

# Game Theory Research Project - Week 5 Report

Jessica Opsahl-Ong

July 2022

## Fixed Strategies

The first thing I did to introduce game theory concepts to this research was encode playable common iterative strategies to the prisoners dilemma. The strategies are as follows:

1. **Cooperator:** Cooperates on every move
2. **Defector:** Defects on every move
3. **Random:** Makes a random move with 50/50 chance
4. **Tit for Tat:** Makes a random move built on probability distribution
5. **Grim Trigger:** Cooperates, until the opponent defects, and thereafter always defects.
6. **Tit for Two Tat:** Cooperates on the first two moves, and defects only when the opponent defects two times.
7. **Two Tit for Tat:** Same as Tit for Tat except that it defects twice when the opponent defects.
8. **Gradual:** Cooperates on the first move, and cooperates as long as the opponent cooperates. After the first defection of the other player, it defects one time and cooperates two times. After the nth defection it reacts with n consecutive defections and then cooperates twice.
9. **Soft Majority:** Cooperates on the first move, and cooperates as long as the number of times the opponent has cooperated is greater than or equal to the number of times it has defected, else it defects.
10. **Hard Majority:** Defects on the first move, and defects if the number of defections of the opponent is greater than or equal to the number of times it has cooperated, else cooperates.
11. **Remorseful Prober:** Like Tit for Tat, but it tries to break the series of mutual defections after defecting 5 times.
12. **Soft Grudger:** Like GRIM except that the opponent is punished with D,D,D,D,C,C.
13. **Prober:** Starts with D,C,C and then defects if the opponent has cooperated in the second and third move; otherwise, it plays TFT.

Furthermore, each strategy can be "noisy" by utilizing the "unexpected prob" parameter, which means that the agent plays unexpectedly "unexpected prob" percent of the time.

Code: <https://github.com/jessica-ops/game-theory/blob/main/strategies/strategy.py>

## Strategy Likelihood

With these strategies, I implemented a way to determine which strategy is being used in a game where one of the players is using one of the 13 strategies above. This is built on Bayesian decision processes and updates the likelihood of each strategy after each round of the game. The process begins with a flat prior and the posterior is calculated each round by the following algorithm:

---

**Algorithm 1:** Calculating Strategy Likelihood

---

**Input** : list of all moves the agent has played thus far,  $agentActs$   
**Input** : list of all moves the opponent has played thus far,  $oppActs$   
**Output:** Probability distribution over all strategies indicating likelihood of each strategy given observed data

- 1 **for** each  $s$  in strategies **do**
- 2   | compute  $P(agentActs, oppActs | \text{strategies} = s)$
- 3  $P(agentActs, oppActs) = \sum_{\text{each } s \text{ in strategies}} P(agentActs, oppActs | \text{strategies} = s) * P(\text{strategies})$
- 4  $P(\text{strategies} | agentActs, oppActs) = [P(agentActs, oppActs | \text{strategies}) * P(\text{strategies})] / P(agentActs, oppActs)$
- 5 **return**  $P(\text{strategies} | agentActs, oppActs)$

---

code: [https://github.com/jessica-ops/game-theory/blob/main/probabilistic\\_playing/strategy\\_inference.py](https://github.com/jessica-ops/game-theory/blob/main/probabilistic_playing/strategy_inference.py)

### Computing likelihood for individual strategies

Because each strategy has distinct behavior, there is a different function for each strategy for the computation in line 2. Depending on the class of strategy, there are generally different ways of computing the likelihood. However, there are variations within the groups as well.

#### Fixed Strategies - Cooperator, Defector

These always have the same expected action. So for a new round of the game, if the agent has played its respective expected action, the likelihood = previous likelihood \* (1 - unexpected probability)

#### Responsive Strategies - Tit for Tat, Tit for Two Tat, Two Tit For Tat, Remorseful Prober, Soft Grudger

All of these strategies rely on a fixed memory of the previous game to play. The most standard is Tit for Tat, whose expected move is always the opponent's previous move. Meaning given a new round  $r$  of the game, if  $agentAct_r = oppAct_{r-1}$ , then likelihood = previous likelihood \* (1 - unexpected probability).

This operates similarly for Tit for Two Tats and Two Tits for Tat, except that there is a memory of two instead of one previous actions required. Similarly, Remorseful Prober requires a memory of five.

