

Multi-Agent RL for the Wolf Sheep Game

Xiao'ao Song
Shahab Nikkhoo

Jun. 8, 2023

Outline

- Game introduction
 - Environment
 - Rules and Objectives
 - Agents
- Methodologies
 - Search algorithm
 - Supervised learning algorithm
 - Training data
 - Feature transformation
 - Reinforcement learning (RL) algorithm
 - Teacher-student framework
 - PPO
- Final demonstration
- Summary
- Contribution and future work

Game introduction - Environment

- Environment: fully observable, dynamic, multi-agent
- Map size: 15 x 19
- Maps elements:
 - Empty "." Grass "g" Rhubarb "r" Fence "#"
- 4 Agents:
 - Player 1: Sheep (S), Wolf(W)
 - Player 2: sheep (s), wolf(w)
- Action space: {up, down, left, right, none}

```
.....#.....
.....#.....
...g...#...w...
.....##..#..##..
.....#..#..#..
.....#..s....
.....#
.....#####
.....#.
.....#.w...
...g#...#...#...
...g##..#..##..
...ggg..#.S...
.....#
.....#.
```

- Map in matrix form



- Map in RGB image form

Game introduction - Rules and Objectives

- The Wolf moves for every second step of the sheep

Invalid move:

- Move outside of the map or on the fence
- Move on the grid where it is occupied

Valid move:

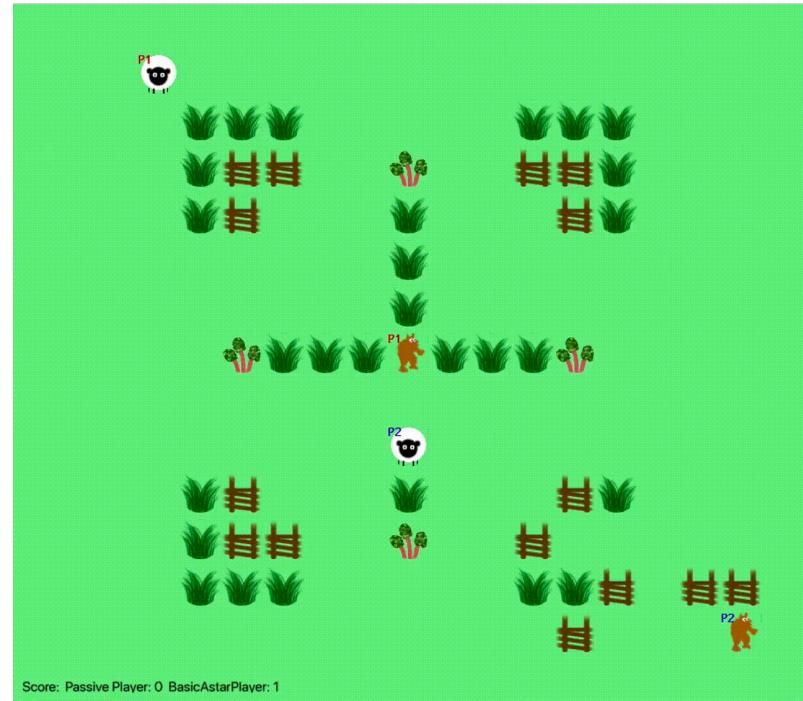
- Consume or sabotage the food
- Move to the empty grid
- Wolf can eat enemy sheep by moving to its grid

Sheep Objectives:

