intelligent, enabling the easy creation of different multi-agent scenarios. For example, in the sample game in Fig. 2 (a), after defining the basic behavior of the agent (i.e., moving and turning) and the “alive” state of the agent (i.e., it stays on the playground), it can be extended to different multi-agent scenarios with minimal effort. For example, (1) five players fight each other until only one agent is left alive (see Fig. 2 (b)), or (2) $5 \times 2$ players form 2 teams, and each agent fights for its own team until all players in a team are dead (see Fig. 2 (c)), or (3) multiple players form multiple teams in hierarchies, where the collaboration and competition relationships between the teams are customized (see Fig. 3).

Thus, Arena is not just a research platform for the evaluation with a fixed set of games, but also a building toolkit for researchers to invent and build novel multi-agent problems.

To achieve the above vision of building a toolkit for multi-agent intelligence, (1) we provide a GUI-configurable tree that defines the social structure of agents, called *social tree*;
and (2) based on the social tree, we propose 5 basic multi-agent reward schemes (BMaRSs), which define different social paradigms at each node in the social tree. Specifically, each BMaRS is a restriction applied to the reward function, so it corresponds to a batch of reward functions that can lead to a specific social paradigm. For each BMaRS, Arena provides multiple ready-to-use reward functions, simplifying the construction of games with complex social relationships. Furthermore, if the agent is controlling each joint of a robot, it has long been a burden for researchers that low-level intelligence (such as the basic skill of moving) must first be built, before they can study high-level multi-agent intelligence (Heess et al. 2017). Thus, Arena provides many ready-to-use dense reward functions in each BMaRS that handle such low-level intelligence. Additionally, Arena also offers a verification option for customized reward functions, so the researchers can make sure that the programmed reward functions lie in one of the BMaRSs that produces a specific social paradigm. Thus, with the above efforts towards a building toolkit for multi-agent intelligence and the provided set of 35 games for a general evaluation platform, one can easily customize a set of games of a new social paradigm to study a yet unexplored problem.

Finally, we provide Python implementations of several state-of-the-art deep multi-agent reinforcement learning baselines, which can be used as starting points for the development of novel multi-agent algorithms, as well as the validation of new environments. Along with the baseline implementations, we also release a set of 100 best agents/teams that we can train with different training schemes for each game, as the base for evaluating agents with population performance (Balduzzi et al. 2019; 2018). So, the research community can perform comparisons under a stable and uniform standard.

To summarize, this paper’s contributions are as follows:

1. a general evaluation platform for multi-agent intelligence with a set of diverse games, most of which are new to the community or still stand as a challenge for state-of-the-art algorithms,
2. a building toolkit for multi-agent games, enabling the easy creation of new social paradigms based on GUI-configurable social trees and BMaRSs,
3. the baseline implementations of 5 state-of-the-art multi-agent algorithms for both competitive and collaborative settings, and
4. sets of benchmark agents/teams for the community to conduct stable and uniform population evaluation (Balduzzi et al. 2018). Code for games, building toolkit, and baselines, as well as all corresponding tutorials have been released online at https://sites.google.com/view/arena-unity/.

The Platform

**State-of-the-Art Engine.** The engine behind Arena is the world-leading game engine Unity (Juliani et al. 2018), which provides Arena with several desirable features on rendering, physics, customizability, and community. There are also other choices of popular engines. Some platforms contain a wide set of diverse games (Bellemare et al. 2013; Nichol et al. 2018; Perez-Liebana et al. 2016; OpenAI 2016). However, they are designed mostly for single-agent scenarios and are extremely hard to customize (adding multiple players or creating new games), since the games are provided as compiled binary ROMs. Other downsides of these choices include deterministic environments, unrealistic rendering, and unrealistic physics. Other platforms (Todorov, Erez, and Tassa 2012; Tassa et al. 2018) are, in nature, more physics engines than game engines, which lack a visual editor for easily creating customized games, and cannot handle more “game-like” features, such as instantiating and destroying objects in real-time during the simulation. The rest of the platforms are limited in the sense that they are built for specific tasks, such as for first-person shoot.
Building Toolkit for Multi-Agent Environments. As the second contribution, we provide a building toolkit for multi-agent environments: we provide (1) a GUI-configurable social tree that defines how agents are grouped together with each other, and (2) 5 basic multi-agent reward schemes (BMaRSs) applied on each node in the social tree, so that different social relationships can be easily built and verified, and low-level intelligence (like motor skills) can be handled.