Soft Grudger technically is a responsive strategy; however, it can be difficult to discern where in the punishing sequence the agent is by just looking back. So in actual practice, I use a similar strategy to Gradual to compute the likelihood for Soft Grudger.

#### Memory Strategies - Grim Trigger, Gradual, Soft Majority, Hard Majority, Prober

These are strategies which must retain some memory of the whole game thus far in order to play their ensuring action. For some of the strategies, the information which they "remember" is relatively straightforward, making their likelihood straight forward to compute.

For Grim Trigger, if the opponent has ever played a 1, Grim Trigger acts as a Defector. Otherwise, it acts as a Cooperator. This means that the likelihood can be computed in a similar way to the Fixed

Strategies.

Prober also adopts a new strategy depending on its history. Depending on the outcome of the first three rounds of the game, Prober will either become a Defector or play Tit for Tat. This means that the likelihood can be computed as either one of those strategies. For Soft Majority and Hard Majority, their action depends on the ratio of opponent defects to total plays. This can be calculated using the action history and easily determine if the expected move is 1 or 0.

Gradual is more complicated in that the number of times that the opponent defects impacts the following sequence that Graudal will play. Because of this, I found that it works best to re-calculate the likelihood from the beginning for each round, instead of relying on the previously calculated likelihood. By doing this, I can keep track of the number of defects of the opponent at each round in the game and subsequently keep track of where Gradual should be in its sequence. For each expected and unexpected move, the likelihood is updated accordingly.

## Random

Because Random will play a random move each round, the likelihood of any move is always 0.5. Meaning the overall likelihood =  $0.5 * \text{previous likelihood}$ .

## Determining the Unexpected Probability

Despite being used in the computation of the likelihood, the unexpected probability is also an unknown and must be estimated while trying to determine which strategy the agent is playing. I currently have a rudimentary way of updating the unexpected probability. The functions which calculate likelihood for each strategy also take in as an input and return as an output the number of unexpected moves the agent has played thus far. The unexpected probability = number of unexpected moves \* epsilon. The epsilon value is a parameter of the Likelihood class. Currently, I typically set the value to 0.01. This means that when the number of rounds = 100, unexpected probability = number of unexpected moves / total moves. And before then, it isn't too hasty with increasing the value of the unexpected probability.

code: <https://github.com/jessica-ops/game-theory/blob/main/strategies/likelihood.py>

# 1 Reinforcement Learning Agents

In order to create an agent which is responsive to different strategies it encounters, I have been using RL to train policies which play the prisoner's dilemma. I have been using the Poximal Policy Optimization (PPO) algorithm from Stable Baselines 3 and made a custom environment in line with the OpenAI Gym framework.

The environment is set up such that one episode is a full game consisting of a set number of rounds (typically I have trained it on 100 rounds) and one step is a single round of the game. The agent is rewarded based off the payout matrix. The observations which are given to the policy to select actions include a history of the previous 10 rounds of actions (both the agent's actions and opponent's). I have experimented with including other observations - including average number of opponent and agent defections as well as the round number.

When starting out, in order to get a feel for the responsiveness of an RL trained agent, I trained against all the strategies. A random opponent strategy was chosen for each episode and the agent would play against it. I also trained agents against the policies which I had saved.

To examine the responsiveness and general strategies of these agents, I played against them and created a "tournament" in which each agent would play games against every other agent and strategy.

