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

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.

<table>
<thead>
<tr>
<th>Citation</th>
<th>Game</th>
<th>Competitive?</th>
<th>Agent number</th>
<th>Stochastic?</th>
<th>Sparse reward?</th>
<th>Game length magnitude</th>
</tr>
</thead>
<tbody>
<tr>
<td>[22, 37]</td>
<td>Atari Games</td>
<td>✗</td>
<td>1</td>
<td>✗</td>
<td>Sometimes</td>
<td>Typically 10^2</td>
</tr>
<tr>
<td>[48]</td>
<td>Go</td>
<td>✓</td>
<td>1</td>
<td>✗</td>
<td>✓</td>
<td>10^2</td>
</tr>
<tr>
<td>[7]</td>
<td>Procgen</td>
<td>✗</td>
<td>1</td>
<td>✓</td>
<td>✗</td>
<td>10^2</td>
</tr>
<tr>
<td>[13, 63]</td>
<td>Honor of Kings</td>
<td>✓</td>
<td>5</td>
<td>✗</td>
<td>✓</td>
<td>10^3</td>
</tr>
<tr>
<td>[12, 30, 58, 61]</td>
<td>GFootball Academy</td>
<td>✗</td>
<td>&lt; 10</td>
<td>✓</td>
<td>✓</td>
<td>10^2</td>
</tr>
<tr>
<td>Our work</td>
<td>GFootball 11 vs. 11</td>
<td>✓</td>
<td>10</td>
<td>✓</td>
<td>✓</td>
<td>10^3</td>
</tr>
</tbody>
</table>

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 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 ⇒ 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 first1. 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 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 an 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 JDiAI Competition Platform: http://www.jidiai.cn/ranking_list?tab=34. The evaluation result was collected on October 28th, 2022.
AAMAS 2023, May 29–June 2, 2023, London, United Kingdom

2.2 Competitive Self-play
Self-play has emerged as a powerful technique to obtain super-human 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 emphasise 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. \(\mathcal{N} \equiv \{1, \ldots, n\}\) 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 \(a \in \mathcal{A} \equiv \mathcal{A}^n\). Agents receive an immediate reward \(r(s, a)\) after taking action \(a\) in state \(s\). The reward is shared by all agents. \(\mathcal{T}(s, a, s') : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \mapsto [0, 1]\) is the dynamics function denoting the transition probability. In a Dec-POMDP, an agent will receive its partially observable observation \(o_i \in O\) according to the observation function \(G(s, i) : \mathcal{S} \times \mathcal{A} \rightarrow O\). Each agent has a policy \(\pi^i(a'_i|o^i_{1:t})\) to produce action \(a'_i\) from local

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

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 B.

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

$$π_0(a_t'|o_{1:t}) \approx \prod_{i=1}^n π_0^i(a_t'|o_{1:t}^i).$$

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

$$L^{CLIP}(θ) = \mathbb{E}_{τ} \left[ \min \left( r_t(θ)A_τ, \text{clip}(r_t(θ), 1 - \epsilon, 1 + \epsilon)A_τ \right) \right],$$

where $r_t(θ) = \frac{π_0^i(a_t'|o_{1:t}^i)}{π_0^i(a_t'|o_{1:t}^i)} = \prod_{t=1}^n \frac{π_0^i(a_t'|o_{1:t}^i)}{π_{θ_0^i}(a_t'|o_{1:t}^i)},$ and $\mathbb{E}_{τ}[\cdot]$ is the expectation using empirical samples. And clip($\cdot$) is a clipping function with clipping hyperparameter $\epsilon$. This is in contrast to vanilla MAPPO, where $r_t(θ)$ is computed on each agent’s policy individually; $r_t(θ, i) = \frac{π_0^i(a_t'|o_{1:t}^i)}{π_{θ_0^i}(a_t'|o_{1:t}^i)}$. 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 details 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 to 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 $M$. Let $i \in M$ be the current training agent, $j \in M$ 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:

$$p_{\text{sample}}(j) = \frac{f_{\text{hard}}(p(i, j))}{\sum_{m \in M} f_{\text{hard}}(p(i, m))},$$  \hspace{1cm} (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.](http://www.jidiai.cn/compete_detail?compete=16)

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 \(\gamma\) 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.

