Fictitious play & Reinforcement Learning for computing Nash Equilibria

Alexandros Filios | MTN2219
Nikolaos Chiotis | MTN2221

Feb '23

Agenda

1. Theoretical Framework
2. Technical Implementation
3. Experimental Results
4. Questions

1. Theoretical Framework

1. Theoretical Framework | Zero-Sum Games

Zero-sum games refer to a type of game where the total amount of reward or payoffs in each round is constant, and the sum of all rewards across all players is **equal to zero.**

In zero sum games, for **every win** of one player, there must be an **equal loss** for the other player.

These games are used to model situations where **there is a conflict of interests between two or more agents.**

1. Theoretical Framework | Nash Equilibrium

Nash Equilibrium is a concept in game theory that refers to a **stable state** where each player's **strategy is optimal** given the strategies of the other players.

In a Nash Equilibrium, **no player has an incentive to change their strategy** as they are already receiving the high.

Nash Equilibrium is applicable in **both one-shot and repeated games** and can be found through mathematical methods **such as linear programming, minimax algorithm, and others.**

1. Theoretical Framework | Fictitious Play

Fictitious Play is an iterative algorithm used to **find Nash Equilibrium** in repeated games.

In each iteration, players update their strategy based on the **observed strategies** of the other players.

Fictitious Play has been **proven to converge to a Nash Equilibrium**, assuming the other players' strategies are stationary.

1. Theoretical Framework | Reinforcement Learning

Reinforcement Learning (RL): a type of machine learning that involves training an agent to make decisions **based on rewards** received in an environment.

Q-Learning: a popular RL algorithm that uses a **Q-table** to keep track of the **expected rewards** of taking each action in each state of the environment.

Q-Learning updates its Q-table using the Bellman equation, which expresses the expected future reward of taking an action in a state as the **sum of the immediate reward and the maximum expected future reward.**

2. Technical Implementation

2. Technical Implementation | Project Structure

class

FPZeroSumPlayer

Implements an **agent** that plays a **zero-sum game** by applying **Fictitious Play** algorithm

class

RLZeroSumPlayer

Implements an **agent** that plays a **zero-sum game** by applying **minimax-Q Learning** algorithm

functions

play_fp_game play_rl_game play_fp_rl_game

Helper functions that **simulate** a zero-sum repeated **game** with **Nash Equilibrium** stopping criteria

2. Technical Implementation | Project Structure

class

FPZeroSumPlayer

Implements an **agent** that plays a **zero-sum game** by applying **Fictitious Play** algorithm

class

RLZeroSumPlayer

Implements an **agent** that plays a **zero-sum game** by applying **minmax Q-Learning** algorithm

functions

play_fp_game play_rl_game play_fp_rl_game

Helper functions that **simulate** a zero-sum repeated **game** with **Nash Equilibrium** stopping criteria

Outcomes

- ✓ Easier & Quicker **Experiments**
- ✓ **Stop the game** when has reached Nash Equilibrium
- ✓ **Performance Metrics** for compare two algorithms

2. Technical Implementation | Project Structure

class
FPZeroSumPlayer

class
RLZeroSumPlayer

functions
play_fp_game
play_rl_game
play_fp_rl_game

2. Technical Implementation | Fictitious Play

class

RLZeroSumPlayer

functions

play_fp_game

play_rl_game

play_fp_rl_game

class

FPZeroSumPlayer

Init()

- Initializes **W** based on user input or by random weights
- Initializes **P** based on W to calculate opponent's actions probabilities
- Updates agent's **Policy**

take_action()

- **Calculates belief** based on opponent's actions probabilities
- **Selects action** based on the best response award
- Returns **action**

learn()

- Takes as argument **opponent's action** for the round
- **Updates W**, by increasing opponent's action count by 1
- **Calculates P**, based on new W
- Updates agent's **Policy**

