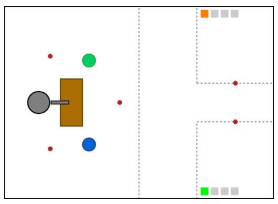


(Q) Yellow robot successfully delivers a Tool-2 to workshop-3. Humans have received all tools, and for robots, the task is done.

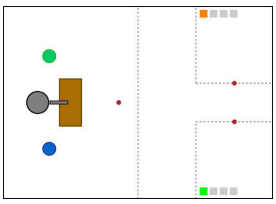
Warehouse-E:



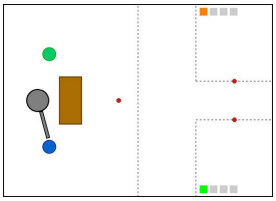
(a) Initial State.



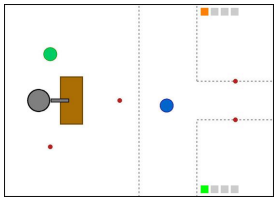
(b) Mobile robots moves towards the table by running ***Get-Tool***, and arm robot runs ***Search-Tool(0)*** to find Tool-0.



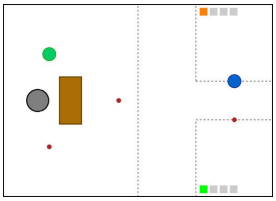
(c) Mobile robots wait there and arm robot keeps looking for Tool-0.



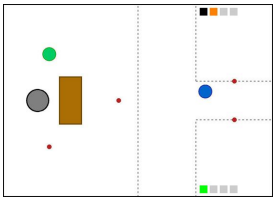
(d) Arm robot executes ***Pass-to-M(1)*** to pass Tool-0 to the blue robot.



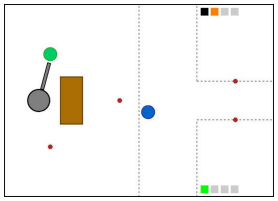
(e) Arm robot runs ***Search-Tool(1)*** to find Tool-1. Blue robot executes ***Go-W(0)*** to go to workshop-0.



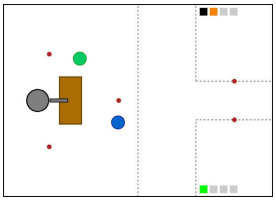
(f) Blue robot successfully delivers Tool-0 to workshop-0.



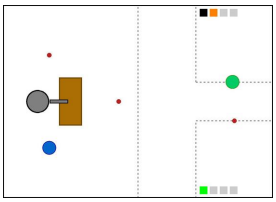
(g) Blue robot runs ***Get-Tool*** to go back table. Human-0 finishes subtask-0 and starts to do subtask-1.



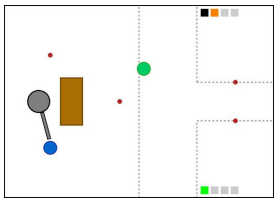
(h) Arm robot executes ***Pass-to-M(0)*** to pass Tool-1 to green robot.



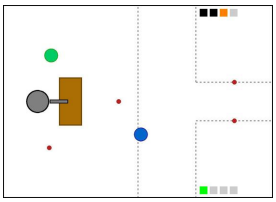
(i) Arm robot runs ***Search-Tool(0)*** to find Tool-0. Green robot moves to workshop-0 by executing ***Go-W(0)***.



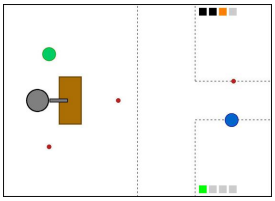
(j) Green robot successfully delivers Tool-1 to workshop-0.



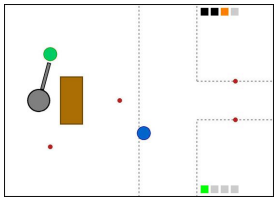
(k) Arm robot executes ***Pass-to-M(1)*** to pass Tool-0 to blue robot. Green robot runs ***Get-Tool*** to go back table.