Other Features. Learning to communicate is an important research area in multi-agent intelligence (Das et al. 2017; Mordatch and Abbeel 2018). Thus, Arena provides a broadcast board at each node of the social tree (accessible for any agent as a child of the node), which enables the study of learning communication at each level. Also, global states may be used in research for different purposes (Lowe et al. 2017; Gupta, Egorov, and Kochenderfer 2017; Foerster et al. 2017; Wu et al. 2018; Savva et al. 2019). Thus, Arena provides the option to broadcast it to all agents. Besides, a top-down view of the global game is often appreciated for visualizing population behavior (Johnson et al. 2016; Wydmuch, Kempka, and Jaśkowski 2018). Thus, Arena by default enables this option. Finally, there is a necessity for competitive agents to evaluate against human players, and also a research trend for collaborative agents to team up with human players. Thus, Arena provides a gaming interface for humans, so that a human player can take the place of any agent in the game.

Basic Multi-Agent Reward Schemes and Social Trees

Preliminaries. We consider a Markov game as defined in (Littman 1994), consisting of multiple agents \( x \in \mathcal{X} \), a finite global state space \( \mathcal{S} \), a finite action space \( \mathcal{A}_x \) for each agent \( x \), and a bounded-step reward space \( r_{x,t} \in \mathbb{R} \) for each agent \( x \). The environment consists of a transition function \( g : \mathcal{S} \times \mathcal{A}_x \times \mathcal{X} \rightarrow \mathcal{S} \), which is a stochastic function \( s_{t+1} \sim g(s_t, (a_{x,t})_{x \in \mathcal{X}}) \), a reward function for each agent \( f_x : \mathcal{S} \times \mathcal{X} \rightarrow \mathbb{R} \), which is a deterministic function \( r_{x,t} = f_x(s_t, (a_{x,t})_{x \in \mathcal{X}}) \), a joint reward function \( f = (f_x)_{x \in \mathcal{X}} \), and episode reward \( R^e = \sum_{t=1}^T r_{x,t} \) for each agent \( x \) under the joint reward function \( f \). For the agent, we consider that it observes \( s_{x,t} \in \mathcal{S}_x \), where \( \mathcal{S}_x \) consists of a part of the information from the global state space \( \mathcal{S} \). Thus, we have a policy \( \pi_x : \mathcal{S}_x \rightarrow \mathcal{A}_x \), which is a stochastic function \( a_{x,t} \sim \pi_x(s_{x,t}) \). Besides, we consider that the agent \( x \) can take a policy \( \pi_x \) from a set of policies \( \Pi_x \) and assume...
that the random seed of all sampling operations is \( k \), which is sampled from the whole seed set \( \mathcal{K} \).

We investigate the effect of \( \{ x : x \in \mathcal{X} \} \) and \( \{ \pi_x : x \in \Pi \} \) on \( \{ R^i_x : x \in \mathcal{X} \} \). By applying different restrictions on the effect, we have different BMaRSs, each one of which is a set of joint reward functions \( \mathcal{F} = \{ f : \cdot \} \) that produce a similar effect on the population \( \mathcal{X} \). The term reward scheme first appears in (Tampuu et al. 2017) as a tabular, which is applied to a special case of Pong. While we define it in a general form and show that many examples are special cases within this general form.

In a non-sequential setting (normal-form game), the reward scheme serves a similar purpose as the payoff matrix (Myerson 2013), which is also represented as a tabular. See Lemma 2 in the extended paper (Song et al. 2019) for how the payoff matrix is aligned with BMaRSs. In the following, we define 5 different BMaRSs. Along defining these BMaRSs, we also describe the ready-to-use reward functions \( f \) within these BMaRSs, which is provided by Arena as a dropdown list.