2. Technical Implementation | Reinforcement Learning

class

FPZeroSumPlayer

functions

play_fp_game

play_rl_game

play_fp_rl_game

class

RLZeroSumPlayer

Init()

- Arguments:
  alpha, gamma, explor, actions, state_init
- Initializes **Q** for each action – opponent's action pair with 1
- Initializes **Pi** uniformly
- Initializes **V** with 1

take_action()

- **Selects action**:
  With probability *explor* selects a **random action**.
  With probability (1-explor) selects the **best response.**
- Returns **action**

learn()

- Arguments:
  reward, opponent's action, state
- Calls **Update_Q**, that first updated alpha and then calculates new Q
- Calls **Update_Pi**, that by using linear programming updates Pi
- Calls **Update_V**

2. Technical Implementation | Games Set-up

class

FPZeroSumPlayer

class

RLZeroSumPlayer

functions

play_fp_game play_rl_game play_fp_rl_game

Arguments

- ***max_iterations***
- ***policy_delta_thres*** Maximum policy threshold, to conclude the the policy is stable. Stable
- **policy_delta_rounds_thres** Number of rounds to conclude that we reached a stable policy

Process

- **Iterates rounds** based on user input
- In each iteration two players **take_action()** and then **learn()**
- Keeps a **game history**
- Checks if a **Nash Equilibrium reached**

Outputs

- Displays **Nash Equilibrium**, if it was reached
- Displays number of **rounds needed & time elapsed** until Nash Equilibrium
- Displays plots of the policy change between rounds.

3. Experimental Results

3. Experimental Results | Matching Pennies

Matching Pennies has **2 actions** for each player, with the below pay-off matrix:

Matching Pennies	**Heads**	**Tails**
Heads	1,-1	-1,1
Tails	-1,1	1,-1

Nash Equilibrium criteria: stable policy for 50 continues rounds, with a 1% max change of policy during these rounds.

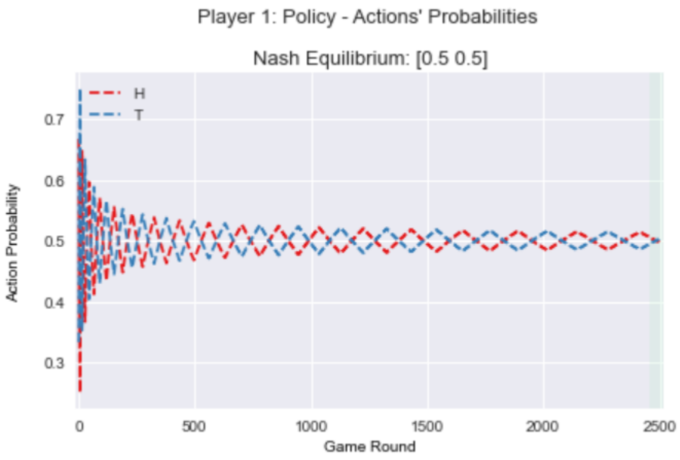
3. Experimental Results | Matching Pennies

Results with Fictitious Play

```
Congrats! A Nash Equilibrium was reached! 🎉

Player 1 policy: [0.5 0.5]
Player 2 policy: [0.5 0.5]

Time elapsed: 231.41s
Number of rounds elapsed: 2506
```