- Avoid the enemy wolf
- Collect as many food as possible

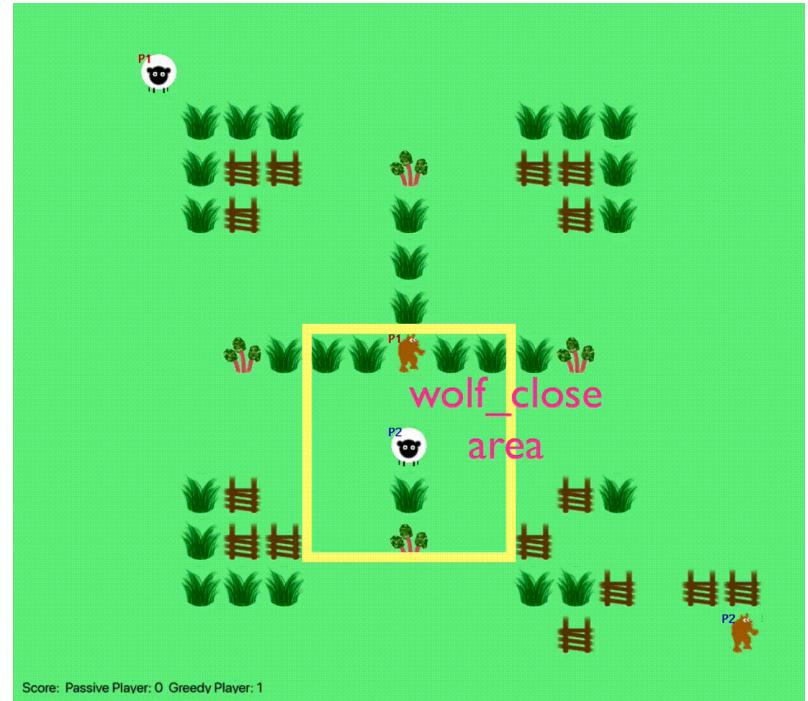
Wolf Objectives:

- Catch the enemy sheep
- Block the enemy wolf
- Sabotage the food



Game introduction – Default agents

- Random player: moves randomly every turn
- Keyboard player: moves by receiving your keyboard command
- Passive player: stays in place every turn
- Greedy player: uses a simply greedy approach. It includes the following helper functions:
 - `get_player_position()`
 - `food_present()`
 - `valid_move()`
 - `closest_goal()`
 - `wolf_close()`
 - `run_from_wolf()`
 - `gather_closest_goal()`
 - `move_wolf()`
 - `move_sheep()`



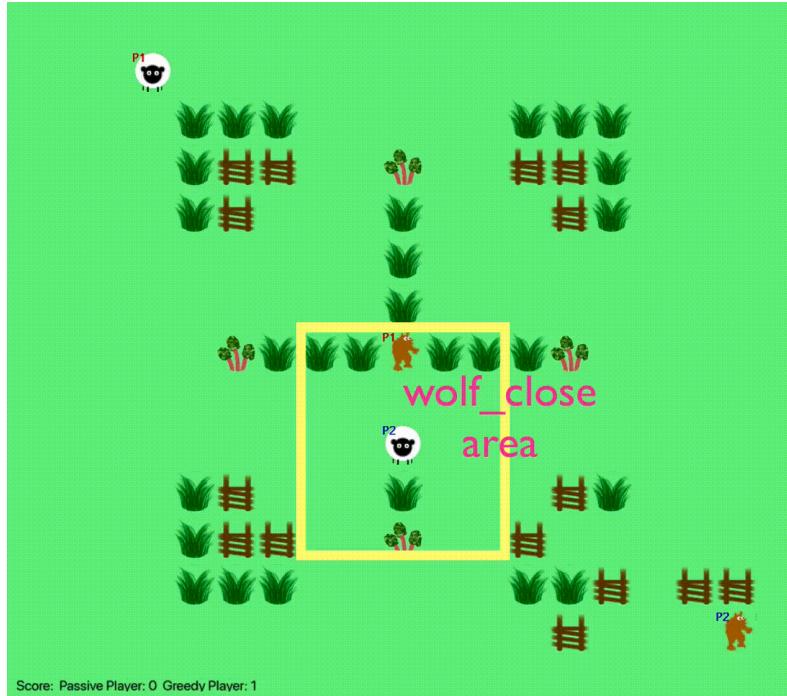
- Drawback of Greedy player



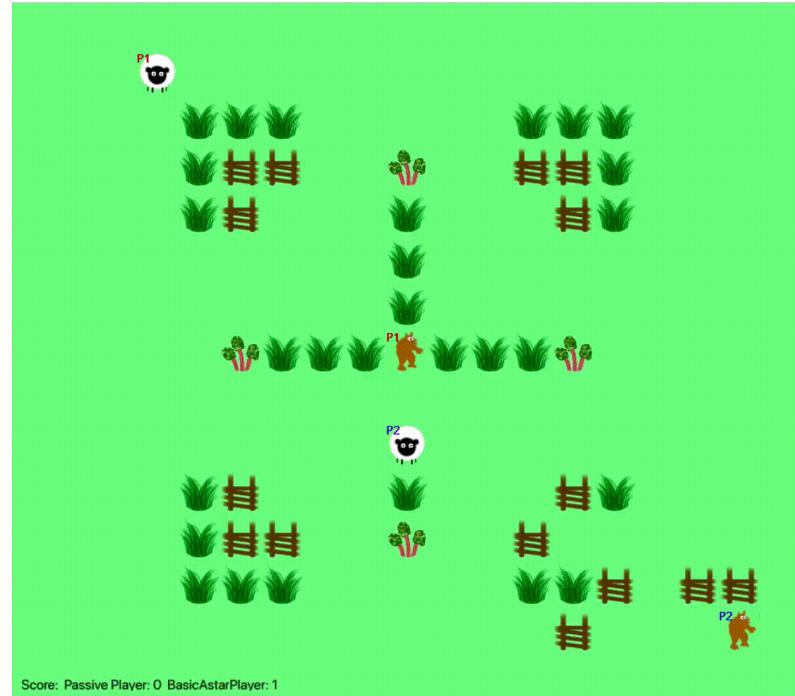
Methodologies – Search-based algorithm: A*