**Non-learnable** BMaRSs \( (\mathcal{F}^{NL}) \) are a set of joint reward functions \( f \) as follows:

\[
\mathcal{F}^{NL} = \{ f : \forall k \in \mathcal{K}, \forall x \in \mathcal{X}, \forall \pi_x \in \Pi_x, \frac{\partial R^i_x}{\partial \pi_x} = 0 \},
\]

where \( 0 \) is a zero matrix of the same size and shape as the parameter space that defines \( \pi_x \). Intuitively, \( \mathcal{F}^{NL} \) means that \( R^i_x \) for any agent \( x \in \mathcal{X} \) cannot be optimized by improving its policy \( \pi_x \).

**Isolated** BMaRSs \( (\mathcal{F}^{IS}) \) are a set of joint reward functions \( f \) as follows:

\[
\mathcal{F}^{IS} = \{ f : f \notin \mathcal{F}^{NL} \text{ and } \forall k \in \mathcal{K}, \forall x \in \mathcal{X}, \forall x' \in \mathcal{X} \setminus \{ x \}, \forall \pi_x \in \Pi_x, \forall \pi_{x'} \in \Pi_{x'}, \frac{\partial R^i_x}{\partial \pi_{x'}} = 0 \},
\]

Intuitively, \( \mathcal{F}^{IS} \) means that the episode reward \( R^i_x \) received by any agent \( x \in \mathcal{X} \) is not related to any policy \( \pi_{x'} \) taken by any other agent \( x' \in \mathcal{X} \setminus \{ x \} \).

Reward functions \( f \) in \( \mathcal{F}^{IS} \) are often called *internal reward functions* in other multi-agent approaches (Hendellass 2004; Jaderberg et al. 2018; Bansal et al. 2018), meaning that apart from the reward functions applied at a population level (such as win/loss), which are too sparse to learn, there are also reward functions directing the learning process towards receiving the population-level rewards, but are more frequently available, i.e., more dense (Heess et al. 2017). \( \mathcal{F}^{IS} \) is especially practical if the agent is a robot requiring continuous control of applying force on each of its joints, which means basic motor skills (such as moving) need to be learned before generating population-level intelligence. Thus, we provide \( f \) in \( \mathcal{F}^{IS} \) of energy cost, punishment of applying a big force, encouragement of keeping a steady velocity, and moving distance towards target.

**Competitive** BMaRSs \( (\mathcal{F}^{CP}) \), inspired by (Cai and Daskalakis 2011), are defined as

\[
\mathcal{F}^{CP} = \{ f : f \notin \mathcal{F}^{NL} \cup \mathcal{F}^{IS} \text{ and } \forall k \in \mathcal{K}, \forall x \in \mathcal{X}, \forall \pi_x \in \Pi_x, \forall \pi_{x'} \in \Pi_{x'}, \frac{\partial R^i_x}{\partial \pi_x} = 0 \},
\]

which intuitively means that for any agent \( x \in \mathcal{X} \), taking any possible policy \( \pi_x \in \Pi_x \), the sum of the episode reward of all agents will not change. If the episode length is 1, it expresses a classic multi-player zero-sum game (Cai and Daskalakis 2011). Useful examples of \( f \) within \( \mathcal{F}^{CP} \) are: (1) agents fight for a limited amount of resources that are always exhausted at the end of the episode, and the agents are rewarded for the amount of resources that it gained, and (2) fight till death, and the reward is given based on the order of death (the reward can also be based on the reversed order, so that the one departing the game first receives the highest reward, such as in some poker games, the one who first discards all cards wins). *Rock, Paper, and Scissors* in normal-form game (Myerson 2013) and *Cyclic Game* in (Balduzzi et al. 2019) are both special cases of \( \mathcal{F}^{CP} \); see Lemmas 2 and 3 in the extended paper (Song et al. 2019).