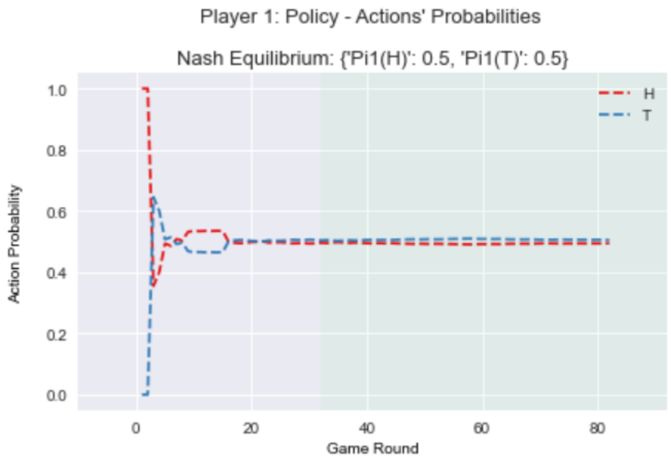


Results with Reinforcement Learning

```
Congrats! A Nash Equilibrium was reached! 🎉

Player 1 policy: {'Pi1(H)': 0.5, 'Pi1(T)': 0.5}
Player 2 policy: {'Pi1(H)': 0.5, 'Pi1(T)': 0.5}

Time elapsed: 16.99s
Number of rounds elapsed: 82
```



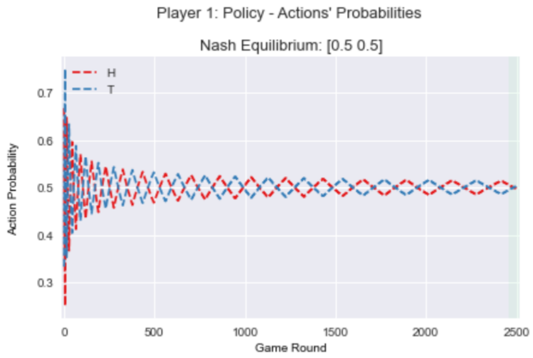
3. Experimental Results | Matching Pennies

Results with Fictitious Play

```
Congrats! A Nash Equilibrium was reached! 🎉

Player 1 policy: [0.5 0.5]
Player 2 policy: [0.5 0.5]

Time elapsed: 231.41s
Number of rounds elapsed: 2506
```

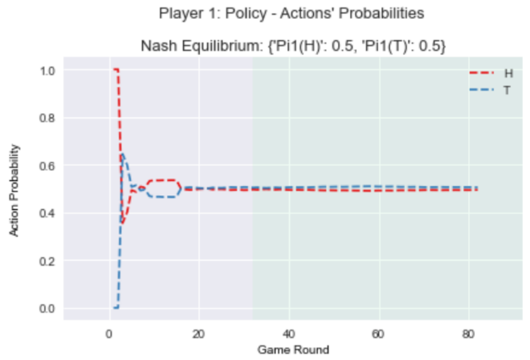


Results with Reinforcement Learning

```
Congrats! A Nash Equilibrium was reached! 🎉

Player 1 policy: {'Pi1(H)': 0.5, 'Pi1(T)': 0.5}
Player 2 policy: {'Pi1(H)': 0.5, 'Pi1(T)': 0.5}

Time elapsed: 16.99s
Number of rounds elapsed: 82
```



Outcomes

Reinforcement Learning can **reach a Nash Equilibrium quicker**, in 16s compared to 3.8min.

Reinforcement Learning Policies seem to merge to equilibrium much earlier than Fictitious Play.

3. Experimental Results | Rock-Paper-Scissors

Matching Pennies has **3 actions** for each player, with the below pay-off matrix:

Rock Paper Scissors	Rock	Paper	Scissors
Rock	0	-1,1	1,-1
Paper	1,-1	0	-1,1
Scissors	-1,1	1,-1	0

Nash Equilibrium criteria: stable policy for 50 continues rounds, with a 1% max change of policy during these rounds.

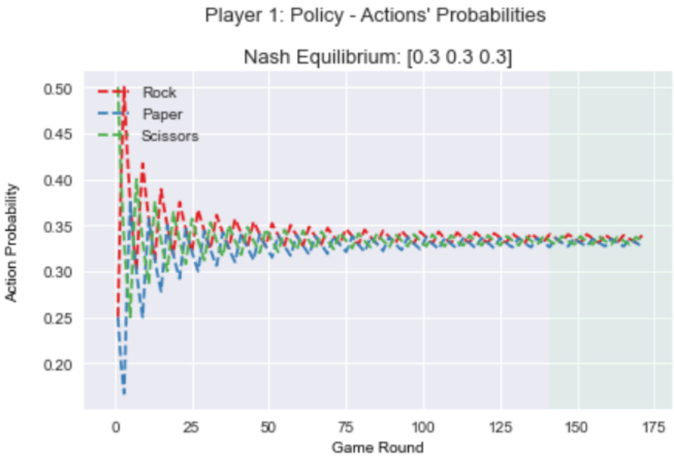
3. Experimental Results | Rock-Paper-Scissors

