Project 3 - CS7642: Reinforcement Learning and Decision Making

Syed Muhammad Husainie
shusainie3@gatech.edu
GitHub Hash: 64a5009
Georgia Institute of Technology
OMSCS – CS 7643: Deep Learning

Abstract

In this project, the Multi-Agent Proximal Policy Optimization (MAPPO) algorithm is implemented to train agents in the Overcooked environment. The objective is to develop a policy that enables agents to cooperatively deliver at least 7 soups successfully across three distinct kitchen layouts. MAPPO leverages a centralized critic and decentralized actor networks, allowing agents to coordinate effectively while exploring the environment. The agents are trained in a simulated version of the Overcooked-AI environment, which includes a predefined set of rules and structured reward signals to guide learning.

1 Overcooked Environment

1.1 State Space

Overcooked is a fully observable Markov Decision Process (MDP) where both agents receive identical and complete observations. Each agent observes a 96-dimensional vector comprising of:

- **Player-features (46 dims)**: This includes orientation, held object at a timestep, distance to nearby items (e.g., onions, dishes, pots), pot states, and wall adjacency.
- **Other player features (46 dims)**: This follows the same state space as the original play, but centered on the teammate.
- **Relative position (2 dims)**: $(\Delta x, \Delta y)$ to the other agent.
- **Absolute position (2 dims)**: Agent's current location on the grid.

1.2 Action Space

The action space for this environment is discrete with six possible actions: up, down, left, right, stay, and "interact" which records actions such as picking up an onion, pot, dish etc. Each layout is unique and has an unlimited supply of items.

1.3 Goal

The objective of this project is to train a multi-agent reinforcement learning (MARL) policies to achieve efficient collaboration in the Overcooked-AI environment, focusing on delivering more than seven soups on average after a 100 episode evaluation period. The key goal is to enable the agents to learn coordinated behaviors such as task allocation, collision avoidance, amount of onions dropped in pot etc., which are crucial for training an effective policy.

2 Algorithm Choice

Algorithm Selection: In this project, Multi-Agent Proximal Policy Optimization (MAPPO) was implemented due to its proven effectiveness in multi-agent reinforcement learning tasks (11; 7).

MAPPO extends the PPO algorithm (9) to multi-agent environments by enabling centralized training with decentralized execution. This approach provides a balance between sample efficiency and training stability, which is essential for environments requiring tight coordination, such as Overcooked.

2.1 Algorithm Implementation

This study implemented a Multi-Agent Proximal Policy Optimization (MAPPO) algorithm, training with a centralized critic using global state information for better value estimates, and evaluating via decentralized execution. This approach suits partially observable cooperative environments by allowing actors to learn from local observations and act independently, with the mean reward used as the return target.

Actor and Critic Networks: In this MAPPO implementation, the actor network learns each agent's policy by mapping local observations to action probabilities through a feedforward neural network with four ReLU-activated hidden layers. The output logits form a Categorical distribution suitable for discrete actions. During training, actions are sampled stochastically (deterministic=False) to encourage exploration, and the log-probabilities of chosen actions are used in PPO's clipped surrogate loss for policy optimization.

The critic network is centralized and receives the joint observations from all agents, providing a full view of the state. This improves value estimation in cooperative environments like Overcooked, where coordination and partial observations limit decentralized critics. Centralized critics better capture inter-agent dependencies and stabilize training, as highlighted by (1). The advantages of centralized value functions in cooperative-competitive tasks have also been demonstrated by (5), outperforming decentralized baselines.

Like the actor, the critic uses a feedforward network with four ReLU layers but takes concatenated joint observations as input and outputs a single scalar value estimating the return. This estimate is used to compute the advantage function, reflecting how well an action performed relative to expectations.

During training, agents act independently based on their policies, while the critic evaluates the joint state to compute advantages. The actor then updates its policy using PPO's clipped objective to maintain stable yet effective learning.

Rollout Buffer: The `RolloutBuffer` class was implemented to store and manage experiences collected during training episodes, essential for the on-policy MAPPO algorithm (9). It stores observations, actions, log-probabilities, rewards, done flags, value estimates, advantages, and returns at each timestep for all agents. To handle multiple agents, the buffer manages data in 3D tensor formats, e.g., `(buffer_size, num_agents, obs_dim)` for observations, with similar shapes for other elements, reshaped as needed during training.

