TiZero: Mastering Multi-Agent Football with Curriculum Learning and Self-Play

Fanqi Lin
Tsinghua University
Beijing, China
lfq20@mails.tsinghua.edu.cn

Shiyu Huang
4Paradigm Inc.
Beijing, China
huangshiyu@4paradigm.com

Tim Pearce
Microsoft Research
London, United Kingdom
tim.pearce@microsoft.com

Wenze Chen
Tsinghua University
Beijing, China
cwz19@mails.tsinghua.edu.cn

Wei-Wei Tu
4Paradigm Inc.
Beijing, China
tuweiwei@4paradigm.com

ABSTRACT

Multi-agent football poses an unsolved challenge in AI research. Existing work has focused on tackling simplified scenarios of the game, or else leveraging expert demonstrations. In this paper, we develop a multi-agent system to play the full 11 vs. 11 game mode, without demonstrations. This game mode contains aspects that present major challenges to modern reinforcement learning algorithms; **multi-agent coordination**, **long-term planning**, and **non-transitivity**. To address these challenges, we present TiZero; a self-evolving, multi-agent system that learns **from scratch**. TiZero introduces several innovations, including adaptive curriculum learning, a novel self-play strategy, and an objective that optimizes the policies of multiple agents jointly. Experimentally, it outperforms previous systems by a large margin on the Google Research Football environment, increasing win rates by over 30%. To demonstrate the generality of TiZero's innovations, they are assessed on several environments beyond football; Overcooked, Multi-agent Particle-Environment, Tic-Tac-Toe and Connect-Four.

KEYWORDS

Multi-agent Reinforcement Learning; Self-play; Google Research Football; Large-scale Training

Figure 1: Screenshot of Google Research Football. TiZero's agents outperform previous systems by leveraging more coordinated strategies, such as passing more often and creating more assists.

1 INTRODUCTION

Deep reinforcement learning (DRL) has achieved great success in many games, including Atari classics [2, 22, 37], first-person-shooters [20, 24, 42], real-time-strategy titles [4, 55, 63], board games [48, 50] and card games [15, 65]. Yet modern DRL systems still struggle in environments containing challenges such as multi-agent coordination [43, 61, 64], long-term planning [9, 54, 66] and non-transitivity [3, 8]. The Google Research Football environment (GFootball) contains all these challenges [28] and more. This paper

presents the first system that successfully deals with all of them, learning to play the full 11 vs. 11 game mode from scratch. Experimentally, our method outperforms previous systems by a large margin with over 30% higher winning rates.

GFootball has attracted the attention of many DRL researchers as it provides a test-bed for complex multi-agent control [12, 19, 30, 58, 61]. As in the popular real-world sport of football/soccer, to win at GFootball agents must combine short-term control techniques with coordinated, long-term global strategies. Challenges in GFootball include multi-agent cooperation, multi-agent competition, sparse rewards, large action space, large observation space and stochastic environments – a combination not present in other RL research environments. In GFootball, each agent needs both to cooperate with teammates, and compete against diverse and unknown opponents. Agents without the ball are hard to optimize as they do not obtain dense rewards, thus resulting in challenging multi-agent credit assignment problem [10, 11, 53, 68]. Moreover, GFootball transitions are stochastic, e.g. performing the same shooting action from the same state may sometimes result in a goal and sometimes a miss. Table 1 contrasts GFootball with other popular RL environments, illustrating its complexity.

Prior work on GFootball has mostly focused on 'Academy' scenarios [12, 30, 58, 61], which are drastically simplified versions of

Table 1: Comparison of the complexity of current popular DRL benchmarks. GFootball 11 vs. 11 presents an increased level of complexity – it combines competitive elements, stochasticity, a large number of agents, a long game horizon and sparse rewards.

Citation	Game	Competitive?	Agent number	Stochastic?	Sparse reward?	Game length magnitude
[22, 37]	Atari Games	✗	1	✗	Sometimes	Typically 10^2
[48]	Go	✓	1	✗	✓	10^2
[7]	Procgen	✗	1	✓	✗	10^2
[13, 63]	Honor of Kings	✓	5	✗	✗	$\mathbf{10^3}$
[12, 30, 58, 61]	GFootball Academy	✗	< 10	✓	✓	10^2
Our work	GFootball 11 vs. 11	✓	**10**	✓	✓	$\mathbf{10^3}$

the full 11 vs. 11 game, for instance requiring agents to score in an empty goal or beat the goalkeeper in a 1-on-1. Any opponents are rules-based bots. In contrast, our work introduces an AI system that plays on the full 11 vs. 11 game mode. This mode requires simultaneous control of ten players. (The goalkeeper is excluded since they have a unique action space and different purpose which would require training of a separate policy. Instead they are controlled via a simple rule-based strategy.) Each player performs 3, 000 steps per match, meaning long-term planning is required. Due to these challenges, previous work tackling the 11 vs. 11 game mode relied on an offline demonstration dataset [19]. The disadvantage of such methods is that the performance of trained agents is bounded by the skill-level of the demonstrators. In this paper, we do not use any demonstration dataset nor any pre-trained model, instead training tabula-rasa via curriculum-learning and self-play.

This paper proposes a **multi-agent**, **curriculum**, **self-play** framework for GFootball, which we name TiZero, that is trained at mass-scale. At its core, it is an actor-critic algorithm [27]. To facilitate **multi-agent learning**, a single centralized value network guides learning of all agent policies in a joint optimization step, while allowing for decentralised execution at test time. To tackle the challenge of long-term planning in a sparse-reward setting, we borrow ideas from **curriculum learning** [47, 62, 67]. Here, the agent initially learns basic behaviors in simpler tasks, strengthens these in increasingly difficult scenarios, eventually enabling high-quality long-term planning. To address the challenge of non-transitivity (that is, player A>player B, player B>player C $\implies$ player A>player C) [3, 8], we develop a **self-play learning** strategy to manage the opponent pool, which ensures training is against a set of diverse and strong opponents. TiZero applies these algorithmic ideas in a large-scale distributed training infrastructure, training across hundreds of processors for about 40 days.

In addition to these core algorithmic ingredients, TiZero adopts a variety of modern DRL advances, including recurrent experience replay [23], action masking, reward shaping [38], agent-conditioned policy [12] and staggered resetting.

To evaluate the performance of TiZero, we compare against previous state-of-the-art methods [1, 19, 56, 69] on GFootball, outperforming them by a large margin. On a public evaluation platform of GFootball, TiZero also ranks first[1]. To understand the generality of

TiZero, we assess our system on several public benchmarks not related to football, such as [59], Multi-agent Particle-world (MPE) [35], Tic-Tac-Toe and Connect-Four, finding that it also shows promise in these new domains.

2 RELATED WORK

In this section, we briefly review relevant work in the areas of multi-agent reinforcement learning, competitive self-play and football game AI.

