Game Theory Project Report

Weiji Xie * **Tianhua Li** * **Kailing Wang** * **Yuchen Mao** * **Dingju Wang** *

Abstract

In this report, we focus on solving two substrates from Melting Pot Contest 2023[1]. For the substrate PD Matrix, we propose **an efficient rule-based solution** with an automatic Policy Sampler. For both substrates, we develop a novel RL-based method, **RIN**. Experiments show that our methods outperform the IPPO baseline and an open-source solution (TESS[5]) by a large margin. We **win a championship** in all two tracks of the JIDI[3] class competition.

1 Introduction

The two substrates are Prisoners Dilemma in the Matrix Repeated (PD Matrix 1.1 for short) and Clean Up 1.2, with 4 and 1 predefined scenarios respectively[1]. We will first introduce the detailed rules of the two substrates, and provide the overview of our report.

1.1 PD Matrix

In this substrate, both players could collect different colored resources, GREEN stand for cooperate and RED stand for defect, as shown in Fig. 1a. When two players meet each other, they compare their storage and determine whether to cooperate or not, the matrix is the same as the classical prisoner's dilemma,

$$A_{\text{row}} = A_{\text{col}}^T = \begin{bmatrix} 3 & 0 \\ 5 & 1 \end{bmatrix}. \tag{1}$$

Initially, the two players will be placed randomly along the outer wall with 1 resource of each color. After collecting m_i GREEN resources and n_i RED resources, the inventories of player i become $[m_i + 1, n_i + 1]$. The policy distribution of player i is calculated as $p_i = \left[\frac{m_i+1}{m_i+n_i+2}, \frac{n_i+1}{m_i+n_i+2}\right]$. The reward for the two players are

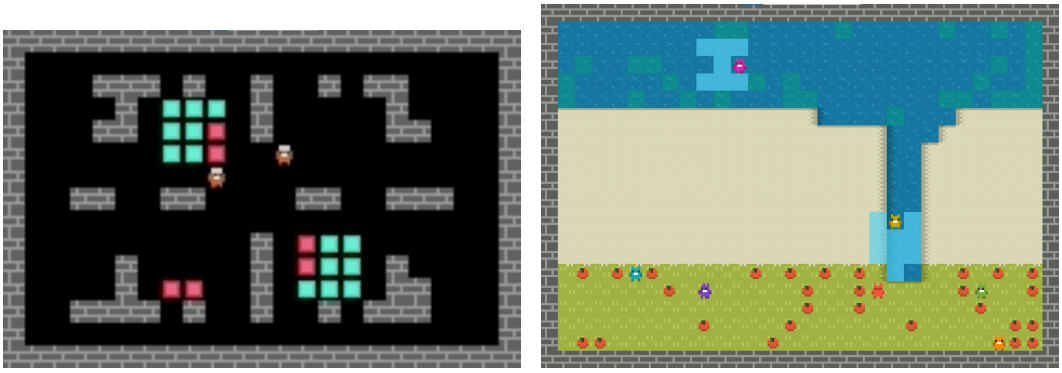
$$r_1 = p_1 A_{\text{row}} p_2^T, r_2 = p_1 A_{\text{col}} p_2^T. \tag{2}$$

Predefined scenarios include 1) random cooperator, 2) fully cooperator, 3) fully defector, and 4) hair-trigger grim reciprocator, i.e. one who initially cooperates but, if defected once, will retaliate by defecting forever after. Besides, there is an open scenario where agents need to compete with each other in JIDI[3] Platform.

1.2 Clean Up

In this substrate, 7 players are rewarded for collecting apples (reward +1). The river accumulates pollution at a constant rate. Apples grow in an orchard at a rate inversely related to the cleanliness of a nearby river. Beyond a certain pollution threshold, the apple growth rate in the orchard drops to zero, as demonstrated in Fig. 1b.

*Author names are randomly sorted. For the detailed contribution of each author, see Sec. 6. Our code is public `https://github.com/Shi-Soul/gtproj`



(a) PD Matrix (b) Clean Up

Figure 1: Overview of the two substrates

In the predefined scenario called Clean Up 7, two players sampled from the focal population join five players sampled from the background population that implements the suspicious conditional cooperation strategy, i.e. the background bot cleans whenever at least two other players are cleaning.