Methodologies – Search-based algorithm: A*



- Greedy player



- A* player

Methodologies – Supervised learning algorithm: MLP

- Sample training data

	field_before	field_after	turn_made_by	move_made	score1	score2	reason
0	['g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', 'g', ...]	['g', 'g', 'g', 'S', 'g', 'g', 'g', 'g', 'g', ...]	player1 sheep	-1	1	0	max_iterations
1	['g', 'g', 'g', 'S', 'g', 'g', 'g', 'g', 'g', ...]	['g', 'g', 'g', 'S', 'g', 'g', 'g', 'g', 'g', ...]	player2 sheep	-1	1	1	max_iterations
2	['g', 'g', 'g', 'S', 'g', 'g', 'g', 'g', 'g', ...]	['g', 'g', 'S', ' ', 'g', 'g', 'g', 'g', 'g', ...]	player1 sheep	-2	2	1	max_iterations
3	['g', 'g', 'S', ' ', 'g', 'g', 'g', 'g', 'g', ...]	['g', 'g', 'S', ' ', 'g', 'g', 'g', 'g', 'g', ...]	player2 sheep	-1	2	2	max_iterations
4	['g', 'g', 'S', ' ', 'g', 'g', 'g', 'g', 'g', ...]	['g', 'g', 'S', ' ', 'g', 'g', 'g', 'g', 'g', ...]	player1 wolf	2	2	2	max_iterations

- In the context of supervised learning, we just need the following data to form (X , Y) pairs:
 - field_before**: map before the move was made.
 - move_made**: the actual move the agent made (e.g. -1).

Methodologies – Supervised learning algorithm: MLP

- Due to limited computing resource, we preprocess the data to reduce the dimensionality of the data from high to low.

```
.....#.....  
.....#.....  
....g..#...W..  
....##..#..##..  
....#..#..#..  
....#..s..  
.....#.....  
  
.....#.....  
.....#.w..  
....g#..#..#..  
....g##..#..##..  
....ggg..#..S..  
.....#.....  
.....#.....
```

$\Phi(\text{input map}) \rightarrow \text{vector}$

Sheep features: dim: $15 \times 19 \rightarrow 1 \times 16$

[sheep_fencing, sheep_stuckable, stay_preferable, food_available, path_available, move_up_closer, move_down_closer, move_left_closer, move_right_closer, target_neighboring, up_walkable, down_walkable, left_walkable, right_walkable, stay_safeable, run_fromwolf]

Wolf features: dim: $15 \times 19 \rightarrow 1 \times 11$

[sheep_stuckable, stay_preferable, path_available, move_up_closer, move_down_closer, move_left_closer, move_right_closer, up_walkable, down_walkable, left_walkable, right_walkable]

- All of the above features only have value 1 or 0, e.g.:

- Sample **Sheep** feature vector: [0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1] and it takes action: UP
- Sample **Wolf** feature vector: [0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1] and it takes action: NONE

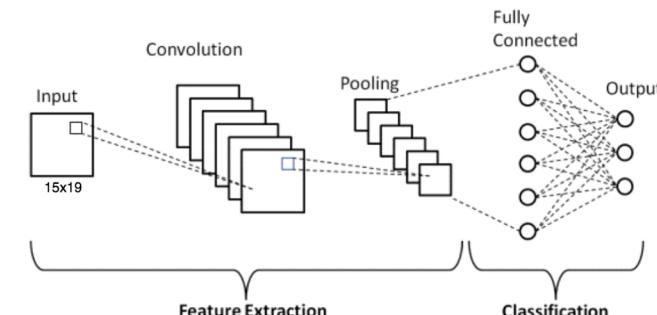
Methodologies – Supervised learning algorithm: MLP