2.1 Multi-agent Reinforcement Learning

Multi-agent reinforcement learning (MARL) has received much research attention [43, 51, 53, 57, 64]. Recent algorithms usually focus on the *centralized training with decentralized execution* (CTDE) setup. In CTDE, the learning algorithm has access to the action-observation histories of all agents during training. But at execution, each agent only has access to its local action-observation history. There are two main kinds of MARL algorithms: 1) value-based algorithms, such as QMIX [43], VDN [53], QPLEX [57]; 2) policy-based algorithms, such as MADDPG [35], MAPPO [64], MAT [61]. As an example of a value-based MARL algorithm, QMIX uses a centralized Q-value network over the joint action-value function. It is designed in a way that constrains the joint Q-value to be monotonic with respect to each individual agent's Q-value (these are mixed in a additive but possibly non-linear function). This allows decentralized execution as each agent can independently perform an argmax operation over their individual Q-values. TiZero builds upon MAPPO [64], which adapts the standard single-agent PPO algorithm [46] to the multi-agent setting by learning individual policies conditioned on local observations, and a centralized value function based on a global state. Several tricks have proven important to stabilize training, including Generalized Advantage Estimation (GAE) [45], observation normalization, value clipping, and orthogonal initialization. MAPPO is competitive with many MARL algorithms such as MADDPG [35], RODE [60], and QMIX, both in terms of sample efficiency and wall-clock time. In this paper, we propose a new variant of MAPPO, which can improve multi-agent coordination and reduce the video memory usage via multi-agent joint-policy optimization.

---

[1]JiDi AI Competition Platform: http://www.jidiai.cn/ranking_list?tab=34. The evaluation result was collected on October 28th, 2022.
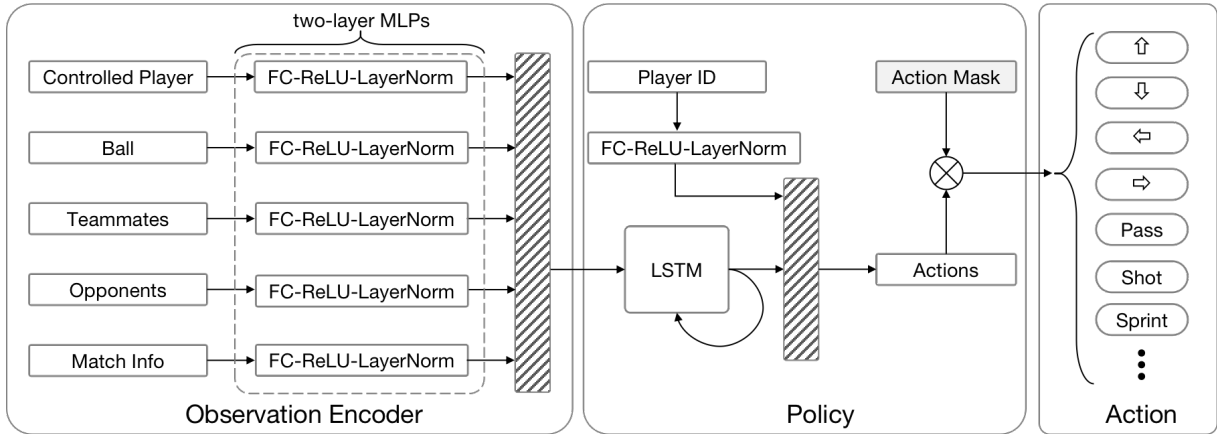
Figure 2: TiZero's network architecture. Six types of information are required as input: the controlled player information, player ID, ball information, teammate information, opponent information and current match information. We use six separate MLPs with two (one for the "player ID") fully-connected layers to encode each part of the observation. An LSTM layer is used to incorporate historic observations. The policy outputs a softmax distribution over the 19 discrete actions.

2.2 Competitive Self-play

Self-play has emerged as a powerful technique to obtain superhuman policies in competitive environments [4, 16, 29, 49, 55]. By playing against recent copies of itself, an agent can continuously improve its performance, avoiding any limitation that might otherwise be imposed by the skill-level of a demonstration dataset. Policy-Space Response Oracle (PSRO) is a population learning framework for learning best-response agents to a mixture of previous agents with a meta-strategy solver [29]. Under this framework, Vinalys et al. [55] proposed a league training framework to train robust agents for StarCraft ('prioritized fictitious self-play'). Meanwhile, OpenAI Five [4] achieved super-human performance on Dota 2 via a simpler self-play strategy – the current set of agents plays against the most recent set of agents with 80% probability, and plays against past agents with 20% probability. However, such empirical successes require huge computing resources, and the training process may become stuck due to the non-transitivity dilemma when strategic cycles exist [3, 8]. Recent work solves this problem by increasing the diversity in the pool of opponent policies [6, 33, 34]. This paper also introduces a novel self-play training strategy that improves the diversity and performance of opponent policies. We design a two-step self-play improvement scheme, with the first step to challenge the prior agent and second step to generalise against the entire opponent pool.

2.3 Football Games and AI

Football environments are valuable for AI research, as they blend several challenges together; control, strategy, cooperation, and competition. Aside from GFootball, several popular simulators have been proposed. *rSoccer* [36] and *JiDi Olympics Football* [21] are simple environments – players are represented as rigid bodies with a limited action space, either moving or pushing the ball. In contrast, GFootball provides a rich action space, adding mechanics such as slide-tackling and sprinting. The *RoboCup Soccer Simulator* [26] and *DeepMind MuJoCo Multi-Agent Soccer Environment* [31, 32]

environments emphasis low-level robotic control, requiring manipulation of the joints of a player. Meanwhile, GFootball abstracts this away, allowing agents to focus on the challenge of developing high-level behaviors and strategies. A competition in the GFootball environment was hosted on Kaggle [14] in 2020 that attracted over 1,000 teams. This required building an agent to control a single 'in-focus' player at any one time, while teammates were controlled by an in-built rules-based AI. The champion of the competition, named WeKick [69], utilized imitation learning and distributed league training. In contrast to this setup, our system tackles the far more challenging task of controlling *all* 10 outfield players on a team simultaneously, in a decentralized fashion. The only prior work directly tackling this objective first collected a demonstration dataset by rolling out WeKick, and then utilized offline RL techniques to train agents. This was named TiKick, [19]. By contrast, our work trains agents via self-play without requiring demonstration data. Through this methodology, TiZero exceeds the performance of all previous systems in GFootball, including the best entry among 1,000 (WeKick), TiKick, and others.

3 PRELIMINARIES

Multi-agent Reinforcement Learning: We formalize the multi-agent reinforcement learning as a decentralized partially observable Markov decision process (Dec-POMDP) [5]. An n-agent Dec-POMDP can be represented as a tuple $(\mathcal{N}, \mathcal{S}, \mathcal{A}, \mathcal{T}, r, O, G, \gamma)$, where $\mathcal{S}$ is the state space, $\mathcal{A}$ is the action space, O is the observation space and $\gamma \in [0, 1)$ is a reward discount factor. $\mathcal{N} \equiv \{1, ..., n\}$ is a set of $n = |\mathcal{N}|$ agents. At time step t, each agent $i \in \mathcal{N}$ takes an action $a^i \in \mathcal{A}$, forming a joint action $\mathbf{a} \in \mathbf{A} \equiv \mathcal{A}^n$. Agents receive an immediate reward $r(s, \mathbf{a})$ after taking action $\mathbf{a}$ in state s. The reward is shared by all agents. $\mathcal{T}(s, \mathbf{a}, s') : \mathcal{S} \times \mathbf{A} \times \mathcal{S} \mapsto [0, 1]$ is the dynamics function denoting the transition probability. In a Dec-POMDP, an agent will receive its partially observable observation $o_t \in O$ according to the observation function $G(s, i) : \mathcal{S} \times \mathcal{A} \to O$. Each agent has a policy $\pi^i(a_t^i | o_{1:t}^i)$ to produce action a_t^i from local

historical observations $o_{1:t}^i$. We use a_t^{-i} as the action of all complementary agents of agent i and use a similar convention for policies π^{-i}. The agents' objective is to learn a joint policy π that maximizes their expected return $\mathbb{E}_{(s_t, \mathbf{a}_t)} \left[\sum_t \gamma^t r(s_t, \mathbf{a}_t) \right]$.