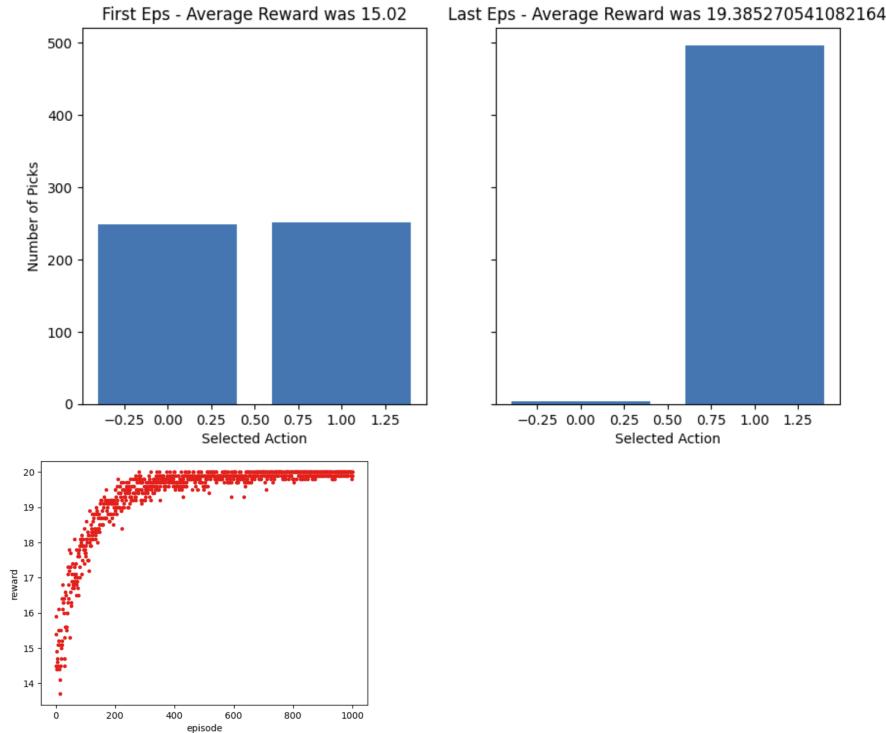
Currently, I am training policies against a single strategy. Because these policies only see a single oppo-

ment when training, it's belief will be that any opponent it faces will be using that strategy. This will allow us to test IRC on a game theory set-up.

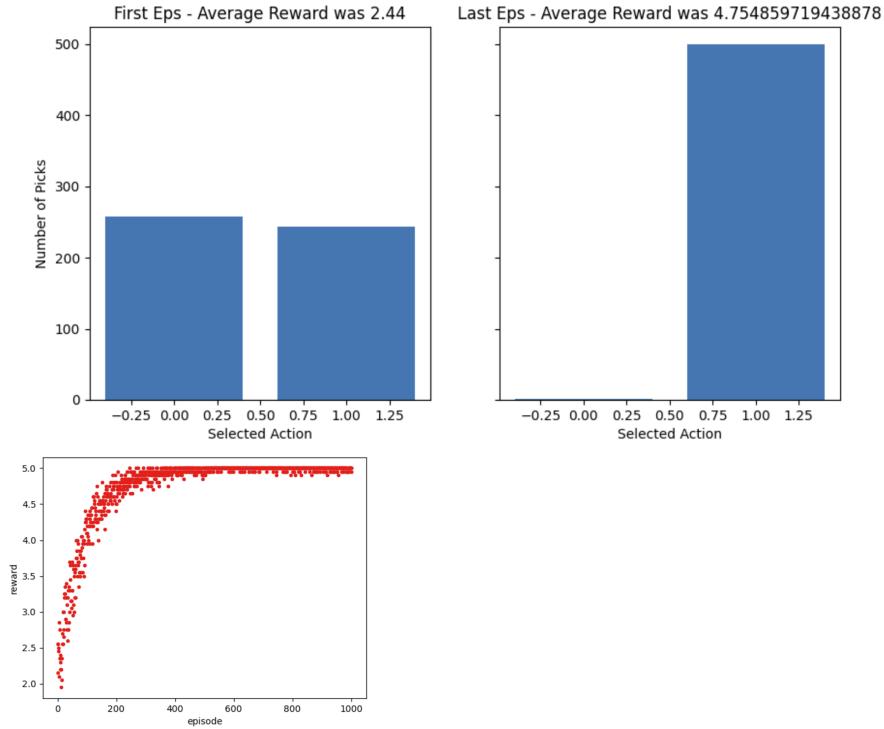
## Results

For the training of each policy, I logged which action it took at each step and the reward for each episode. I used this to create a graph of the average reward for each episode and a distribution of the actions it took at the beginning of training (a random distribution) versus at the end of training (what it has learned).