1.3 Report Overview

This report demonstrates the method used to solve the above two substrates.

For ruled-based method, we first conducted a deep analysis for PD Matrix, then described our method including a Policy Sampler (Sec 2.1) and efficient coding for agent behavior 2.2.

For RL method, we first analyze previous methods, IPPO baseline (Sec. 3.1) and TESS[5] (Sec 3.2) in Sec. 4, then propose our method **RIN** (Sec. 3.3), which is based on TESS and equipped with a powerful reward shaping.

2 Rule-based Method for PD Matrix

To further understand the PD Matrix game, we looked deep into this substrate. For fixed scenarios, the key to acquiring high rewards is: 1) efficiently collecting resources, 2) finding an appropriate time to shoot, and 3) detecting the background player's policy and finding the optimal reaction. Since the total timestamp is fixed, efficient collecting and shooting would allow the focal player to have more game rounds, and an optimal reaction would help the agent get more reward in each round. Whether to collect enough resources or before that end the game when possible is an important trade-off to consider. For open scenarios, the goal is to earn more reward than the opponent so the policy selection becomes more important.

Based on the above considerations, we designed a rule-based agent for both fixed and open scenario, which automatically determine the collection policy and efficiently find its way to the target.

2.1 Policy Sampler

To determine a good collection target, we need to estimate the policy of the opponent. The Bayes Model and Kalman Filter are popular choices to estimate a distribution based on observations. However, PD Matrix only lasts for 500 timestamps. For each round, localization and collection usually finish in less than 20 steps, but interaction stiffness and blank time between rounds can take over 25 steps. That is, there are typically 8-12 game rounds, and we are to estimate the distribution on usually less than 10 observations. Simply fitting the observation into a Gaussian model is enough for our task.

Initially, we set a fixed policy. Considering the partner might be a hair-trigger grim reciprocator, the initial policy is $\text{target}_0 = [5, 1]$ to play cooperate.

Let the policy distribution $p = [coor, 1 - coor]$. According to equation 2, given the reward r_t at round t and the cooperate probability of the focal player $coor_{f,t}$, the cooperate probability of the background player is $coor_{b,t} = \frac{r_t + coor_{f,t} - 1}{4 - coor_{f,t}}$.

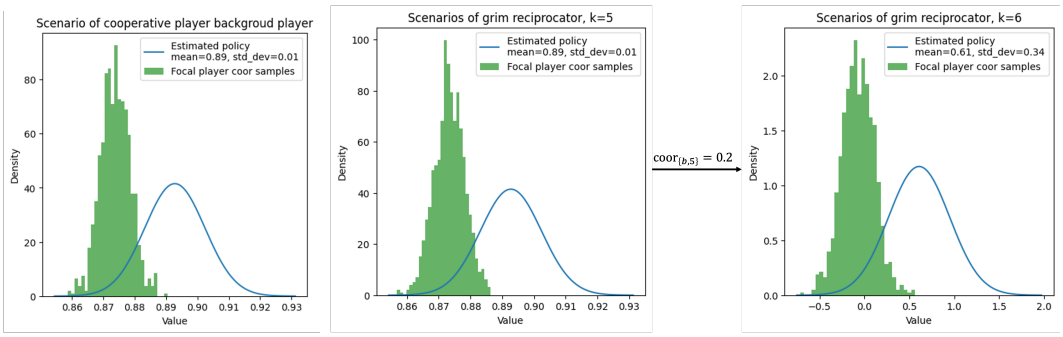
From round $k > 1$, we fit a Gaussian on the history cooperate probabilities of the background player weight by time. To be exact,

$$\{coor_{b,t} \text{ with weight } \frac{1}{k-t} \text{for } t \text{ in } [1, 2 \dots k-1]\} \sim \mathcal{N}(\mu, \sigma^2). \tag{3}$$