4 METHODOLOGY

This section introduces TiZero in detail. Firstly, we describe the agent's architecture and observation space. We then introduce a new multi-agent learning algorithm and self-play strategy. We further describe several important implementation details found to be helpful. Finally, we introduce the distributed training framework used to scale up our training.

4.1 Agent Design

Observation space. GFootball provides the observations in several formats, such as an RGB image of the game or a rendered mini-map. Our agents learn from only state vectors – by default this is provided as a 115-dimensional vector, containing information such as player and ball coordinates. We follow previous work [19] which showed benefit in extending this vector with more auxiliary features (such as offside flags to mark potential offside teammates and relative positions among agents) to create a 268-dimensional vector. Instead of using this full vector directly as input, we split the agent observation into six parts, including the controlled player information, player ID, ball information, teammate information, opponent information and current match information. The value-network needs to approximate the value function for the whole team, thus we design a global state vector for the value-network with five parts, including ball information, information of ball holder, teammate information, opponent information and current match information. More details about our observation space can be found in the Appendix M.

Network architecture. Six separate MLPs with two (one for the "player ID") fully-connected layers separately encode each part of the observation. These extracted hidden features are then concatenated together and processed by an LSTM layer [18], which provides the agent with memory. All hidden layers have layer normalization and ReLU non-linearities. We use the orthogonal matrix [44] for parameter initialization and the Adam optimizer [25]. To accelerate learning, we mask out any illegal actions by setting their probability of selection to zero. Figure 2 shows the overall policy network architecture. We also construct a similar architecture for the value network, which is trained with Mean Square Error (MSE) loss. Further hyperparameter details and also the value network structure are provided in the Appendix D.

4.2 Multi-agent Algorithm

We now introduce the Joint-ratio Policy Optimization (JRPO), a modified version of MAPPO [64]. MAPPO is a direct extension of PPO [46] to the multi-agent setting, with the value-network $V_{total}(s_t)$ centralized across all agents, learning from the global state. GAE can then be used to compute the global advantage function $A_{total}(s_t, \mathbf{a}_t)$ (abbreviated as $\hat{A}_t$). In MAPPO, this advantage function guides the improvement of each agent's policy independently $\pi_\theta^i(u_t^i | o_{1:t}^i), i \in \mathcal{N}$, where θ are the parameters of the policy

network. Instead of this, we optimize the *joint*-policy using a decentralized factorization:

$$\pi_\theta(\mathbf{a}_t | \mathbf{o}_{1:t}) \approx \prod_{i=1}^n \pi_\theta^i(a_t^i | o_{1:t}^i). \tag{1}$$

This allows us to write the joint-policy objective as:

$$L^{CLIP}(\theta) = \hat{\mathbb{E}}_t \left[\min \left(r_t(\theta) \hat{A}_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon) \hat{A}_t \right) \right], \tag{2}$$

where $r_t(\theta) = \frac{\pi_\theta(\mathbf{a}_t | \mathbf{o}_{1:t})}{\pi_{\theta_{old}}(\mathbf{a}_t | \mathbf{o}_{1:t})} = \prod_{i=1}^n \frac{\pi_\theta^i(a_t^i | o_{1:t}^i)}{\pi_{\theta_{old}}^i(a_t^i | o_{1:t}^i)}$, and $\hat{\mathbb{E}}_t[\cdots]$ is the expectation using empirical samples. And $\text{clip}(\cdots)$ is a clipping function with clipping hyperparameter ϵ. This is in contrast to vanilla MAPPO, where $r_t(\theta)$ is computed on each agent's policy individually, $r_t(\theta, i) = \frac{\pi_\theta^i(a_t^i | o_{1:t}^i)}{\pi_{\theta_{old}}^i(a_t^i | o_{1:t}^i)}$. By using our factorization, all agent policies are optimized jointly. Previous work has identified that this objective enjoys monotonic guarantees similar to PPO [52]. Our later experiments evidence the empirical advantage of this joint-policy objective, compared to vanilla MAPPO. We believe that framing the objective jointly may encourage the agents to achieve better coordination. Additionally, we found it reduces memory usage and improves training speed.

4.3 Curriculum & Self-play Training Strategy

TiZero's training strategy can be divided into two stages. In the first stage ('Curriculum Self-play'), the difficulty of the scenarios the agents are trained in gradually increases, guiding them to learn basic behaviors in an efficient manner. The opponents in this stage are versions of the agent trained in easier scenarios. In the second stage ('Challenge & Generalise Self-play'), the difficulty-level is fixed at maximum. Agents play against a diverse pool of increasingly strong opponents (previous copies of the agent), providing a route towards superhuman performance.

Curriculum Self-play: The rewards provided in the GFootball 11 vs. 11 game mode are very sparse. This causes vanilla MARL algorithms to struggle. To address this issue, we design a curriculum self-play mechanism, in which agents are trained on a sequence of progressively more difficult scenarios, where the opponent is a copy of the agent from the previous difficulty-level scenario. We design ten difficulty levels by configuring the GFootball environment settings. The difficulty level is determined by two aspects; 1) The strength of opponent players, which can be varied from 0 to 1, with values closer to 1 meaning players are quicker and have better stamina. 2) The initial positions of players and the ball. For example, players can be set in positions closer to the opponent's goal and the ball is also set in position closer to the opponent's goal. At the beginning of training, agents are initialized with random weights, and learn on the lowest difficulty scenario (lowest opponent strength and positioned closest to the goal). This allows agents to receive denser rewards that encourage basic shooting and passing behaviors. When agent performance meets some threshold in the current scenario, the difficulty level automatically increases. Finally in the highest difficulty level, agents must compete with the whole opponent team that encourages more advanced tactics and team cooperations. The Appendix B provides further detail about the curriculum design.
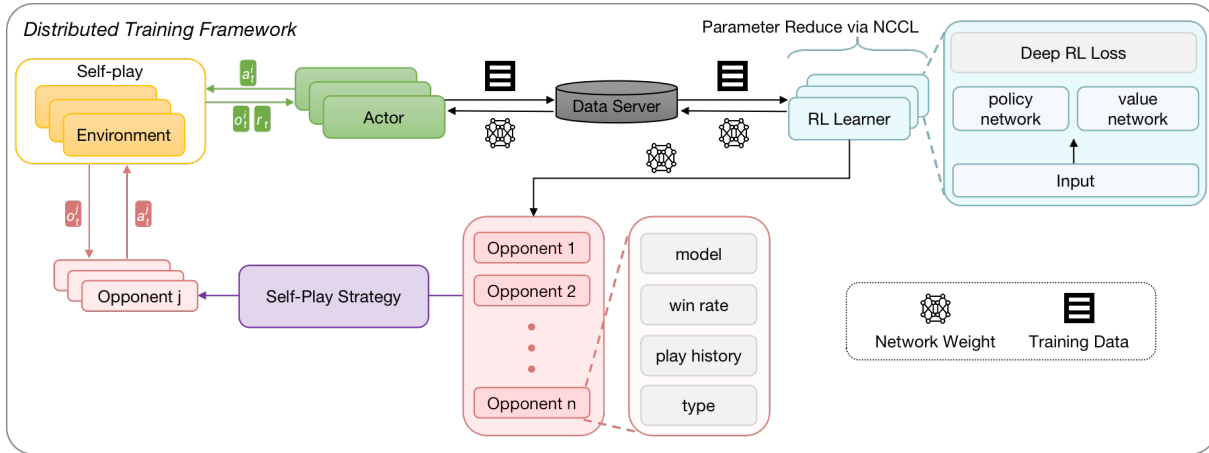
Figure 3: Overview of our distributed training framework. There are two types of modules in the framework; Actors and Learners. Actor roll out models in environments, while storing observations, rewards, actions and LSTM hidden states. The data collected by the Actors is sent to the Learners via a data server. Learners train the value and policy networks, and the gradients are averaged through NCCL reduction. The trained network parameters from the Learners are synchronized to the Actors through the data server. A dynamic opponent pool is maintained during training. A new opponent will be added to the pool when the training model meets certain performance criteria. A configurable self-play strategy module is used to sample opponents for self-play.