At each environment step, the buffer's `store()` method saves data, converting from NumPy arrays to PyTorch tensors while ensuring consistent shapes. When full or ready for update, `compute_advantages()` calculates agent-specific advantages and average return targets using Generalized Advantage Estimation (GAE) (8), balancing bias and variance via the λ parameter. The method processes data backward, maintaining actor-specific advantages and a centralized critic value for stability.

The buffer also provides a `get_minibatches()` generator to support batching and shuffling during PPO updates, enabling multiple training epochs on sampled subsets. The `reset()` method clears pointers post-update. This modular buffer aligns with PPO and MAPPO training needs for efficient storage, return computation, and sampling (11). However, minibatch sampling was done directly in the `MAPPOAgent` class for simplicity rather than using the buffer's generator.

Agent: The `MAPPOAgent` class implements the Multi-Agent Proximal Policy Optimization (MAPPO) algorithm (11), combining decentralized actors with a centralized critic. The constructor

initializes hyperparameters including learning rates, discount factor (γ), GAE smoothing factor (λ), clipping range, and entropy coefficient, alongside actor and critic networks with their optimizers. To enhance training stability, both clipping range and entropy coefficient decay exponentially over episodes.

The `act()` method selects actions by sampling from the actor's policy distribution (or choosing the most probable action in deterministic mode). The centralized critic estimates the value of the joint observation, reshaped appropriately for input.

The main learning occurs in the `update()` method, which batches and reshapes data from the buffer for joint optimization. It computes standardized advantages using Generalized Advantage Estimation (GAE) (8) and applies the PPO clipped surrogate objective (9) for stable updates. Actor parameters are updated over multiple minibatch epochs with policy loss and entropy regularization, while the critic is updated by minimizing mean squared error between predicted and actual returns averaged across agents.

The class also includes `save()` and `load()` methods for storing and retrieving model parameters. For evaluation and visualization, `get_action_probs()` and `get_value()` compute action probabilities and value estimates, respectively. This design ensures modularity and efficient training for cooperative multi-agent environments like *Overcooked*.

Trainer: The `MAPPOTrainer` class handles the training loop of the MAPPO agent within the multi-agent *Overcooked* environment. It manages rollout collection, reward accumulation, advantage estimation, and policy updates. Various logging structures track progress metrics such as episode rewards, soup deliveries, and object placements.

The main training routine is implemented in the `run()` method, where agents interact with the environment for a set number of rollouts. Each rollout involves environment resets, action selection via `act()`, execution of actions, and storage of transitions in the experience buffer. The method ensures sufficient environment sampling before each update. When enough episodes are collected, the next state's value is estimated to bootstrap advantage computation using Generalized Advantage Estimation (GAE) (8), followed by policy and value updates via PPO's clipped objective (9).

Training progress is periodically reported, showing metrics such as average episodic reward and soups prepared. Additional functions like `evaluate()`, `plot_rolling_average()`, and `plot_placement_in_pot_per_episode()` support performance visualization and debugging, enabling monitoring of both task success (e.g., soup delivery) and intermediate behaviors (e.g., onion pickup), which are essential in cooperative multi-agent learning.

Implementation: The `Implementation` script sets up and runs the complete training pipeline for a Multi-Agent Proximal Policy Optimization (MAPPO) agent within the *Overcooked-AI* environment. The training layout is specified via the `layout` variable, which is passed to the `OvercookedEnv` wrapper for environment interaction.

The training loop starts by obtaining environment details such as the number of agents, observation space dimensions, and action space size. These parameters define the architecture of the policy and value networks within the `MAPPOAgent`. A `RolloutBuffer` is also initialized to store experience tuples (observations, actions, rewards, etc.) throughout episodes.

Training is initiated by calling `trainer.run()`, which manages environment rollouts, reward calculation, advantage estimation, and policy updates. After training, performance is visualized using rolling averages of episodic rewards and soup deliveries. Intermediate behavior metrics such as onion pickups, dish pickups, and pot placements are also plotted.

Finally, the trained model is evaluated over 100 episodes using the `evaluate()` method to assess its cooperative behavior in terms of reward and soup production consistency.