Results with Fictitious Play

```
Congrats! A Nash Equilibrium was reached! 🎉

Player 1 policy: [0.3 0.3 0.3]
Player 2 policy: [0.3 0.3 0.3]

Time elapsed: 9.46s
Number of rounds elapsed: 171
```

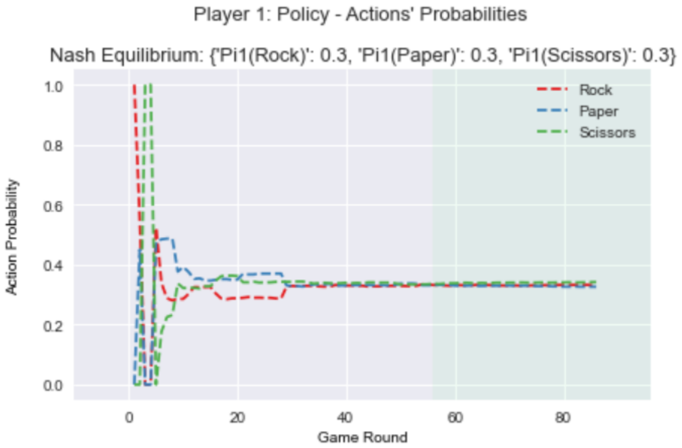


Results with Reinforcement Learning

```
Congrats! A Nash Equilibrium was reached! 🎉

Player 1 policy: {'Pi1(Rock)': 0.3, 'Pi1(Paper)': 0.3, 'Pi1
Player 2 policy: {'Pi1(Rock)': 0.3, 'Pi1(Paper)': 0.3, 'Pi1

Time elapsed: 17.07s
Number of rounds elapsed: 86
```



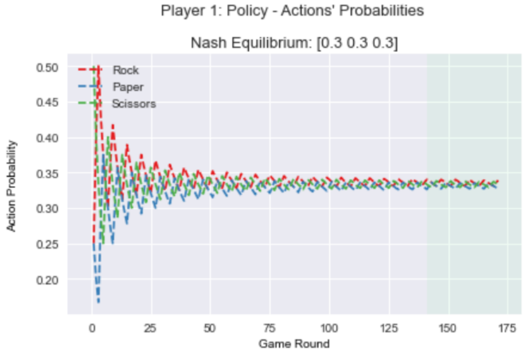
3. Experimental Results | Rock-Paper-Scissors

Results with Fictitious Play

```
Congrats! A Nash Equilibrium was reached! 🎉

Player 1 policy: [0.3 0.3 0.3]
Player 2 policy: [0.3 0.3 0.3]

Time elapsed: 9.46s
Number of rounds elapsed: 171
```

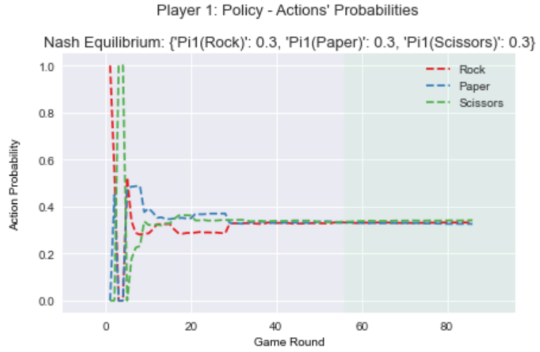


Results with Reinforcement Learning

```
Congrats! A Nash Equilibrium was reached! 🎉

Player 1 policy: {'Pi1(Rock)': 0.3, 'Pi1(Paper)': 0.3, 'Pi1
Player 2 policy: {'Pi1(Rock)': 0.3, 'Pi1(Paper)': 0.3, 'Pi1

Time elapsed: 17.07s
Number of rounds elapsed: 86
```