Challenge & Generalise Self-play: Through curriculum self-play our agents achieve a basic performance level against a single opponent. To improve performance against a range of opponents, we design an algorithm that produces a monotonically-improving sequence of policies. This consists of two steps. 1) **Challenge Self-play.** Current agents play against the most recently saved agents with probability of 80%, and play against older versions with probability of 20%. The main purpose of this step is ensure the current system can defeat the strongest agents seen so far. 2) **Generalise Self-play.** Current agents play against the whole opponent pool, sampling opponents according to their strength as follows. Denote the opponent pool $\mathcal{M}$. Let $i \notin \mathcal{M}$ be the current training agent, $j \in \mathcal{M}$ all other agents in pool, and $p(i, j)$ be the probability that agent i defeats agent j. We sample model j to play against with probability:

$$p_{\text{sample}}(j) = \frac{f_{\text{hard}}\left(p(i, j)\right)}{\sum_{m \in \mathcal{M}} f_{\text{hard}}\left(p(i, m)\right)}, \quad (3)$$

where $f_{\text{hard}}(x) = (1-x)^2$. This sampling strategy focuses our agents training on opponents it is less likely to win against. Therefore, the training agent will maximize its performance over all existing opponents. Prioritized fictitious self-play addresses the non-transitivity dilemma and improves the robustness of agents [55]. Once agents perform well on this step, they are themselves added to the opponent pool for future versions to train against. The whole self-play algorithm and details of the our self-play strategy can be found in Appendix C & F.

4.4 Implementation Details

This section summarizes several details that were found to be important to achieving good performance.

Recurrent Experience Replay: Allowing agents to learn from earlier observations in an episode helps agents with long-term planning and inferring an opponent's strategy. Hence, we utilize an LSTM [18]. Rather than initializing hidden states with zeros during training on each sequence, we reset to the hidden state that was actually generated by the agent at that timestep – these are stored in the replay buffer during roll outs. When collecting the training data, models are rolled out for sequences of 500 timesteps. During optimization, the LSTM backpropagates through timesteps with length of 25.

Action Masking: Action masking reduces the effective action space by preventing selection of inappropriate actions. In GFootball, we mask out actions that are unavailable or nonsensical during certain situations, for instance slide-tackling is disabled when a teammate holds the ball.

Reward Shaping: The basic rewards provided by GFootball are 1 for scoring a goal, and −1 for conceding a goal. This is sparse and hard to optimize. To improve training efficiency, we design several more dense reward signals. Note all agents on a team receive rewards equally.

- **Holding-Ball reward**: When the ball is controlled by an agents' team, a reward of 0.0001 per timestep is received.
- **Passing-Ball reward**: When agents execute successful pass before a goal, they receive a reward of 0.05. This encourages agents to learn coordinated passing strategies.
- **Grouping penalty**: If agents on the same team gather too closely together, a reward of −0.001 is received. This encourages agents to spread out across the pitch.
- **Out-of-bounds penalty**: A reward of −0.001 is received when an agent is outside of the playing area.

The reward across the two teams is balanced to be zero-sum, which is a basic requirement for self-play to succeed. Hence, if one team receives a positive reward, the other team receives a negative reward of the same magnitude.

Player-ID Conditioning: Instead of training a separate policy network per agent, we use a single shared policy network which receives the player ID as input. Hence, one set of parameters is used by all the agents on a team, while still allowing for decentralized execution. This makes the multi-agent training more sample efficient, since experience from one agent helps improve the network of other agents. It also improves inference speed – only one forward pass of the observation encoder and LSTM is required for the whole team. This produces the embedding that is used by *all* agents after appending their player-ID embedding to the output of the LSTM unit (Figure 2).

Staggered Resetting: To encourage episodes to be initialized in interesting situations, we use the 'staggered resetting' trick when collecting the training data. After the first time resetting the environment, we apply random actions for all players for a variable number of timesteps. This avoids each episode beginning from the same state, which could introduce undesirable correlations in our data.

4.5 Distributed Training Framework

Training a deep neural network at the scale demanded by GFootball requires a large amount of computation. In this work, we built a scalable and loosely-decoupled distributed infrastructure for multi-agent self-play training . There are two types of modules in our framework; **Actors** and **Learners**. These modules are decoupled, enabling researchers to develop their algorithm on a single local machine and then easily launch a large-scale training with **zero-code change**. The main function of the **Actor** is rolling out models in environments (or simulators), while storing observations, rewards, actions and LSTM hidden states. All the data collected by the **Actors** is sent to the **Learners** via a data server. Learners will train the value and policy networks using GPUs and the gradients are averaged through NCCL reduction [39]. The trained network parameters from the **Learners** are synchronized to the **Actors** through the data server. Last but not least, we maintain a dynamic opponent pool and continuously add new opponent to the pool when the training model meets certain performance indexes. Researchers can utilize the stored opponent information (such as winning rates and historical play information) and are flexible to design their own self-play strategies to sample opponents for the self-play. Besides, our model definition and training is done using Pytorch [41]. Figure 3 summarises our distributed training framework.