Initial Hyperparameters:

- **Network Architecture:** Actor and centralized critic networks with hidden layers of size (32, 32).
- **Mini-batch Size:** 400 observations with no random sampling.
- **Total Episodes:** 5000 episodes used for training.
- **Actor Learning Rate:** 1e-4.
- **Critic Learning Rate:** 1e-4.
- **Clipping Range (PPO):** No clip decay. Clip range = 0.2.
- **Entropy Coefficient:** No entropy decay. Entropy coefficient = 0.01.
- **Update Epochs:** 10 epochs of gradient descent per policy update.
- **Rollout Buffer Size:** 400 transitions per agent stored before each update.
- **Episodes per Update:** Training updates were triggered after collecting observations from just one episode.
- **Episode Horizon:** 400 timesteps per episode (as per environment constraints).

Initial Reward Shaping: The following rewards were used as a combined tensor for each agent, and fed into the

- `PLACEMENT_IN_POT_REWARD`: +3 reward for successfully placing an ingredient into a pot.
- `DISH_PICKUP_REWARD`: +3 reward for picking up a clean dish.
- `SOUP_PICKUP_REWARD`: +5 reward for picking up a prepared soup.
- **Sparse Reward:** The default Overcooked environment provides +20 reward for each soup successfully delivered.

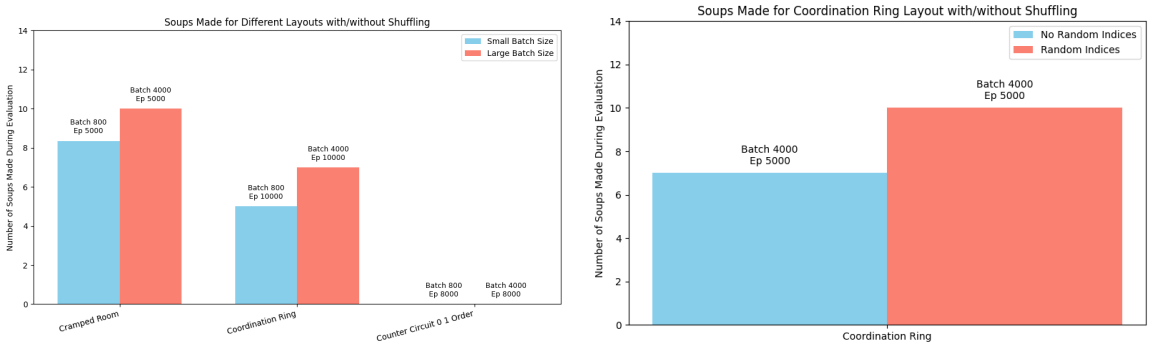
3 Evaluation

- **Collaboration and Coordination:** Overcooked-AI requires strong collaboration, achieved via centralized critics with access to joint observations and actions during training (3). Reward shaping further encourages cooperative tasks such as placing ingredients and delivering soups (6; 2). Mini-batching and joint state representations support synchronization between agents.
- **Handling Challenges:**
  - *Credit Assignment:* Shaped rewards guide agents by attributing individual contributions to team objectives (10), with advantages computed per agent using a centralized critic.
  - *Training Stability:* PPO's clipped objective (9), along with mini-batch updates and rollout buffers, ensures stable training. Additional stability comes from reward normalization, early stopping, deeper networks, and decaying exploration.
  - *Environment Stochasticity:* Randomness in Overcooked is mitigated by training over many episodes and using sufficient sample sizes (4).
- **Ablation Study:** Reward shaping and mini-batching improve convergence and coordination. Variants of buffer sizes, rollout lengths, entropy coefficients, and network depths were tested to evaluate their effect on performance and stability.

3.1 Results Discussion

When the MAPPO implementation was first evaluated on Layout 1, the trainer achieved an average of up to 10 soups after 5000 training episodes, indicating a promising and functional implementation. However, on Layout 2, the same implementation struggled to produce more than 5 soups on average, even after over 10,000 training episodes, and showed signs of overtraining with diminishing rewards. Despite this, the results confirmed that the core implementation was sound and provided a solid foundation for further tuning or the application of more advanced methods.

3.1.1 Comparing Roll-out Buffer Implementations



(a) Performance of small vs large buffer size (no random sampling)

(b) Performance with and without shuffling of mini-batches

Figure 1: Comparison of training settings

The first optimization improved the replay buffer's size and sampling strategy. Initially, updates occurred after each episode using a basic buffer, limiting training data diversity and volume.