To get a sample with a higher probability, we sample from $\mathcal{N}(\mu, (\frac{\sigma}{2})^2)$, and add some probability to the defect,

$$\text{Sample } \hat{coor}_{b,k} \sim \mathcal{N}(\mu, (\frac{\sigma}{2})^2), \hat{coor}_{f,k} \leftarrow \hat{coor}_{b,k} - \lambda_d \sigma, \lambda_d = 2. \tag{4}$$

As shown in Fig. 2a, if the background player holds a fixed policy, the σ is small enough and this policy sampler would act as **an action imitator**. However, suppose the background player suddenly shifts policy, like a hair-trigger grim reciprocator. In that case, the sampler can react optimistically, **quickly shifting to defect**, as in Fig. 2b. The $\lambda_d = 2$ is designed for the random background player. If the background player plays random policy or cooperate/defect by 50%, the estimated σ should be around 0.3. Setting $\lambda_d = 2$ ensures the focal player plays **defect towards a random background player**.



(a) Fully collaborator (b) Policy shift on trigger

Figure 2: Gaussian distribution and histogram of sampled best coordinates

We select the collection target with the closest $coor_{f,k}$ from a pre-defined collection set as the target collection.

2.2 Efficient Collection

This part is more of a problem of coding efficiency. In this substrate, the MAP is fixed, while the color of resources is randomized only at the beginning of the scenario. We can cache the global map M_G in the agent's memory for localization. We use direct matching to filter possible positions. Since the agent should be along the outer wall initially, we let the agent walk away from the wall for faster localization. After that, the agent tends to keep its direction. The possible weight of the movement in each direction r is:

$$p(r) = \begin{cases} 3, \text{if the agent can walk along } r \text{ and } r \text{ is the last direction} \\ 3, \text{if the agent can walk along but not against } r \\ 1, \text{if the agent can walk both along and against } r \\ 0, \text{if the agent cannot walk along } r \end{cases}, \tag{5}$$

and a **Direction Sampler** samples a direction according to the weight above.

After localization, we used the Aster algorithm to find the path to the cards. Given the collection target from the policy sampler, we set the color with more need as the priority, collect this color of

resource until enough, and then collect another color. Local observations will be cached in memory until the target is shifted to prevent the agent from getting stuck near resources.

After collection, the agent wanders to the center of the grid. If the opponent is in sight on the way, the agent will chase after it until it can shoot the opponent. The overall framework of the rule-based method is illustrated in Alg. 1.

Algorithm 1 Rule-based PD Matrix

Input: Observation M_{rgb}, Reward r
Output: Action
update memory, process M_{rgb} to get local grid map M_L
cache observation in M_L
collection target $\leftarrow$ Policy_Sampler(r) (Sec. 2.1), determine priority color c_p
if not localized **then**
    match M_L with M_G, compare with history match
    if only one possible match **then**
        localized $\leftarrow$ True
    end if
    return walk along the direction sampled by Direction_Sampler (Eq. 5)
end if
if collection not finished **then**
    return follow the Astar path to the target card center or card of c_p
else
    return chase the opponent and shoot
end if

3 Reinforcement Learning based methods

3.1 Baseline: Independent PPO

Independent Proximal Policy Optimization (Independent PPO or IPPO) is a variation of the Proximal Policy Optimization (PPO) algorithm, which applies the PPO algorithm to multiple agents independently. Each agent acts independently and optimizes its own policy without considering the policies or actions of other agents directly.

After numerous experiments, we ultimately discarded the IPPO algorithm due to its overall low performance for the following reasons:

- The Ray Rllib[4] implementation of IPPO is quite complex, making it difficult to modify and optimize.
- IPPO showed low performance in two substrates: for the PD Matrix, the training return fluctuated around 20, and for Clean Up, it stabilized at 0. We believe this comes from the inherence of independent learning and those games, as both substrates contain social dilemmas. For example, for Clean Up, all agents stay at the orchard while nobody cleans, which is indeed a Nash equilibrium.

3.2 TESS

Overall, TESS [5] is a multi-task variant of the PPO algorithm, which is trained in a self-play paradigm with parameters sharing. For each task, it designs two tasks for the model to train on.