5 EXPERIMENTS

This section empirically evaluates the performance of TiZero. We first test the full system on our target domain, GFootball. We then explore the generality of several components of the system on environments unrelated to football. Specifically we show that JRPO improves over MAPPO, and that our Challenge & Generalise Self-Play framework produces stronger and more diverse strategies than alternatives.
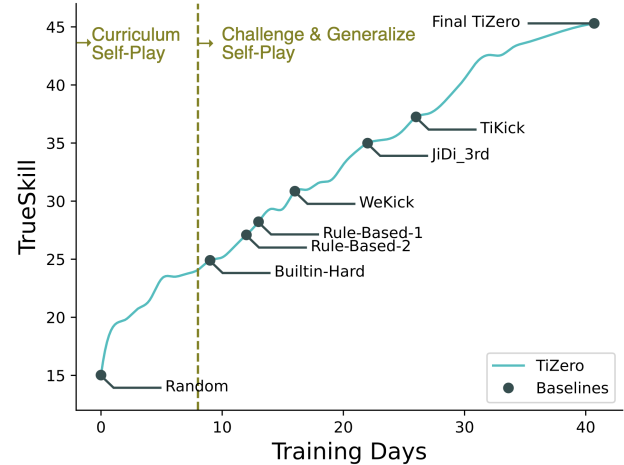

Figure 4: Evolution of TiZero strength over 45 days of training. Dashed green line denotes transition from Curriculum Self-Play to Challenge & Generalize Self-Play. We can observe that the Curriculum Self-play stage allows TiZero to rapidly improve early in training to a level just below the 'Built-in Hard' agent. Through Challenge & Generalize self-play, TiZero continues to gradually improve performance beyond the level of previous systems, eventually plateauing at a rating around 45.

5.1 GFootball Evaluation

Experimental Settings: We train and evaluate TiZero on the full 11 vs. 11 game mode in GFootball. Each match lasts for five minutes, or $3,000$ timesteps (no injury time). The team with highest number of goals wins. Standard football rules are applied by the game, such as offside, penalty kicks and yellow/red cards. TiZero was trained over 45 days on a cluster with 800 CPUs and two NVIDIA A100 GPUs. The batch size for each GPU is set to $2,150,000$, the hidden size of the LSTM layer is 256, and the discount factor γ is 0.999. We used the Adam optimizer with learning rate of 0.0001. Further hyperparameters can be found in the Appendix H.

We compare TiZero to several strong baselines:

- **WeKick** [69]: An RL-based agent that placed first from over 1,000 entries in the 2020 Kaggle Football Competition [14] (see Section 2).
- **TiKick** [19]: The current strongest agent for the GFootball full game, controls all players in the game (see Section 2).
- **JiDi_3rd**: An agent initialized with a pre-trained model and improved via self-play. This was the second runner-up of the 2022 JiDi Football Competition[2].
- **Built-in Hard**: Agent provided by the GFootball environment, which directly controls the underlying game engine.
- **Rule-based**: Two strong rules-based agents from the Kaggle Football Competition, which were hand designed. We use Rule-Based-1 [56] and Rule-Based-2 [1].

We use the TrueSkill rating system [17] to evaluate all systems. Ratings are computed by running a large amount of matches over

---
[2]http://www.jidiai.cn/compete_detail?compete=16

Table 2: Comparison of TiZero with baseline systems on GFootball. Higher is better except for all metrics except 'Draw Rate' and 'Lose Rate'. As well as a better win rate and goal difference, TiZero agents demonstrate a higher level of teamwork, passing the ball more and creating more assists.

| Metric	TiZero (Ours)	TiKick	WeKick	JiDi_3rd	Built-in Hard	Rule-Based-1	Rule-Based-2
Assist	**1.30**(1.02)	0.61(0.79)	0.20(0.47)	0.35(0.62)	0.20(0.55)	0.28(0.59)	0.22(0.53)
Pass	**19.2**(3.44)	6.99(2.71)	5.33(2.44)	3.96(2.33)	11.5(4.63)	7.28(2.77)	7.50(3.12)
Pass Rate	**0.73**(0.07)	0.65(0.17)	0.53(0.18)	0.44(0.19)	0.66(0.12)	0.64(0.17)	0.63(0.19)
Goal	**3.42**(1.69)	1.79(1.41)	0.88(0.88)	1.43(1.34)	0.52(0.91)	0.73(0.69)	0.64(0.82)
Goal Difference	**2.27**(1.93)	0.71(2.08)	-0.47(1.68)	-0.02(2.14)	-1.06(1.93)	-0.60(1.03)	-0.71(1.45)
Draw Rate (%)	**8.50**	22.2	29.0	23.2	24.8	28.7	27.8
Lose Rate (%)	**6.50**	23.5	44.2	33.8	59.6	48.2	49.5
Win Rate (%)	**85.0**	54.3	26.8	43.0	15.6	23.1	22.7
TrueSkill	**45.2**	37.2	30.9	35.0	24.9	28.2	27.1
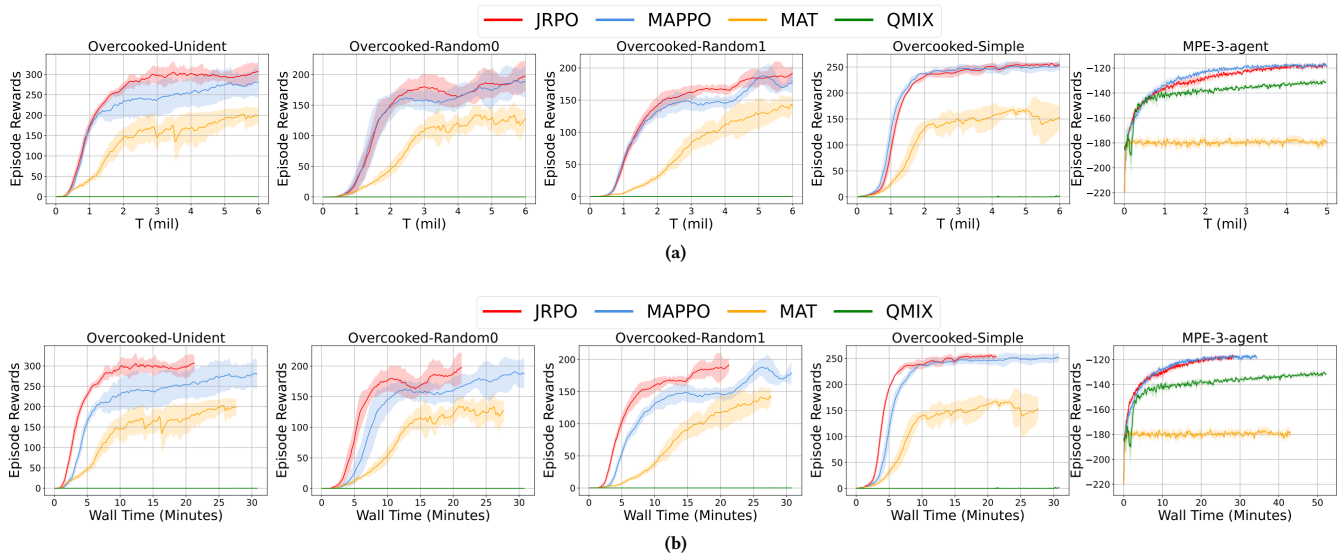
Figure 5: Comparison of multi-agent algorithms. (a) Training curves over environment steps. JRPO's performance is equal or better than MAPPO, MAT and QMIX across all tasks. (b) Training curves over wall-clock time. JRPO trains faster, leading to improvements in terms of wall-clock training time.

a fixed pool of all systems – this pool comprises all the models checkpointed during TiZero's training process, as well as baseline systems. Figure 4 shows the evolution of the TrueSkill rating over 45 days of training. We also mark the TrueSkill rating of the baseline systems, all of which TiZero exceeds by a wide margin. One can observe that the curriculum self-play stage allows TiZero to rapidly improve early in training to a level just below the 'Built-in Hard' agent. Through challenge & generalize self-play, TiZero continues to gradually improve performance beyond the level of previous systems, eventually plateauing at a rating around 45. Table 2 presents TrueSkill ratings and win rates for all systems. To verify our system independently, we also submitted our best TiZero system to a public evaluation platform[1], which maintains a public leaderboard of GFootball systems. At present, TiZero ranks first with a score of 9.7 and win rate of 95.8%.