**Collaborative** BMaRSs \( (\mathcal{F}^{CL}) \), inspired by (Cai and Daskalakis 2011), are defined as

\[
\mathcal{F}^{CL} = \{ f : f \notin \mathcal{F}^{NL} \cup \mathcal{F}^{IS} \text{ and } \forall k \in \mathcal{K}, \forall x \in \mathcal{X}, \forall x' \in \mathcal{X} \setminus \{ x \}, \forall \pi_x \in \Pi_x, \forall \pi_{x'} \in \Pi_{x'}, \frac{\partial R^i_x}{\partial \pi_x} = 0 \},
\]

which intuitively means that there is no conflict of interest \( (\frac{\partial R^i_x}{\partial \pi_x} < 0) \) for any pair of agents \( \{ x, x' \} \). Besides, since \( f \notin \mathcal{F}^{NL} \cup \mathcal{F}^{IS} \), there is at least one pair of agents \( \{ x, x' \} \) that makes \( \frac{\partial R^i_x}{\partial \pi_x} > 0 \). This indicates that this pair of agents shares a common interest, so that improving \( R^i_x \) for agent \( x \) means improving \( R^i_{x'} \) for agent \( x' \). The most common example of \( f \) within \( \mathcal{F}^{CP} \) is that \( f_{ex} \) for all \( x \in \mathcal{X} \) is identical, such as the moving distance of an object that can be pushed forward by the joint effort of multiple agents, or the alive duration of the population (as long as there is at least one agent alive in the population, the population is alive). Thus, we provide \( f \) in \( \mathcal{F}^{CL} \): living time of the team (both positive and negative, since some games require the team to survive as long as possible, while other games require the team to depart as early as possible, such as poker).

**Competitive and Collaborative Mixed** BMaRSs \( (\mathcal{F}^{CC}) \) are defined as a catch-all for any other than the above four ones. First, the term \( \frac{\partial \sum_{x \in \mathcal{X}} R^i_x dx'}{\partial \pi_x} = 0 \) in (3) can be written as \( \sum_{x \in \mathcal{X}} \frac{\partial R^i_x}{\partial \pi_x} dx' = 0 \) (see Lemma 1 in extended paper (Song et al. 2019), which makes an alternative (3). Considering \( \mathcal{F}^{CP} \) in this alternative (3) and \( \mathcal{F}^{CL} \) in (4), an intuitive explanation of \( \mathcal{F}^{CC} \) is that there exist circumstances when \( \frac{\partial R^i_x}{\partial \pi_x} < 0 \), meaning that the agents are competitive at this point. But the derivative of total interest \( \sum_{x \in \mathcal{X}} \frac{\partial R^i_x}{\partial \pi_x} dx' \) is not always 0; thus, the total interest can be maximized with specific policies, meaning that the agents are collaborative at this point.
Apart from providing several practical $f$ in each BMaRS, we also provide a verification option for each BMaRS, meaning that one can customize an $f$ and use this verification option to make sure that the programmed $f$ lies in a specific BMaRS. The implementation of verification option can be found in Section 1 in the extended paper (Song et al. 2019).

The Social Tree. The BMaRSs defined above apply to an agent group of all sizes. To define more complex and structured social paradigms, we use a tree structure (social tree) to organize the agents and apply BMaRSs on each node of the tree. We illustrate this by an example. The GUI interface in Fig. 3 (a) defines a tree structure in Fig. 3 (b), representing a population of 4 agents.