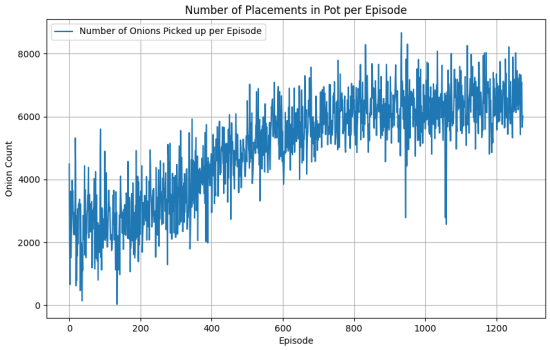
A mini-batch buffer was introduced to accumulate larger experience batches and perform updates with random sampling. This reduced temporal correlations in the data, benefiting on-policy learning.

These changes notably improved evaluation in Layout 2, consistently achieving at least 7 soups. The gains stemmed from:

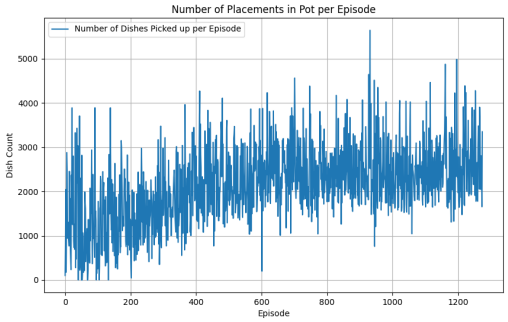
- **Larger buffer size**, providing more diverse training samples.
- **Random sampling**, preventing overfitting to local temporal patterns.

Layout 3 performance remained poor, likely due to its complexity. Nonetheless, adopting a mini-batch buffer with random sampling enhanced training stability and policy effectiveness, enabling further improvements.

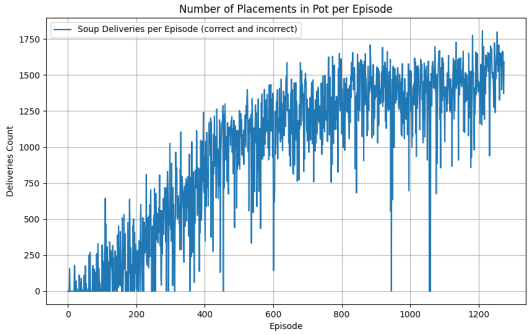
3.1.2 Tuning Rewards



(a) Number of Onions Picked



(b) Number of Dishes Picked Up



(c) Soup Deliveries Count

Figure 2: Agent activity and performance metrics in Layout 1: (a) onions picked, (b) dishes picked up, and (c) soups delivered over training episodes.

Layout 3 posed significant challenges due to its complexity and coordination demands. Initially, a basic shaped reward (+3 for ingredient/dish pickup, +5 for soup pickup, +20 for delivery) failed to guide agents effectively, resulting in zero soups on average. To improve this, a refined reward shaping approach incorporated additional event-based signals such as *useful onion pickup/drop*, *optimal onion potting*, and *soup delivery*, normalized per agent and combined with sparse rewards. This richer feedback enhanced credit assignment and broke temporal correlations. As shown in Table 1, average soups delivered in Layout 3 rose from 0 to around 4–5.

Figure 2 illustrates key training metrics on Layout 1: (a) onions picked, (b) dishes picked, and (c) soups delivered. The upward trends confirm agents progressively learn cooperative behaviors. The high onion pickup rate suggests some inefficiency or repetition, possibly explaining Layout 3's incomplete solution. However, the steady increase in soup deliveries validates the reward shaping and training approach. Improvements in onion and dish pickups indicate agents successfully execute necessary subtasks to optimize performance. These results demonstrate the framework's effectiveness and potential for tuning to harder layouts.