We conducted an analysis to understand how cooperative TiZero's decentralized agents are. It's possible that superior performance could be achieved through expert control of a single player, rather than through coordinated teamwork leveraging strategies such as 'tiki-taka', crosses and long balls. Table 2 reports statistics that reveal TiZero's agents indeed leverage cooperation more than other systems. TiZero has a higher number of passes or long balls that directly lead to scoring ('Assists'), more passes between teammates ('Pass'), and a higher chance of passes being successfully received by teammates ('Pass Rate'). The Appendix A provides detailed visualizations of TiZero's coordination behaviors.

5.2 Policy Optimization Ablation

This section compares the joint-policy version of PPO (JRPO) we introduced in Section 4.2, with established state-of-the-art MARL algorithms MAPPO [64], MAT [61] and QMIX [43]. We test across the environments Overcooked [59] and Multi-agent Particle-Environment (MPE) [35]. Overcooked is a grid-world game in which agents cooperate to complete a series of tasks, such as finding vegetables, making soup and delivering food. MPE is a continuous 2D world with multiple movable particles. More information about these environments can be found in Appendix G. To be comparable, all three MARL algorithms use the same desktop machine, neural network architecture and hyperparameters. Each method is run over 6 random seeds. More details about the setup of each environment can found in the Appendix I.

Table 3: GPU memory consumption of each method. Lower is better. Results show that JRPO consumes less GPU memory than MAPPO, especially when there are more agents (such as MPE with 20 agents and GFootball with 10 agents).

Task	JRPO	MAPPO
MPE-3-agent	**1.21**GB	1.24GB
MPE-20-agent	**1.63**GB	2.14GB
Overcooked	**5.27**GB	6.65GB
GFootball	**121**GB	196GB

Figure 5 shows the training curves of each methods w.r.t. environment steps and wall-clock time. JRPO's performance is equal or better than MAPPO, MAT and QMIX across all tasks. JRPO also achieves better results with faster wall-clock training time. Table 3 shows the GPU memory consumption of JRPO and MAPPO. Results show that JRPO consumes less GPU memory than MAPPO, especially when there are more agents (such as MPE with 20 agents and GFootball with 10 agents). More experimental results can be found in the Appendix K.
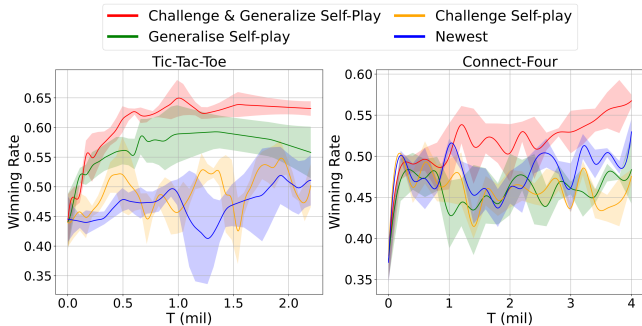
5.3 Self-Play Strategy Ablation

Figure 6: Self-play strategy evaluation. Results show that our method outperforms other baselines in both training efficiency and final performance. It also appears more stable.

This section evaluates our 'Challenge & Generalize Self-Play' strategy on two classical 2-player board games, Tic-Tac-Toe and Connect-Four. In the Tic-Tac-Toe game, two players take turns placing marks on a three-by-three grid. Players win by placing three of their marks in a line. Our agent is a deep neural network with a 27-dimension state vector as input (O's, X's and blanks are one-hot encoded for the 9 cells). Connect-Four is an extended version of Tic-Tac-Toe with a four-by-four board and with placing four marks in a row to win. Our self-play strategy was described in Section 4.3. This ablation omits the curriculum self-play stage since the rewards are non-sparse. Thus, we focus our test on the challenge & generalize self-play stage. We construct several baselines as a comparison: (1) **Challenge Self-Play**: The training agent uses the Challenge Self-Play as described in Section 4.3; (2) **Generalize Self-Play**: The training agent uses the Generalize Self-Play as described in Section 4.3; (3) **Newest**: The training agent only combats with itself. All methods share the same network architecture and learning paradigm. The only difference is how they sample opponents from the pool for self-play.

Table 4: Diversity Index of policies in the opponent pool of each method, measured on Tic-Tac-Toe. Larger values indicate a higher variance of strategies in the opponent pool.

Self-Play Sampling Strategy	Diversity Index
Newest	6.65
Challenge Self-Play	7.19
Generalize Self-Play	7.15
Challenge & Generalize Self-Play	**8.11**

Figure 6 shows the training curves of different self-play strategies over three random seeds. Results show that our method outperforms other baselines in both training efficiency and final performance. For Tic-Tac-Toe, we quantitatively evaluate the Diversity Index[40] of policies in the opponent pool of each method, as shown in Table 4. We see that 'Challenge & Generalize Self-Play' achieves a larger Diversity Index than baselines, which indicates our self-play strategy can produce more diverse policies. Further experimental details and visualizations are given in Appendix J & L.

6 CONCLUSION

This paper presented a distributed multi-agent reinforcement learning system for the complex Google Research Football environment. This environment encapsulates several challenges; multi-agent coordination, sparse rewards, and non-transitivity. Our work is the first to train strong agents for the GFootball 11 vs. 11 game mode from scratch, controlling all 10 outfield players in a decentralized fashion. Achieving this required combining existing techniques, with several innovations – a joint-policy optimization objective, curriculum self-play and a challenge & generalize self-play strategy. Our experiments in GFootball showed that TiZero outperforms previous systems by a wide margin in terms of win rate and goal difference. TiZero also utilizes complex coordination behaviors more often than prior systems. Experiments on other MARL and self-play benchmarks further evidence the effectiveness and generality of our algorithmic innovations.

REFERENCES

[1] Sarvar Anvarov. 2020. Solution ranked 35th in Kaggle Football Competition. https://github.com/Sarvar-Anvarov/Google-Research-Football.

[2] Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, Pablo Sprechmann, Alex Vitvitskyi, Zhaohan Daniel Guo, and Charles Blundell. 2020. Agent57: Outperforming the atari human benchmark. In *International Conference on Machine Learning*. PMLR, 507–517.

[3] David Balduzzi, Marta Garnelo, Yoram Bachrach, Wojciech Czarnecki, Julien Perolat, Max Jaderberg, and Thore Graepel. 2019. Open-ended learning in symmetric zero-sum games. In *International Conference on Machine Learning*. PMLR, 434–443.