Outcomes

RL can reach a Nash Equilibrium in **less rounds**, however each round seems to be more time consuming

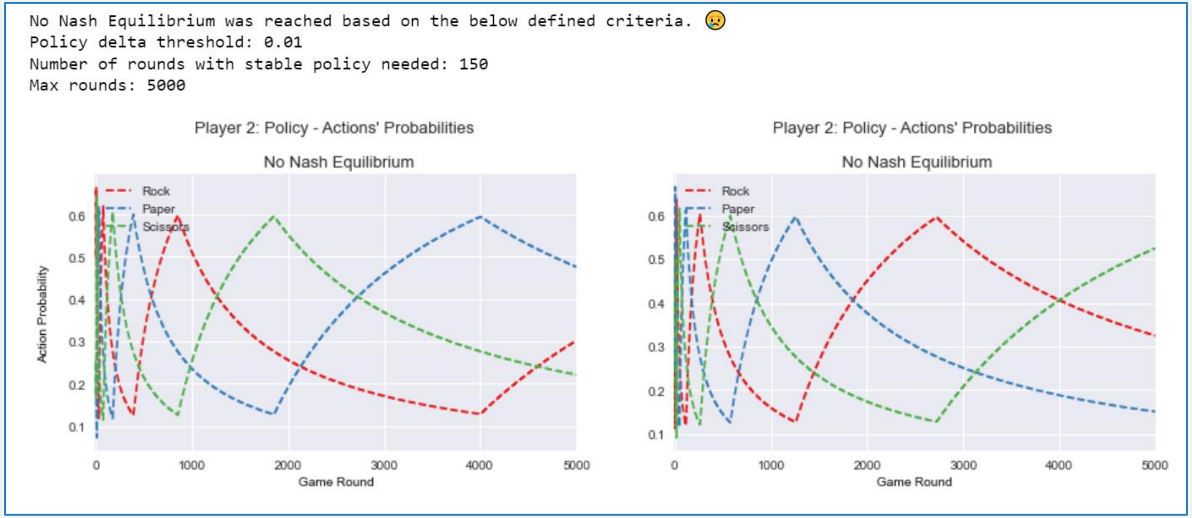
FP needs less time per round, due to its simple process.

In cases that both methods can find the Nash Equilibrium in few rounds, **Fictitious Play is faster**.

FP results seems to be **sensitive to initial beliefs**

3. Experimental Results | Rock-Paper-Scissors

Results with Fictitious Play using different initial beliefs



Outcomes

RL can reach a Nash Equilibrium in **less rounds**, however each round seems to be more time consuming

FP needs less time per round, due to its simple process.

In cases that both methods can find the Nash Equilibrium in few rounds, **Fictitious Play is faster**.

FP results seems to be **sensitive to initial beliefs**

3. Experimental Results | Game with Pure Equilibrium

Since the other games, concluded in a **mixed strategy Nash Equilibrium**, we wanted to test a game that was designed to have a **pure Nash Equilibrium.**
It has 3 actions for each players, with the below pay-off matrix:

Pure Equilibrium Game	Action 1	Action 2	Action 3
Action 1	2,-2	0,0	1,-1
Action 2	-4,4	-3,3	2,-2
Action 3	1,-1	-2,2	-2,2

Nash Equilibrium criteria: stable policy for 50 continues rounds, with a 1% max change of policy during these rounds.