The tree structure can be easily reconfigured by dragging, duplicating, or deleting nodes in the GUI interface in Fig. 3 (a). In this example, each agent has an agent-level BMaRS. The agent is a robot ant, so that the agent-level BMaRSs are $F_{IS}$, specifically, the option of ant-motion that directs the learning towards basic motion skills such as moving forward, as shown in Fig. 3 (c). Each two agents form a team (which is a set of agents or teams), the two agents have team-level BMaRSs. In this example, the two robot ants collaborate with each other to push a box forward, as shown in Fig. 3 (d). Thus, the team-level BMaRSs are $F_{CL}$, specifically, the moving distance of the box. On the two teams, we have global-level BMaRSs. In this example, the two teams are set to have a match regarding which team pushes its box to the target point first, as shown in Fig. 3 (e). Thus, the global-level BMaRSs are $F_{CP}$, specifically, the ranking of the box reaching the target. The final reward function applied to each agent is a weighted sum of the above three BMaRSs at three levels. One can imagine defining a social tree of more than three levels, where small teams form bigger teams, and BMaRSs are defined at each node to give more complex and structured social problems. After defining the social tree and applying BMaRSs on each node, the environment is ready for use with an abstraction layer handling everything else, such as assigning viewports to each agent in the window, applying the team color, displaying the agent ID, and generating a top-down view.

The Learning Agents

The Baselines. We provide Python implementations of several state-of-the-art baselines that can be used as starting points for the development of novel multi-agent algorithms, as well as for the validation of new environments. Specifically, we first implement a fully decentralized system, where each agent is a self-contained PPO (Schulman et al. 2017), with independent critic, actor, and optimizer. We also implement two state-of-the-art methods based on self-play in (OpenAI 2018) (SP) and population-based training in (Jaderberg et al. 2018; DeepMind 2019) (PB). For collaborative agents, we implement two state of the arts: centralized critic (Lowe et al. 2017) (CC) and centralized critic with a counterfactual baseline (Foerster et al. 2018) (CF).

The Evaluation Metric. It is recently raising attention that evaluating an agent against a single-agent or hand-coded bot is unstable and misleading (Balduzzi et al. 2018). Thus, the population performance is introduced to evaluate an agent’s (or an agent group’s) performance among a base population. To enable population evaluation, we release 100 best agents, which we can train with different training schemes for each game as the base population. One can call the provided function to get the ranking of an agent among the base population, or get the averaged ranking of a population among the base population. Moreover, we provide a human ranking among the base population, which provides an indication of human-level intelligence in the game. We will accept the submission of agents from the community as well as keep implementing algorithms introduced in the future, so that the base population will be upgraded, as the level of research in multi-agent intelligence advances.

Experiments

Experiments are conducted from three aspects. First, we evaluate our game set from the perspective of stochasticity, realistic rendering, and simulation speed, all of which are presented in the extended paper (Song et al. 2019) due to page limit. Other advantages from the Unity engine have been verified by (Juliani et al. 2018). Second, we evaluate our design of the extensible multi-agent building toolkit with a case study, showing that by applying different social trees and BMaRSs, different population-level strategies can be learned. Third, we report the experimental results of 5 baselines that we implemented and show that by using the provided population performance evaluation metric, the training progress can be visualized in a less noisy and more analyzable way.

Case Study of Social Tree and BMaRSs. We use the game Crossroads from Arena to study the effectiveness of
Figure 4: Visualizing training progress over episode reward (a,b) and population performance (c,d) of different baselines: D-PPO (Decentralized Proximal Policy Optimization), SP (Self-Play), PB (Population-Based training), CC (Centralized Critic), and CF (Counterfactual Baseline).

Baselines and Evaluation Metric. We compare 5 baselines on two games: (1) Crossroads in Fig. 5 (a) with the BMaRS settings of Fig. 5 (d) and (2) PushBox in Fig. 3 (e) with the BMaRS settings of Fig. 3 (b). The BMaRS settings of both games contain competitive as well as collaborative social relationships, i.e., multiple agents form collaborative teams, and teams compete with each other. Thus, we investigate SP and PB baselines at the level of teams competing with each other, as well as investigate CC and CF baselines at the level of agents collaborating with each other in a team. As can be seen, the curve of episode reward shown in Fig. 4 (a) and (b) is extremely noisy, as the environment is non-stationary with the strategy of other collaborators and/or competitors evolving during the training. However, in Fig. 4 (c) and (d), which is the curve of ranking in the released base population, i.e., population performance, all methods are comparable with clear performance gaps.