[4] Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Chris Hesse, et al. 2019. Dota 2 with large scale deep reinforcement learning. *arXiv preprint arXiv:1912.06680* (2019).

[5] Daniel S Bernstein, Robert Givan, Neil Immerman, and Shlomo Zilberstein. 2002. The complexity of decentralized control of Markov decision processes. *Mathematics of operations research* 27, 4 (2002), 819–840.

[6] Wenze Chen, Shiyu Huang, Yuan Chiang, Ting Chen, and Jun Zhu. 2022. DGPO: Discovering Multiple Strategies with Diversity-Guided Policy Optimization. *arXiv preprint arXiv:2207.05631* (2022).

[7] Karl Cobbe, Chris Hesse, Jacob Hilton, and John Schulman. 2020. Leveraging procedural generation to benchmark reinforcement learning. In *International conference on machine learning*. PMLR, 2048–2056.

[8] Wojciech M Czarnecki, Gauthier Gidel, Brendan Tracey, Karl Tuyls, Shayegan Omidshafiei, David Balduzzi, and Max Jaderberg. 2020. Real world games look like spinning tops. *Advances in Neural Information Processing Systems* 33 (2020), 17443–17454.

[9] Adrien Ecoffet, Joost Huizinga, Joel Lehman, Kenneth O Stanley, and Jeff Clune. 2021. First return, then explore. *Nature* 590, 7847 (2021), 580–586.

[10] Lei Feng, Yuxuan Xie, Bing Liu, and Shuyan Wang. 2022. Multi-Level Credit Assignment for Cooperative Multi-Agent Reinforcement Learning. *Applied Sciences* 12, 14 (2022), 6938.

[11] Jakob Foerster, Gregory Farquhar, Triantafyllos Afouras, Nantas Nardelli, and Shimon Whiteson. 2018. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[12] Wei Fu, Chao Yu, Zelai Xu, Jiaqi Yang, and Yi Wu. 2022. Revisiting some common practices in cooperative multi-agent reinforcement learning. *arXiv preprint arXiv:2206.07505* (2022).

[13] Yiming Gao, Bei Shi, Xueying Du, Liang Wang, Guangwei Chen, Zhenjie Lian, Fuhao Qiu, Guoan Han, Weixuan Wang, Deheng Ye, et al. 2021. Learning Diverse Policies in MOBA Games via Macro-Goals. *Advances in Neural Information Processing Systems* 34 (2021), 16171–16182.

[14] Google. 2020. Google Research Football Competition 2020. https://www.kaggle.com/c/google-football.

[15] Yang Guan, Minghuan Liu, Weijun Hong, Weinan Zhang, Fei Fang, Guangjun Zeng, and Yue Lin. 2022. PerfectDou: Dominating DouDizhu with Perfect Information Distillation. *arXiv preprint arXiv:2203.16406* (2022).

[16] Johannes Heinrich, Marc Lanctot, and David Silver. 2015. Fictitious self-play in extensive-form games. In *International conference on machine learning*. PMLR, 805–813.

[17] Ralf Herbrich, Tom Minka, and Thore Graepel. 2006. Trueskill™: A Bayesian skill rating system. In *Proceedings of the 19th international conference on neural information processing systems*. 569–576.

[18] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation* 9, 8 (1997), 1735–1780.

[19] Shiyu Huang, Wenze Chen, Longfei Zhang, Ziyang Li, Fengming Zhu, Deheng Ye, Ting Chen, and Jun Zhu. 2021. TiKick: Towards Playing Multi-agent Football Full Games from Single-agent Demonstrations. *arXiv preprint arXiv:2110.04507* (2021).

[20] Shiyu Huang, Hang Su, Jun Zhu, and Ting Chen. 2019. Combo-Action: Training Agent For FPS Game with Auxiliary Tasks. AAAI.

[21] JiDi. 2022. JiDi Olympics Football. https://github.com/jidiai/ai_lib/blob/master/env/olympics_football.py.

[22] Steven Kapturowski, Víctor Campos, Ray Jiang, Nemanja Rakićević, Hado van Hasselt, Charles Blundell, and Adrià Puigdomènech Badia. 2022. Human-level Atari 200x faster. *arXiv preprint arXiv:2209.07550* (2022).

[23] Steven Kapturowski, Georg Ostrovski, John Quan, Remi Munos, and Will Dabney. 2018. Recurrent experience replay in distributed reinforcement learning. In *International conference on learning representations*.

[24] Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. 2016. Vizdoom: A doom-based ai research platform for visual reinforcement learning. In *2016 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 1–8.

[25] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[26] Hiroaki Kitano, Minoru Asada, Yasuo Kuniyoshi, Itsuki Noda, and Eiichi Osawa. 1997. Robocup: The robot world cup initiative. In *Proceedings of the first international conference on Autonomous agents*. 340–347.

[27] Vijay Konda and John Tsitsiklis. 1999. Actor-critic algorithms. *Advances in neural information processing systems* 12 (1999).

[28] Karol Kurach, Anton Raichuk, Piotr Stanczyk, Michal Zajac, Olivier Bachem, Lasse Espeholt, Carlos Riquelme, Damien Vincent, Marcin Michalski, Olivier Bousquet, et al. 2019. Google research football: A novel reinforcement learning environment. *arXiv preprint arXiv:1907.11180* (2019).

[29] Marc Lanctot, Vinicius Zambaldi, Audrunas Gruslys, Angeliki Lazaridou, Karl Tuyls, Julien Pérolat, David Silver, and Thore Graepel. 2017. A unified game-theoretic approach to multiagent reinforcement learning. *Advances in neural information processing systems* 30 (2017).

[30] Chenghao Li, Chengjie Wu, Tonghan Wang, Jun Yang, Qianchuan Zhao, and Chongjie Zhang. 2021. Celebrating Diversity in Shared Multi-Agent Reinforcement Learning. *arXiv preprint arXiv:2106.02195* (2021).

[31] Siqi Liu, Guy Lever, Josh Merel, Saran Tunyasuvunakool, Nicolas Heess, and Thore Graepel. 2019. Emergent coordination through competition. *arXiv preprint arXiv:1902.07151* (2019).

[32] Siqi Liu, Guy Lever, Zhe Wang, Josh Merel, SM Eslami, Daniel Hennes, Wojciech M Czarnecki, Yuval Tassa, Shayegan Omidshafiei, Abbas Abdolmaleki, et al. 2021. From Motor Control to Team Play in Simulated Humanoid Football. *arXiv preprint arXiv:2105.12196* (2021).

[33] Siqi Liu, Luke Marris, Daniel Hennes, Josh Merel, Nicolas Heess, and Thore Graepel. 2022. NeuPL: Neural Population Learning. *arXiv preprint arXiv:2202.07415* (2022).

[34] Xiangyu Liu, Hangtian Jia, Ying Wen, Yaodong Yang, Yujing Hu, Yingfeng Chen, Changjie Fan, and Zhipeng Hu. 2021. Unifying behavioral and response diversity for open-ended learning in zero-sum games. *arXiv preprint arXiv:2106.04958* (2021).