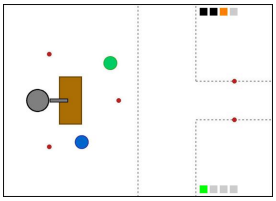
(l) Arm robot runs ***Search-Tool(2)*** to find Tool-2. Blue robot moves to workshop-1 by executing ***Go-W(1)***. Human-0 finishes subtask-1 and starts to do subtask-2.



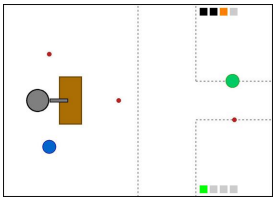
(m) Blue robot successfully delivers Tool-0 to workshop-1.



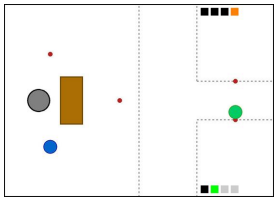
(n) Arm robot executes ***Pass-to-M(0)*** to pass Tool-2 to green robot. Blue robot runs ***Get-Tool*** to go back table.



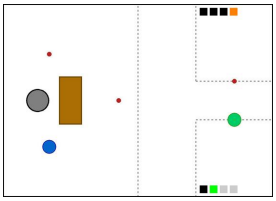
(o) Arm robot runs ***Search-Tool(1)*** to find Tool-1. Green robot moves to workshop-0 by executing ***Go-W(0)***.



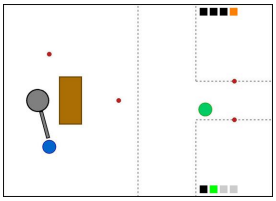
(p) Green robot successfully delivers Tool-2 to workshop-0.



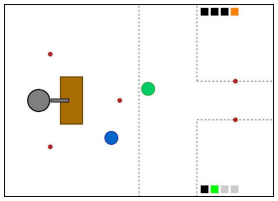
(q) Green robot moves to workshop-1 by executing ***Go-W(1)*** to observe human-1's status. Human-0 finishes subtask-2 and starts to do subtask-3.



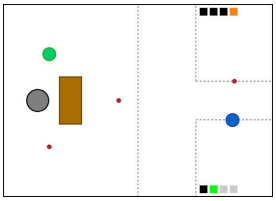
(r) Green robot reaches workshop-1.



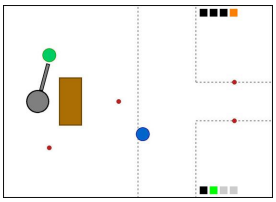
(s) Arm robot executes ***Pass-to-M(1)*** to pass Tool-1 to blue robot. Green robot runs ***Get-Tool*** to go back table.



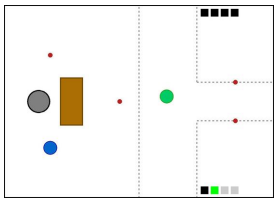
(t) Arm robot runs ***Search-Tool(2)*** to find Tool-2. Blue robot moves to workshop-1 by executing ***Go-W(1)***.



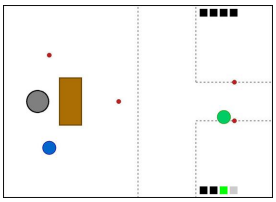
(u) Blue robot successfully delivers Tool-1 to workshop-1.



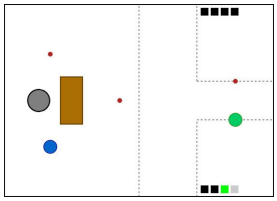
(v) Blue robot runs ***Get-Tool*** to go back table. Arm robot executes ***Pass-to-M(0)*** to pass Tool-2 to green robot. Blue robot runs ***Get-Tool*** to go back table.



(w) Green robot moves to workshop-1 by executing ***Go-W(1)***.



(x) Human-1 finishes subtask-1 and start to do subtask-2.



(y) Green robot successfully delivers Tool-2 to workshop-1. Humans have received all tools, and for robots, the task is done.