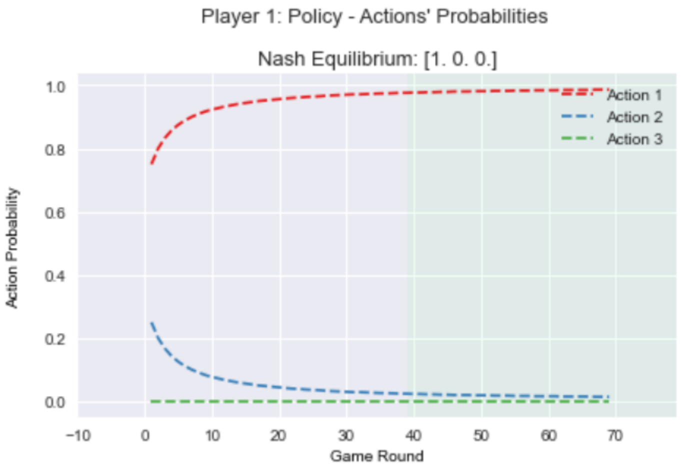
3. Experimental Results | Game with Pure Equilibrium

Results with Fictitious Play

```
Congrats! A Nash Equilibrium was reached! 🎉

Player 1 policy: [1. 0. 0.]
Player 2 policy: [0. 1. 0.]

Time elapsed: 4.21s
Number of rounds elapsed: 69
```

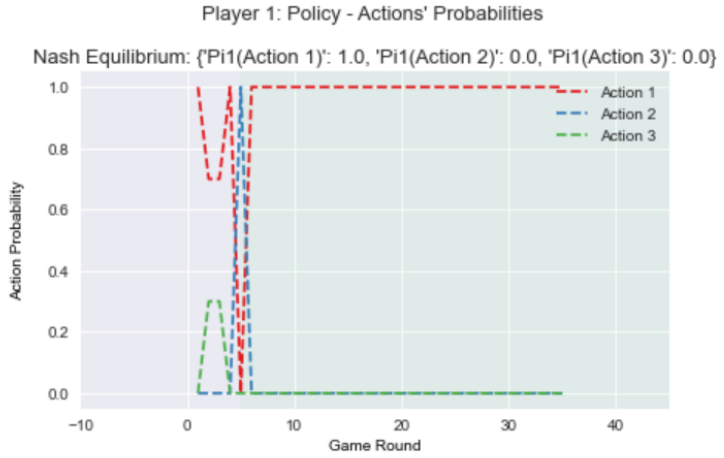


Results with Reinforcement Learning

```
Congrats! A Nash Equilibrium was reached! 🎉

Player 1 policy: {'Pi1(Action 1)': 1.0, 'Pi1(Action 2)': 0.0,
Player 2 policy: {'Pi1(Action 1)': 0.0, 'Pi1(Action 2)': 1.0,

Time elapsed: 8.14s
Number of rounds elapsed: 35
```



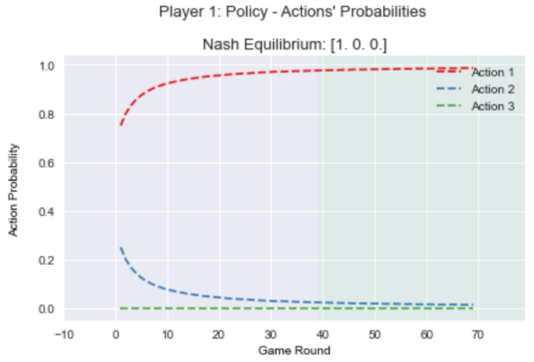
3. Experimental Results | Game with Pure Equilibrium

Results with Fictitious Play

```
Congrats! A Nash Equilibrium was reached! 🎉

Player 1 policy: [1. 0. 0.]
Player 2 policy: [0. 1. 0.]

Time elapsed: 4.21s
Number of rounds elapsed: 69
```