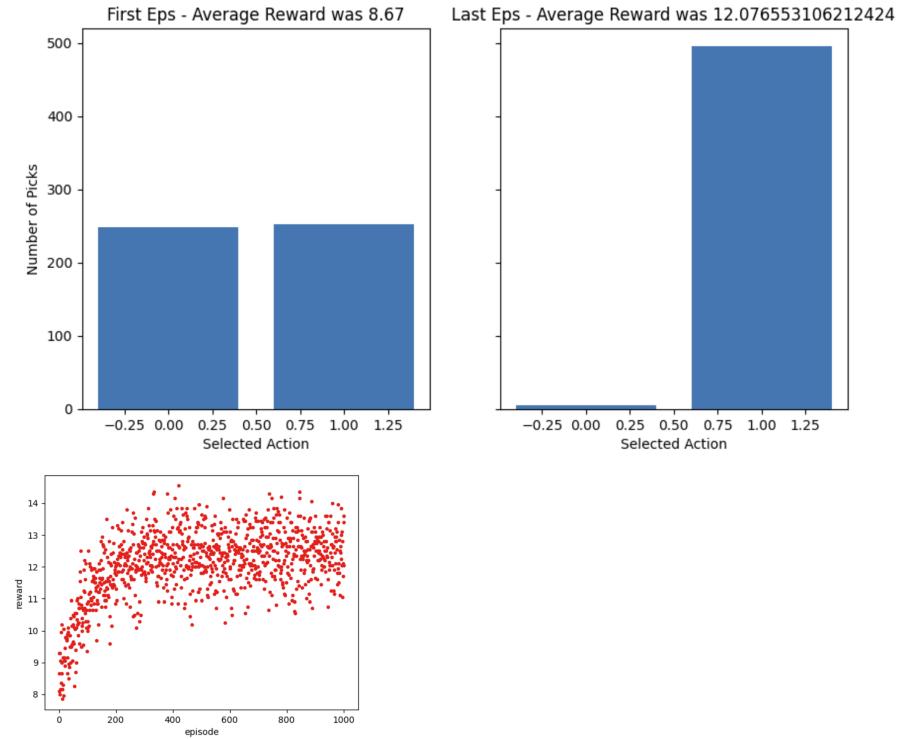
### Cooperator



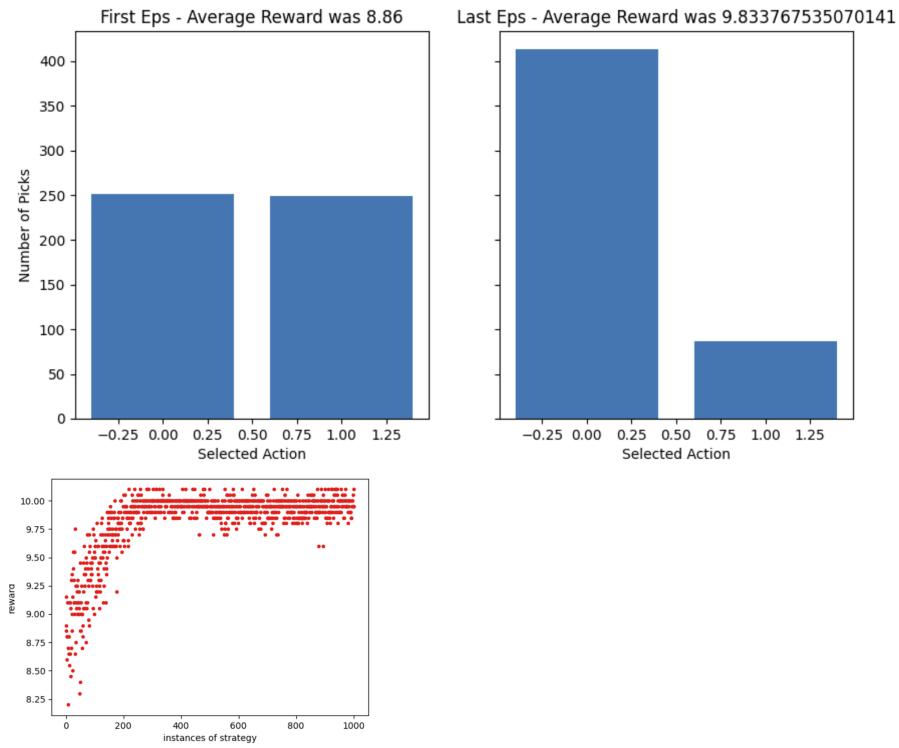
## Defector