<table>
<thead>
<tr>
<th>Metric</th>
<th>TiZero (Ours)</th>
<th>TiKick</th>
<th>WeKick</th>
<th>JiDi_3rd</th>
<th>Built-in Hard</th>
<th>Rule-Based-1</th>
<th>Rule-Based-2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Assist</td>
<td>1.30(1.02)</td>
<td>0.61(0.79)</td>
<td>0.20(0.47)</td>
<td>0.35(0.62)</td>
<td>0.20(0.55)</td>
<td>0.28(0.59)</td>
<td>0.22(0.53)</td>
</tr>
<tr>
<td>Pass</td>
<td>19.2(3.44)</td>
<td>6.99(2.71)</td>
<td>5.33(2.44)</td>
<td>3.96(2.33)</td>
<td>11.5(4.63)</td>
<td>7.28(2.77)</td>
<td>7.50(3.12)</td>
</tr>
<tr>
<td>Pass Rate</td>
<td>0.73(0.07)</td>
<td>0.65(0.17)</td>
<td>0.53(0.18)</td>
<td>0.44(0.19)</td>
<td>0.66(0.12)</td>
<td>0.64(0.17)</td>
<td>0.63(0.19)</td>
</tr>
<tr>
<td>Goal</td>
<td>3.42(1.69)</td>
<td>1.79(1.41)</td>
<td>0.88(0.88)</td>
<td>1.43(1.34)</td>
<td>0.52(0.91)</td>
<td>0.73(0.69)</td>
<td>0.64(0.82)</td>
</tr>
<tr>
<td>Goal Difference</td>
<td>2.27(1.93)</td>
<td>0.71(2.08)</td>
<td>-0.47(1.68)</td>
<td>-0.02(2.14)</td>
<td>-1.06(1.93)</td>
<td>-0.60(1.03)</td>
<td>-0.71(1.45)</td>
</tr>
<tr>
<td>Draw Rate (%)</td>
<td>8.50</td>
<td>22.2</td>
<td>29.0</td>
<td>23.2</td>
<td>24.8</td>
<td>28.7</td>
<td>27.8</td>
</tr>
<tr>
<td>Lose Rate (%)</td>
<td>6.50</td>
<td>23.5</td>
<td>44.2</td>
<td>33.8</td>
<td>59.6</td>
<td>48.2</td>
<td>49.5</td>
</tr>
<tr>
<td>Win Rate (%)</td>
<td>85.0</td>
<td>54.3</td>
<td>26.8</td>
<td>43.0</td>
<td>15.6</td>
<td>23.1</td>
<td>22.7</td>
</tr>
<tr>
<td>TrueSkill</td>
<td>45.2</td>
<td>37.2</td>
<td>30.9</td>
<td>35.0</td>
<td>24.9</td>
<td>28.2</td>
<td>27.1</td>
</tr>
</tbody>
</table>

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 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.

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 environment steps and wall-clock time. JRPO’s performance is equal or better than MAPPO, especially when there are more agents (such as MPE with 20 agents and GFootball with 10 agents).

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).

<table>
<thead>
<tr>
<th>Task</th>
<th>JRPO</th>
<th>MAPPO</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPE-3-agent</td>
<td>1.21GB</td>
<td>1.24GB</td>
</tr>
<tr>
<td>MPE-20-agent</td>
<td>1.63GB</td>
<td>2.14GB</td>
</tr>
<tr>
<td>Overcooked</td>
<td>5.27GB</td>
<td>6.65GB</td>
</tr>
<tr>
<td>GFootball</td>
<td>121GB</td>
<td>196GB</td>
</tr>
</tbody>
</table>

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

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.

<table>
<thead>
<tr>
<th>Self-Play Sampling Strategy</th>
<th>Diversity Index</th>
</tr>
</thead>
<tbody>
<tr>
<td>Newest</td>
<td>6.65</td>
</tr>
<tr>
<td>Challenge Self-Play</td>
<td>7.19</td>
</tr>
<tr>
<td>Generalize Self-Play</td>
<td>7.15</td>
</tr>
<tr>
<td>Challenge &amp; Generalize Self-Play</td>
<td>8.11</td>
</tr>
</tbody>
</table>

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.
AAMAS 2023, May 29–June 2, 2023, London, United Kingdom

Session 1A: Multiagent Reinforcement Learning I

REFERENCES


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