Related Work

Surveys of multi-agent intelligence research can be found in (Hernandez-Leal, Kartal, and Taylor 2018). Different ideas have been explored on competitive and collaborative multi-agent settings.

Collaborative Settings. The simplest way to deploy multi-agent collaborative systems is to make each agent have a completely independent learning process (fully decentralized) (Matignon, Laurent, and Le Fort-Piat 2012). However, collaborative behaviors are hardly observed under such fully decentralized setting; thus, a fully centralized system is utilized in (Peng et al. 2017), where the policy has access to the global state and is shared by all agents. However, it is impractical, since the global state is mostly unavailable in practice, and the system does not support extending the number of agents. Thus, centralized training and decentralized execution are gaining attention (Kraemer and Banerjee 2016). For multi-agent systems, this idea is mostly explored under actor-critic algorithms (Foerster et al. 2018). Other ideas include using a joint action-value function, (Lauer and Riedmiller 2004) addressing the variance problem by a large batch size (Bansal et al. 2018), and learning grounded cooperative communication protocols between agents (Foerster et al. 2016).

Competitive Settings. Competitive multi-agent intelligence originally comes from computational game theory

ture games, such as Poker and Go (Moravčík et al. 2017; Silver et al. 2017). Later, D-MARL was applied to more diverse problems, such as high-dimensional video games (OpenAI 2018; DeepMind 2019) and those involving physics control (Bansal et al. 2018). When solving more practical problems, many issues have been raised, such as ensuring diversity amongst agents (Marivate 2015), avoiding overfitting to the policy of the opponents (Lanctot et al. 2017). Many ideas address such issues (Kleiman-Weiner et al. 2016). Following on D-MARL, a very promising recent direction is self-play (Tesauro 1995). Fictitious self-play (Heinrich and Silver 2016) first shows promising performance on the competitive game Leduc Poker. However, as the stability and parallelizability are improving with the invention of new reinforcement learning algorithms, state-of-the-art approaches adopt a simpler form of self-play (OpenAI 2018), which produces a superior-human intelligence on large video games, like Dota2. Another promising recent idea is population-based training, as adopted in StarCraft (DeepMind 2019).

Summary and Outlook
This paper has introduced the first general evaluation platform for multi-agent intelligence research. Besides, with the efforts on a building toolkit of multi-agent environments, the platform also allows for easily building new multi-agent problems. Additionally, with the released implementations of several state-of-the-art baselines, researchers can start their adventure instantly. Finally, by releasing a base population, the community can conduct comparisons under a stable and uniform evaluation metric.

Acknowledgments. This work was supported by the China Scholarship Council under the State Scholarship Fund, by the Graduate Travel and Special Project Grants from the Somerville College of the University of Oxford, by the Alan Turing Institute under the UK EPSRC grant EP/N510129/1, by the AXA Research Fund, by the National Natural Science Foundation of China under the grants 61906063, 61876013, and 61922009, by the Natural Science Foundation of Tianjin under the grant 19JCQNJC00400, by the “100 Talents Plan” of Hebei Province, and by the Yuan- guang Scholar Fund of Hebei University of Technology. The Arcade learning environment: An evaluation platform for general agents. JAIR. Bowling, M.; Burch, N.; Johanson, M.; and Tammelin, O. 2015. Heads-up limit hold’em poker is solved. Science. Brodeur, S.; Perez, E.; Anand, A.; Golemo, F.; Celotti, L.; Strub, F.; Rouat, J.; Larochelle, H.; and Courville, A. 2017. Home: A household multimodal environment. arXiv:1711.11017. Cai, Y., and Daskalakis, C. 2011. On minmax theorems for multiplayer games. In SIAM.

References


Mordatch, I., and Abbeel, P. 2018. Emergence of grounded compositional language in multi-agent populations. In AAAI.


Tian, Y.; Gong, Q.; Shang, W.; Wu, Y.; and Zitnick, C. L. 2017. ELF: An extensive, lightweight and flexible research platform for real-time strategy games. In NIPS.