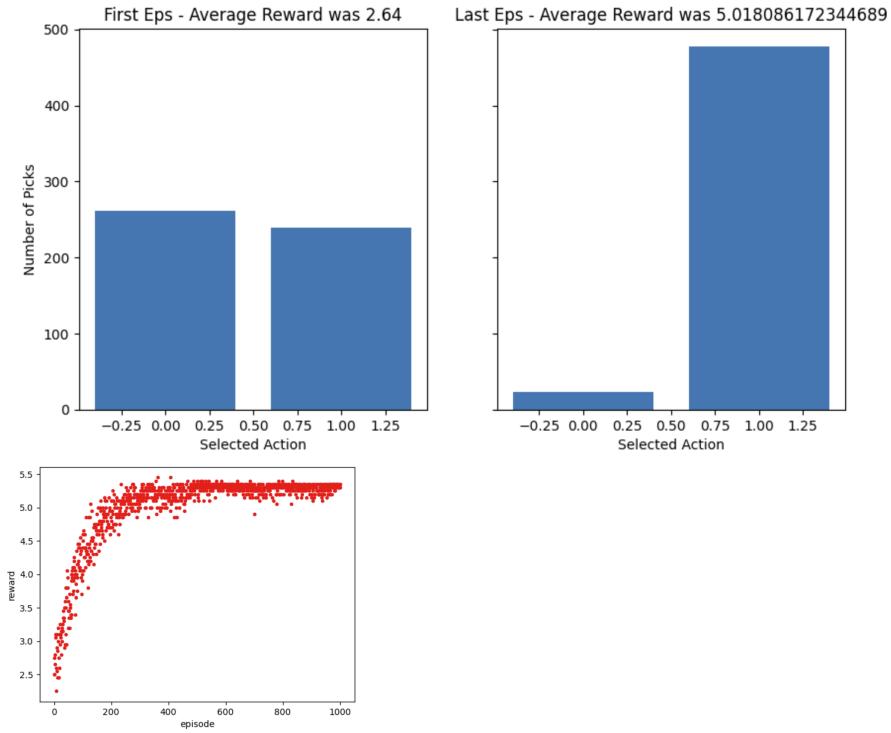
## Random



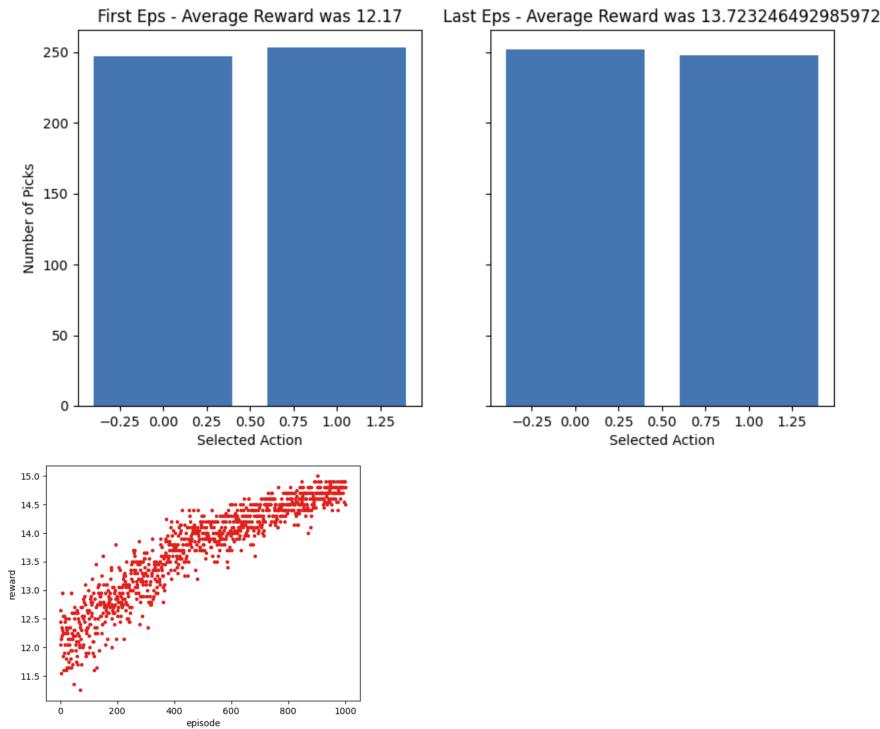
## Tit for Tat



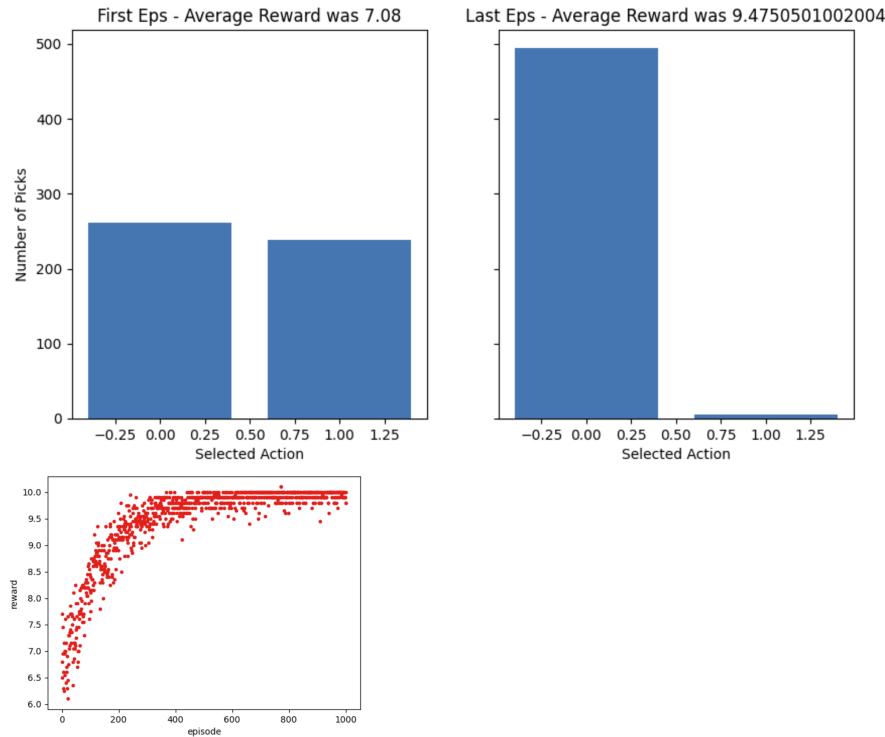