Table 1: Effect of Reward Shaping on Layout 3 Performance

Reward Structure	**Average Soups (Evaluation)**
Initial reward shaping	0
Tuned reward shaping (event-based)	4–5

3.1.3 Tuning Hidden Dimensions

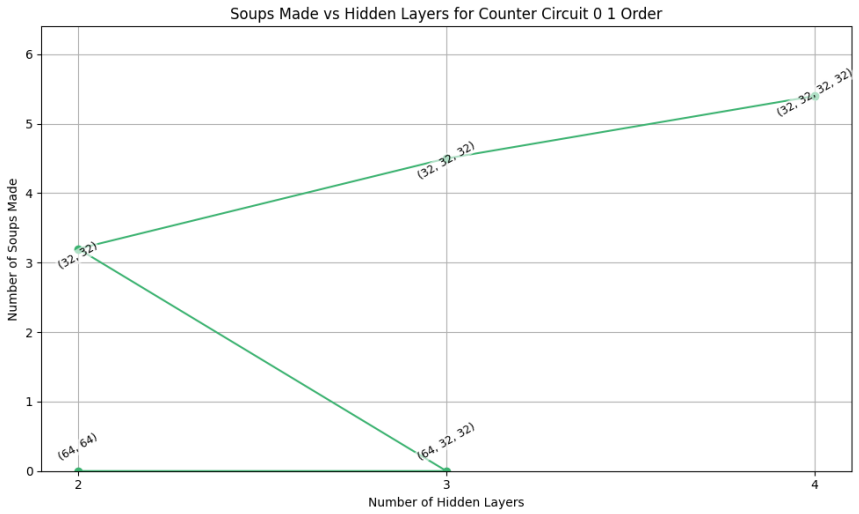


Figure 3: Effect of increasing hidden dimensions and length for actor and critic networks on the average number of soups made during evaluation

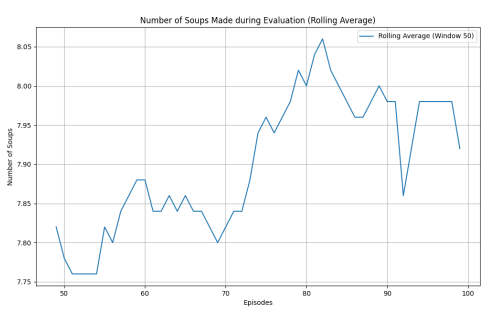
The results illustrated in Figure 3 highlight the impact of varying network architecture on the number of average soups made during evaluation. Specifically, increasing the dimensionality of the hidden layers from $(32, 32)$ to $(64, 64)$ resulted in a noticeable decline in performance. This suggests that a larger model capacity led to overfitting, where the network learned spurious patterns that did not generalize well to new episodes. Consequently, a simpler architecture with moderate dimensionality, such as $(32, 32)$, proved more effective. On the other hand, increasing the depth of the network—by adding more hidden layers while maintaining the dimensionality at 32—had a positive impact. A clear, progressive improvement in the number of soups was observed when increasing from two layers to three and then four layers. This indicates that deeper architectures were able to better capture complex patterns in the environment without the risk of overfitting, as long as the dimensionality remained modest. However, further experiments revealed that increasing the dimensionality of the **first layer** in deeper networks (e.g., using a $(64, 32, 32)$ configuration) negatively affected performance. This again pointed to the issue of overfitting and instability during training. These findings suggest that while deeper networks can improve performance, they should maintain a constrained width to ensure robust generalization.

Other hyperparameters were also tuned, including the exponential decay rate of entropy and the clipping range, as well as the learning rates for both the actor and critic networks. These adjustments had a significant impact on enhancing the algorithm's training performance.

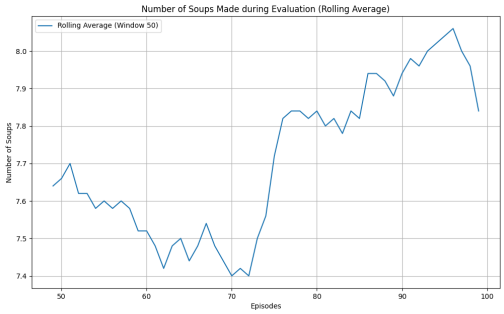
3.2 Evaluating the Three Layouts

Figure 3 illustrates the performance of the trained agents on Layout 1, Layout 2, and Layout 3 respectively, measured by the number of soups successfully delivered during evaluation episodes. Layout 1 demonstrates a strong upward trend in soup deliveries, indicating that the agents effectively learned the coordination required for the relatively simpler layout. Layout 2 shows further improvement, consistently reaching or surpassing the target threshold of 7 soups per episode, which validates the efficacy of the implemented reward shaping and training optimizations.