3.2.1 Model Architecture

TESS trains only one neural network for all agents. The network architecture comes from IMPALA[2], which consists of three parts:

1. Convolutional feature extractor;
2. LSTM memory module;

3. Output head for multi-task value function and policy function.

The network models the value function, policy function, and task estimation (i.e., which task should the agent take now) for each task.

$$f(s) = [V_1(s), V_2(s), \pi_1(s), \pi_2(s), e_1(s), e_2(s)] \tag{6}$$

The value functions for each task are defined and optimized the same as in the single-task setting, with their respective rewards. Besides, PopArt[6] is used for adaptive reward normalization when learning value function.

The agent's actual policy is a weighted sum of both single-task policies.

$$\pi(s) = w_1(s)\pi_1(s) + w_2(s)\pi_2(s), w(s) = \text{Softmax}(e(s)) \tag{7}$$

When optimizing policy, each task is equally treated, and the final policy loss is the sum of two PPO policy losses.

$$\begin{cases} L = L_1 + L_2 \\ L_i = L_{\text{PPO-Clip}}(V_i, \pi, \pi_{\text{old}}) \end{cases} \tag{8}$$

3.2.2 Multi-task Reward Shaping

TESS divides PD Matrix into collecting resources and interaction tasks and divides Clean Up into cleaning water and collecting apple tasks. Each task has a unique reward function.

The reward function for PD Matrix is:

$$\begin{cases} r^t_{\text{Collect}} = 0.1 * N_t[\text{number of resources collected by any agent}] \\ r^t_{\text{Interaction}} = (r^t_1 + r^t_2)\mathbb{1}_t[\text{agents interact}] \\ r^t_1, r^t_2 \text{ is the matrix game reward for each agent} \end{cases} \tag{9}$$

The reward function for Clean Up is as follows, where r^t_{Clean} and r^t_{Apple} are the rewards for the current timestep. In a nutshell, it additionally rewards the agent for cleaning water. It keeps 10 steps clean history for each agent. Whenever someone collects an apple, every cleaner gets the sum of its clean history as a reward.

$$\begin{cases} T = 10 \\ H_t = \sum_{i=t-T}^{t-1} r^i_{\text{clean history}} \\ r^t_{\text{clean history}} = 0.01 * N_t[\text{number of pollution current agent cleans}] \\ r^t_{\text{Clean}} = 10 * r^t_{\text{clean history}} + H_t * N_t[\text{number of apple collected by any agent}] \\ r^t_{\text{Apple}} = 0.1 * \mathbb{1}_t[\text{current agent collects apple}] \end{cases} \tag{10}$$

Both reward function designs aim to facilitate the training process. In PD Matrix, it makes two agents have the same goal to play, eliminating the oscillation of strategy and resulting in a full cooperator policy. In Clean Up, it consistently encourages agents to clean water actively without ignoring collecting apples.

3.2.3 Weakness

With experiments, we found that the reward shaping in TESS is effective but far from perfect.

In PD Matrix, the agent can only perform a fully cooperative strategy, which prevents it from properly reacting to a defector opponent.

In Clean Up, returns are unevenly distributed, and production is surplus. Cleaner agents are so hardworking that they don't leave their work and can't collect apples. As a result, while few agents

are collecting apples, the speed of picking apples is slower than the growth of apples, leaving apples unpicked when reaching the time limit.

With these insights, we took the thought of reward shaping and re-designated the reward to conquer the weakness shown in TESS.

3.3 RIN: Reward shaping Is all you Need

For PD Matrix, considering that a full defector strategy can get higher scores in the fixed scenarios and is very competitive in the open scenario, we want to design a corresponding reward setting for learning it. Besides, since effective game-playing, especially the ability to quickly find the opponent and end a game round is key to obtaining a high reward, we want to encourage agents to interact more with opponents. Hence, we design the following reward function.

$$\begin{cases} r^t_{\text{Collect}} = (0.1 + \frac{n+2}{m+n+4}) * \mathbb{1}_t[\text{current agent collect resources}] \\ r^t_{\text{Interaction}} = (0.1 + \frac{n+2}{m+n+4} + \mathbb{1}_t[\text{for the first time the opponent is seen}])\mathbb{1}_t[\text{agents interact}] \\ m, n \text{ is the number of cooperation and defection resources the current agent has collected} \end{cases} \tag{11}$$

For Clean Up, we wish agents could introduce shift work, where each agent works and enjoys fairly. To achieve this, our key idea is to make cleaner agents take a rest periodically, and when there are not enough cleaner agents, some of the agents who gain without effort must go out and clean the water.