## Grim Trigger



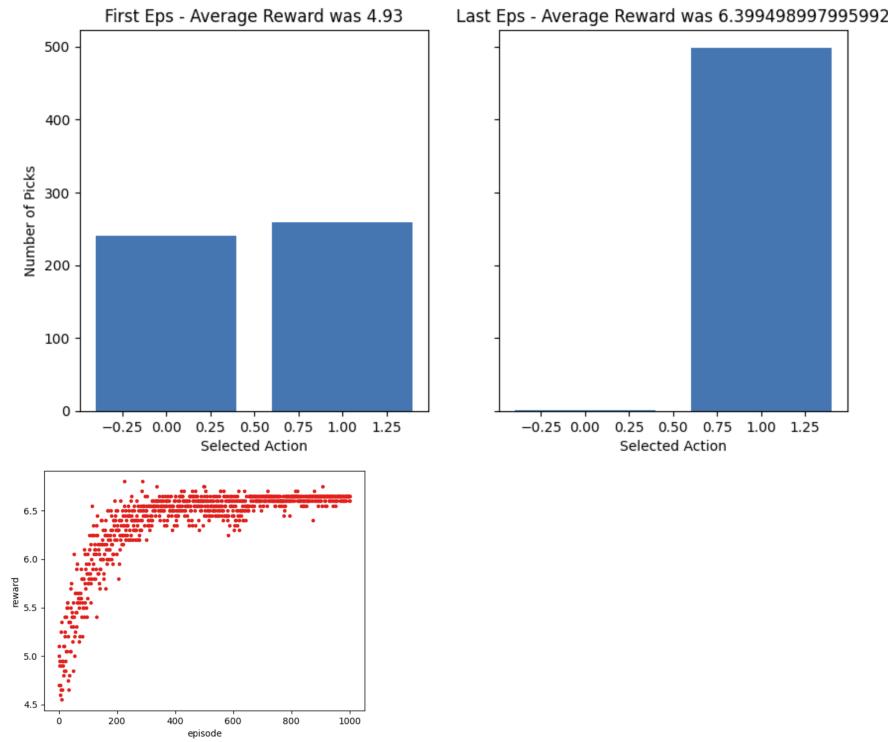
## Tit for Two Tat



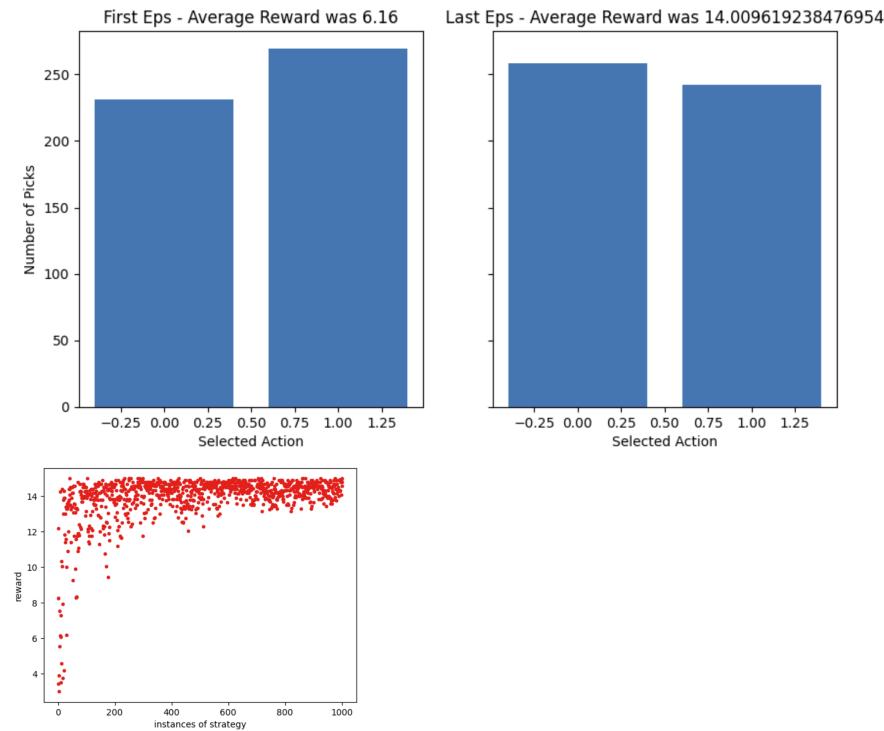