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

[36] Felipe B. Martins, Mateus G. Machado, Hansenclever F. Bassani, Pedro H. M. Braga, and Edna S. Barros. 2021. rSoccer: A Framework for Studying Reinforcement Learning in Small and Very Small Size Robot Soccer. arXiv:2106.12895 [cs.LG]

[37] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. 2015. Human-level control through deep reinforcement learning. *Nature* 518, 7540 (2015), 529.

[38] Andrew Y Ng, Daishi Harada, and Stuart Russell. 1999. Policy invariance under reward transformations: Theory and application to reward shaping. In *Icml*, Vol. 99. 278–287.

[39] NVIDIA. 2016. NCCL. https://github.com/NVIDIA/nccl.

[40] Jack Parker-Holder, Aldo Pacchiano, Krzysztof M Choromanski, and Stephen J Roberts. 2020. Effective diversity in population based reinforcement learning. *Advances in Neural Information Processing Systems* 33 (2020), 18050–18062.

[41] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems* 32 (2019), 8026–8037.

[42] Tim Pearce and Jun Zhu. 2022. Counter-Strike Deathmatch with Large-Scale Behavioural Cloning. *2022 IEEE Conference on Games (CoG)* (2022), 104–111. https://doi.org/10.1109/CoG51982.2022.9893617

[43] Tabish Rashid, Mikayel Samvelyan, Christian Schroeder, Gregory Farquhar, Jakob Foerster, and Shimon Whiteson. 2018. Qmix: Monotonic value function factorisation for deep multi-agent reinforcement learning. In *International Conference on Machine Learning*. PMLR, 4295–4304.

[44] Andrew M Saxe, James L McClelland, and Surya Ganguli. 2013. Exact solutions to the nonlinear dynamics of learning in deep linear neural networks. *arXiv preprint arXiv:1312.6120* (2013).

[45] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. 2015. High-dimensional continuous control using generalized advantage estimation. *arXiv preprint arXiv:1506.02438* (2015).

[46] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347* (2017).

[47] Ruggiero Seccia, Francesco Foglino, Matteo Leonetti, and Simone Sagratella. 2022. A novel optimization perspective to the problem of designing sequences of tasks in a reinforcement learning framework. *Optimization and Engineering* (2022), 1–16.

[48] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of Go with deep neural networks and tree search. *Nature* 529, 7587 (2016), 484–489.

[49] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel,

et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815* (2017).

[50] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2018. A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play. *Science* 362, 6419 (2018), 1140–1144.

[51] Kyunghwan Son, Daewoo Kim, Wan Ju Kang, David Earl Hostallero, and Yung Yi. 2019. Qtran: Learning to factorize with transformation for cooperative multi-agent reinforcement learning. In *International Conference on Machine Learning*. PMLR, 5887–5896.

[52] Mingfei Sun, Sam Devlin, Jacob Beck, Katja Hofmann, and Shimon Whiteson. 2022. Monotonic Improvement Guarantees under Non-stationarity for Decentralized PPO. *arXiv preprint arXiv:2202.00082* (2022).

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

[54] Adrien Ali Taiga, William Fedus, Marlos C Machado, Aaron Courville, and Marc G Bellemare. 2019. On bonus based exploration methods in the arcade learning environment. In *International Conference on Learning Representations*.

[55] Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, Michaël Mathieu, Andrew Dudzik, Junyoung Chung, David H Choi, Richard Powell, Timo Ewalds, Petko Georgiev, et al. 2019. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* 575, 7782 (2019), 350–354.

[56] w9PcJLyb. 2020. Solution ranked 15th in Kaggle Football Competition. https://github.com/w9PcJLyb/GFootball.

[57] Jianhao Wang, Zhizhou Ren, Terry Liu, Yang Yu, and Chongjie Zhang. 2020. Qplex: Duplex dueling multi-agent q-learning. *arXiv preprint arXiv:2008.01062* (2020).

[58] Li Wang, Yupeng Zhang, Yujing Hu, Weixun Wang, Chongjie Zhang, Yang Gao, Jianye Hao, Tangjie Lv, and Changjie Fan. 2022. Individual Reward Assisted Multi-Agent Reinforcement Learning. In *International Conference on Machine Learning*. PMLR, 23417–23432.

[59] Rose E Wang, Sarah A Wu, James A Evans, Joshua B Tenenbaum, David C Parkes, and Max Kleiman-Weiner. 2020. Too many cooks: Coordinating multi-agent collaboration through inverse planning. (2020).

[60] Tonghan Wang, Tarun Gupta, Anuj Mahajan, Bei Peng, Shimon Whiteson, and Chongjie Zhang. 2020. RODE: Learning Roles to Decompose Multi-Agent Tasks. *arXiv preprint arXiv:2010.01523* (2020).

[61] Muning Wen, Jakub Grudzien Kuba, Runji Lin, Weinan Zhang, Ying Wen, Jun Wang, and Yaodong Yang. 2022. Multi-Agent Reinforcement Learning is a Sequence Modeling Problem. *arXiv preprint arXiv:2205.14953* (2022).

[62] Yuxin Wu and Yuandong Tian. 2016. Training agent for first-person shooter game with actor-critic curriculum learning. (2016).

[63] Deheng Ye, Guibin Chen, Wen Zhang, Sheng Chen, Bo Yuan, Bo Liu, Jia Chen, Zhao Liu, Fuhao Qiu, Hongsheng Yu, et al. 2020. Towards playing full moba games with deep reinforcement learning. *Advances in Neural Information Processing Systems* 33 (2020), 621–632.

[64] Chao Yu, Akash Velu, Eugene Vinitsky, Yu Wang, Alexandre Bayen, and Yi Wu. 2021. The Surprising Effectiveness of MAPPO in Cooperative, Multi-Agent Games. *arXiv preprint arXiv:2103.01955* (2021).

[65] Daochen Zha, Jingru Xie, Wenye Ma, Sheng Zhang, Xiangru Lian, Xia Hu, and Ji Liu. 2021. Douzero: Mastering doudizhu with self-play deep reinforcement learning. In *International Conference on Machine Learning*. PMLR, 12333–12344.

[66] Tianjun Zhang, Huazhe Xu, Xiaolong Wang, Yi Wu, Kurt Keutzer, Joseph E Gonzalez, and Yuandong Tian. 2020. BeBold: Exploration Beyond the Boundary of Explored Regions. *arXiv preprint arXiv:2012.08621* (2020).

[67] Nianzhao Zheng, Jialong Li, Zhenyu Mao, and Kenji Tei. 2022. From Local to Global: A Curriculum Learning Approach for Reinforcement Learning-based Traffic Signal Control. In *2022 IEEE 2nd International Conference on Software Engineering and Artificial Intelligence (SEAI)*. IEEE, 253–258.

[68] Meng Zhou, Ziyu Liu, Pengwei Sui, Yixuan Li, and Yuk Ying Chung. 2020. Learning implicit credit assignment for cooperative multi-agent reinforcement learning. *Advances in Neural Information Processing Systems* 33 (2020), 11853–11864.

[69] Fengming Zhu Ziyang Li, Kaiwen Zhu. 2020. WeKick. https://www.kaggle.com/c/google-football/discussion/202232.