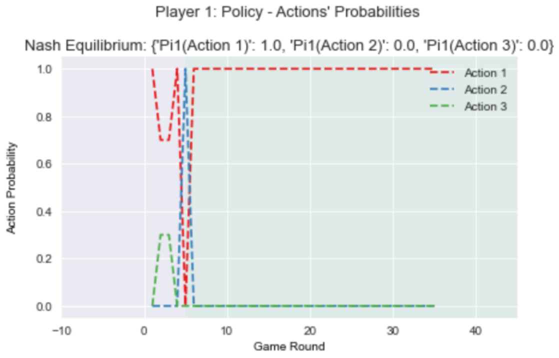


Results with Reinforcement Learning

```
Congrats! A Nash Equilibrium was reached! 🎉

Player 1 policy: {'Pi1(Action 1)': 1.0, 'Pi1(Action 2)': 0.0,
Player 2 policy: {'Pi1(Action 1)': 0.0, 'Pi1(Action 2)': 1.0,

Time elapsed: 8.14s
Number of rounds elapsed: 35
```

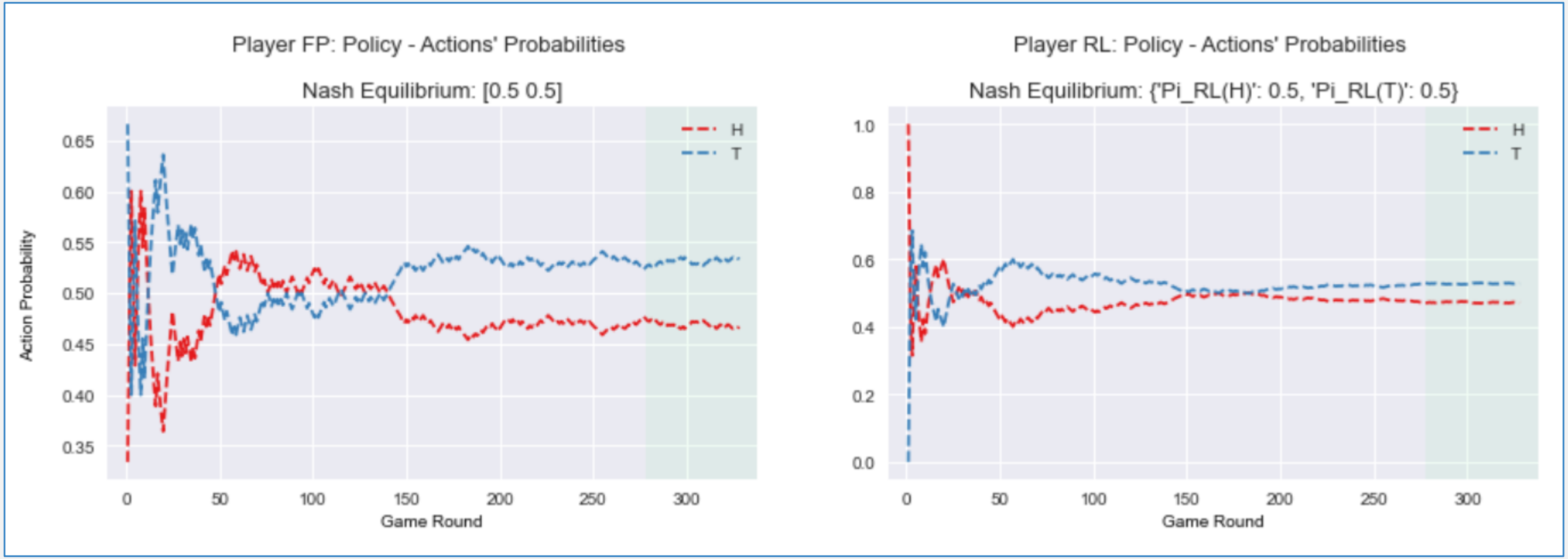


Outcomes

Again, Reinforcement Learning reached a Nash equilibrium in less rounds and with moved earlier close to the equilibrium than Fictitious Play.

However, due to the computation time caused by the solving of the linear programming at each round, Fictitious Play was faster in terms of time.

3. Experimental Results | FP vs RL



Thank you!

Questions?

3. Experimental Results | FP vs RL