## Two Tit for Tat



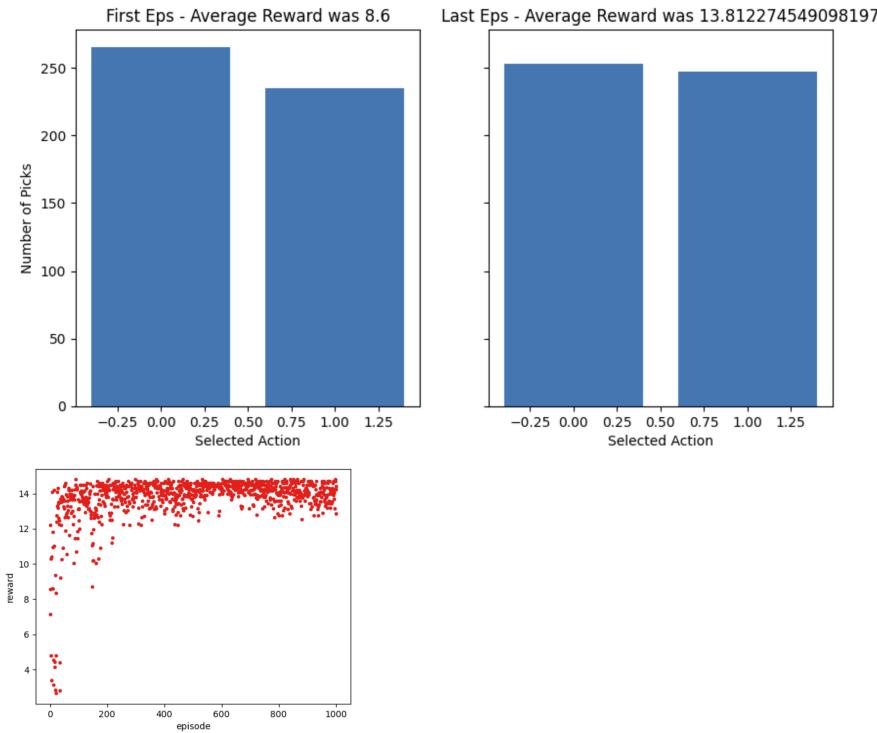
## Gradual



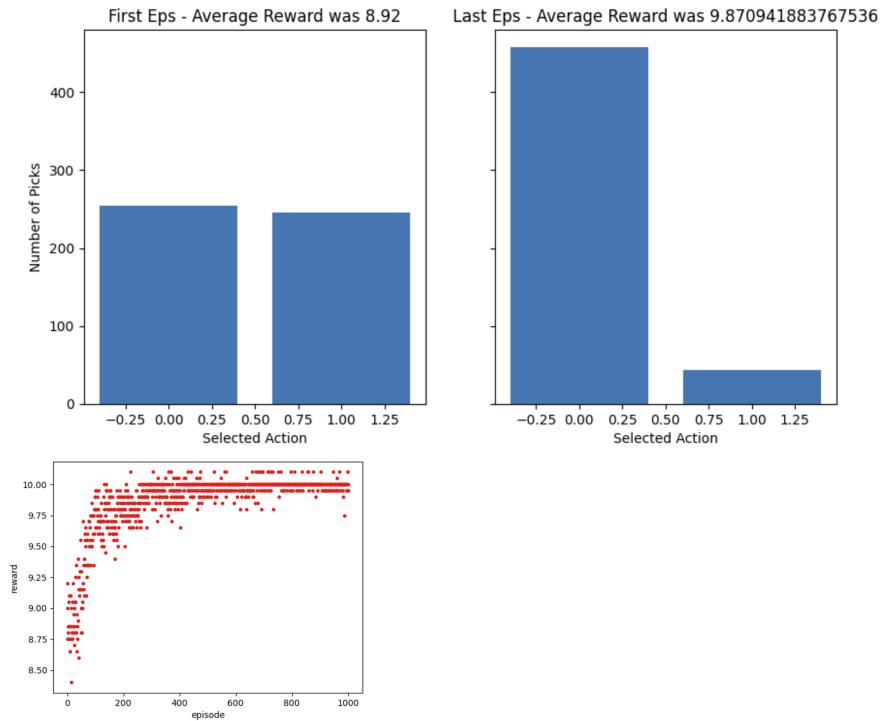