In our design, the original task division is reserved. Each agent keeps a clean history of 400 timesteps, which enables us to time the work-enjoy period. $r^t_{\text{clean history}}$ contains an additional factor negatively correlated to clean history, which ensures the longer an agent cleans, the less clean reward it gets. The term $\frac{H_t}{0.01 \cdot 5}$ is used to approximately reflect the number of timesteps the agent cleans, where 0.01 should offset the factor 0.01 in $r^t_{\text{clean history}}$ and 5 comes from the fact that one clean action can remove many pollution blocks. Finally, we add the factor 0.01 in the second term of r^t_{Clean} because a longer clean history is kept.

$$\begin{cases} T = 400 \\ H_t = \sum_{i=t-T}^{t-1} r^i_{\text{clean history}} \\ r^t_{\text{clean history}} = 0.01 * \text{Sigmoid}(T - \frac{H_t}{0.01 \cdot 5}) * N_t[\text{number of pollution current agent cleans}] \\ r^t_{\text{Clean}} = 10 * r^t_{\text{clean history}} + 0.01 H_t * N_t[\text{number of apple collected by any agent}] \\ r^t_{\text{Apple}} = 0.1 * \mathbb{1}_t[\text{current agent collects apple}] \end{cases} \tag{12}$$

Although reward shaping is described in a series of simple formulas, it's a super powerful tool to implement our will into RL algorithms. We will see that these reward functions work pretty well, and the trained agent faithfully follows our design purpose in Sec. 4.

In high-level, multi-task reward shaping in multi-agent settings, as a tool for facilitating learning, can be viewed as explicit team advantage decomposition. By giving the agent multiple learning directions (possibly conflicting with each other), the agent needs to learn how to accomplish each task and make a balance, hence taking the benefits of the teammate into consideration. In fact, it projects the contradiction between multiple agents onto the conflict of different tasks of a single agent, the latter, however, is much easier to deal with.

4 Experiment Results

4.1 PD Matrix

For each experiment setting, we run the game 10 times and calculate the mean and std as results. In the JIDI[3] competition, we submit RIN to the fixed scenarios since it has a slightly higher score, and submit Rule-based agent to the open scenario since it's more adaptive to unknown opponents.

4.1.1 The effect of Policy Sampler

To estimate the estimation ability of the policy sampler, we designed an upper bound. The upper-bound method uses the same framework as the rule-based agent but knows the background players' policy. In the given four scenarios, the upper bound plays defect, defect, defect, and cooperate respectively. We also set a baseline that does not have a policy sampler, and collect randomly instead. Table. 1 shows the results of the baseline, policy sampler, and upper bound.

Method	**Random Baseline**	**Policy Sampler**	**Upper Bound**
Reward	74.13 ± 14.80	90.14 ± 6.72	108.26 ± 16.32

Table 1: Result of Policy Sampler

Results show that the Policy Sampler does improve the overall reward. The gap between Policy Sampler and the upper bound is caused by the around 10 games playing with a full collaborator. The upper bound always defects to get a 5 reward while the Policy Sampler will try to cooperate and get 3. However, if we betray the full collaborator, we'd lose the corresponding 2 reward from the grim reciprocator.

4.1.2 RL-based methods

We use all methods in Sec. 3 to train an agent respectively and evaluate them in the fixed scenarios. Table. 2 shows the results. It shows that our method RIN outperforms all RL methods and is on par with our rule-based agent.

RIN and the rule-based agent have similar outcomes because:

- The game strategy of RIN is as good as the rule-based agent in evaluation, since it chooses to defect to both the grim reciprocator and the full collaborator.
- RIN as an RL agent has a more flexible behavior pattern than the rule-based agent. It does not require extra steps for localization, and after collecting resources it can actively go to find and chase the opponent even when the opponent is not in the view.