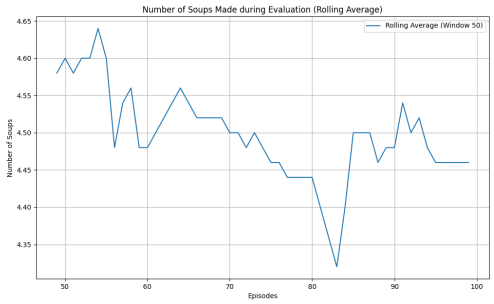
For Layout 3, the evaluation results reveal more modest performance, averaging only 3 to 6 soups per episode. This shortfall suggests that the higher complexity and coordination challenges of Layout 3 remain difficult for the agents to overcome fully. Despite the enhanced reward shaping and training



(a) Layout 1



(b) Layout 2



(c) Layout 3

Figure 4: Evaluation performance across Overcooked layouts. Layouts 1 and 2 meet the target soup deliveries, while Layout 3 underperforms.

methods, the agents have yet to consistently achieve the target number of soup deliveries in this environment, highlighting opportunities for further tuning or architectural adjustments.

Overall, these evaluations underscore the importance of environment complexity in multi-agent training, and the need for increasingly sophisticated approaches to handle coordination in more challenging settings. For future work, improving Layout 3 performance can involve incorporating more advanced coordination mechanisms such as communication protocols or hierarchical policies to better manage complex tasks. Additionally, exploring alternative reward structures or multi-agent credit assignment techniques could also enhance cooperation. Finally, scaling up model capacity and leveraging more extensive hyperparameter tuning might yield further improvements in such challenging layouts.

3.3 Conclusions

This project shows the effectiveness of the Multi-Agent Proximal Policy Optimization (MAPPO) on the Overcooked-AI environment. It showcases improvements in agent coordination by implementing a centralized critic and decentralized actor network. Introducing the mini-batch rollout buffer further enganced training stability and sample efficiency, specially for layouts 1 and 2. While simpler layouts achieved optimal performance,

References

[1] M. Carroll, R. Shah, M. Ho, T. Griffiths, P. Abbeel, and A. Dragan. On the utility of learning about humans for human-ai coordination. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2019.

[2] Sam Devlin and Daniel Kudenko. Potential-based shaping and q-value initialization are equivalent. In *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, 2014.

[3] Jakob N Foerster, Gregory Farquhar, Tim Afouras, Nantas Nardelli, and Shimon Whiteson. Counterfactual multi-agent policy gradients. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 32, 2018.

[4] Pablo Hernandez-Leal, Bilal Kartal, and Matthew E Taylor. A survey and critique of multiagent deep reinforcement learning. *Autonomous Agents and Multi-Agent Systems*, 33:750–797, 2019.

[5] R. Lowe, Y. Wu, A. Tamar, J. Harb, P. Abbeel, and I. Mordatch. Multi-agent actor-critic for mixed cooperative-competitive environments. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2017.

[6] Andrew Y Ng, Daishi Harada, and Stuart J Russell. Policy invariance under reward transformations: Theory and application to reward shaping. *ICML '99: Proceedings of the Sixteenth International Conference on Machine Learning*, pages 278–287, 1999.

[7] Gabriel Schroeder, Andres Tampuu, Shagun Bansal, Daniel Kudenko, and Doina Precup. Multiagent reinforcement learning in sequential social dilemmas. In *Proceedings of the 19th International Conference on Autonomous Agents and MultiAgent Systems (AAMAS)*, 2020.

[8] John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. High-dimensional continuous control using generalized advantage estimation. In *arXiv preprint arXiv:1506.02438*, 2016.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. In *Advances in Neural Information Processing Systems*, pages 7115–7123, 2017.

[10] Ziyu Wang, Hao Li, Mingming Liu, Haoyuan He, Yaodong Wang, and Shimon Chen. Learning to cooperate via policy search. *arXiv preprint arXiv:1809.07152*, 2018.

[11] Yong Yu, Jun Zhang, Weizhe Wang, Tianhao Gao, and David Silver. The surprising effectiveness of ppo in cooperative multi-agent games. *arXiv preprint arXiv:2103.01955*, 2021.