## Soft Majority



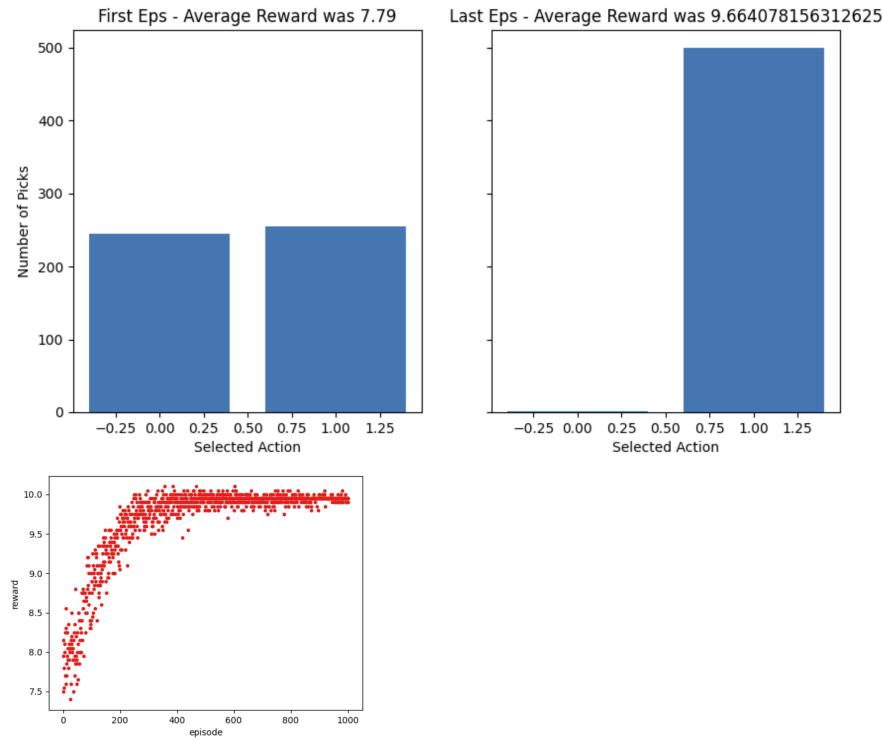
## Hard Majority



## Remorseful Prober



## Soft Grudger



## Prober