Method	**IPPO**	**TESS**	**RIN**
Reward	18.60 ± 6.97	80.08 ± 20.04	91.85 ± 10.38

Table 2: Result of RL-based Agent for PD Matrix

4.2 Clean Up

For each experiment setting, we run the game 100 times and calculate the mean and std as results. We use all methods in Sec. 3 to train an agent respectively and evaluate them both in Clean Up 7 and by self-play.

Our agent has a higher score than both IPPO and TESS in the Mean Agent Score (i.e., focal per capita returns). Additionally, in both evaluation scenarios, our agent achieves a smaller value of the difference between the Highest Agent Score and the Lowest Agent Score than TESS. It indicates that our method indeed improves the fairness of returns distribution without damaging collective interests. By observing sample videos of game replay, we can confirm that there is a shift work mechanism that emerged from these agents.

5 Conclusion

In this project, we design and implement a Rule-based method for the PD Matrix, which is composed of a novel Policy Sampler and an efficient behavior logic implementation. Moreover, our proposed RL method, RIN, achieves substantial performance gains over previous work in both the PD Matrix and Clean Up domains.

Method	/	IPPO	TESS	RIN
Self-Play	Mean Agent Score	0.00 ± 0.00	380.90 ± 129.66	**450.80** ± 174.14
	Lowest Agent Score	0.00 ± 0.00	19.60 ± 52.70	290.68 ± 155.58
	Highest Agent Score	0.00 ± 0.00	695.79 ± 205.59	571.10 ± 213.95
Clean Up 7	Mean Agent Score	57.75 ± 73.35	902.46 ± 309.81	**1161.38** ± 377.92
	Lowest Agent Score	46.00 ± 70.76	781.98 ± 328.12	1141.93 ± 376.55
	Highest Agent Score	69.50 ± 78.73	1022.94 ± 325.46	1180.83 ± 380.06

Table 3: Result of RL-based Agent for Clean Up

It's worth noting that we successfully drive agents to adopt shift work mechanisms as a solution for social dilemmas, which unlocks cooperative potential, drives socially beneficial outcomes, and has a significant social value not limited to the current scenario.

6 Contribution

- Kailing Wang: Design and implement the rule-based method. Experiments for the rule-based part. Report writing. Presentation.
- Weiji Xie: Code and experiments for RL-based methods. Report Writing. Presentation.
- Tianhua Li: Review code of the Melting Pot Env. Part of the code and experiments. Report Writing.
- Yuchen Mao: Method survey. Part of experiments. Report Writing.
- Dingju Wang: Env analysis. Part of experiments. Report Writing.

References

[1] John P. Agapiou, Alexander Sasha Vezhnevets, Edgar A. Duéñez-Guzmán, Jayd Matyas, Yiran Mao, Peter Sunehag, Raphael Köster, Udari Madhushani, Kavya Kopparapu, Ramona Comanescu, DJ Strouse, Michael B. Johanson, Sukhdeep Singh, Julia Haas, Igor Mordatch, Dean Mobbs, and Joel Z. Leibo. Melting pot 2.0, 2023.

[2] Lasse Espeholt, Hubert Soyer, Remi Munos, Karen Simonyan, Volodymir Mnih, Tom Ward, Yotam Doron, Vlad Firoiu, Tim Harley, Iain Dunning, Shane Legg, and Koray Kavukcuoglu. Impala: Scalable distributed deep-rl with importance weighted actor-learner architectures, 2018.

[3] Collective Decision Intelligence Lab Institute of Automation Chinese Academy of Sciences. Jidi. `http://www.jidiai.cn/`, 2024.

[4] Eric Liang, Richard Liaw, Philipp Moritz, Robert Nishihara, Roy Fox, Ken Goldberg, Joseph E. Gonzalez, Michael I. Jordan, and Ion Stoica. Rllib: Abstractions for distributed reinforcement learning, 2018.

[5] utkuearas. Meltingpot-tess-v1. `https://github.com/utkuearas/MeltingPot-Tess-v1`, 2023. Accessed: 2024-05-31.

[6] Hado van Hasselt, Arthur Guez, Matteo Hessel, Volodymyr Mnih, and David Silver. Learning values across many orders of magnitude, 2016.