- Now, we can learn a simple network based on many ($\Phi(X)$, Y) pairs:

- We have our policy networks now!

```
sheep_model = MLPClassifier(solver='lbfgs',
                            activation='relu',
                            alpha=1e-4,
                            hidden_layer_sizes=(16,16),
                            random_state=1,
                            max_iter=200,
                            tol = 0.0001 )
sheep_model.fit(X_sheep,Y_sheep)
```

- A few notes on preprocessing:
 - We hand-craft features to reduce dimensionality, then train a simple MLP
 - However, they are totally learnable (e.g. CNN)
 - If no budget limit, one can use end-to-end learning without any preprocessing (a comparison is provided at the end).



Methodologies – RL: PPO

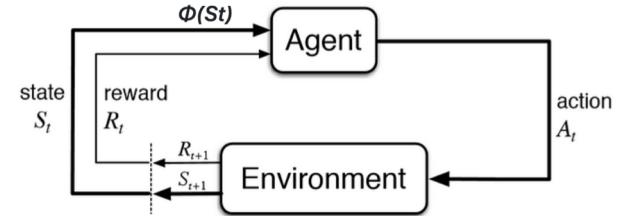
- In contrast to supervised learning, RL has no labels but learns from the rewards.
- Proximal Policy Optimization (PPO) :
 - On-policy optimization algorithms.
 - Balance between sample efficiency and stability.
 - Robust to hyperparameter choices and often converges reliably.
 - Employs a surrogate objective function with a clipping mechanism to ensure stable updates

Algorithm 1 PPO, Actor-Critic Style

```

for iteration=1,2,... do
  for actor=1,2,...,N do
    Run policy  $\pi_{\theta_{\text{old}}}$  in environment for  $T$  timesteps
    Compute advantage estimates  $\hat{A}_1, \dots, \hat{A}_T$ 
  end for
  Optimize surrogate  $L$  wrt  $\theta$ , with  $K$  epochs and minibatch size  $M \leq NT$ 
   $\theta_{\text{old}} \leftarrow \theta$ 
end for

```



Methodologies – RL: PPO feature vector learning

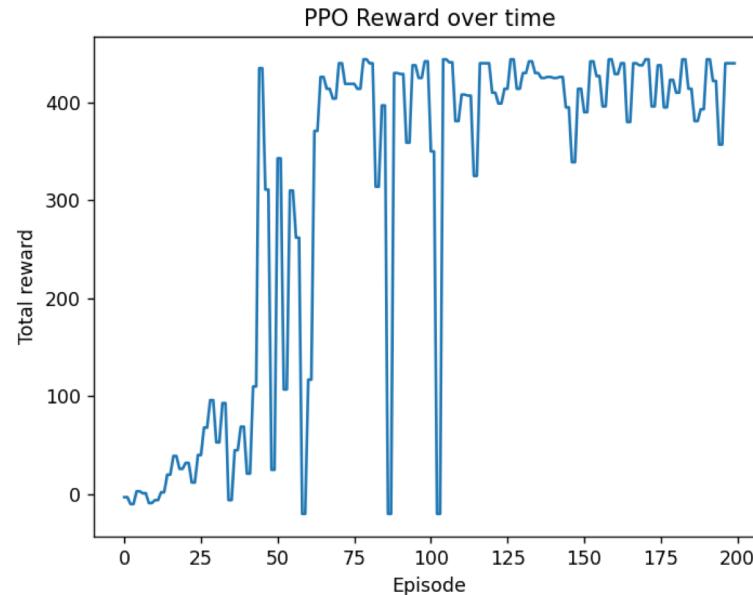
Design the rewards (for Sheep)

- Collect grass/rhubarb : +1 / +5; Trapping enemy sheep: +100; Each step: -0.2; Got eaten: -100

```

0   .....
1   .W..S.....
2   ....ggg.....ggg....
3   ....g##..r..##g....
4   ....g#...g...#g....
5   .....g.....
6   .....g.....
7   .....rggggggggr...
8   .....g.....
9   .....g.....
10  ....g#...g...#g....
11  ....g##..r..##g....
12  ....ggg.....ggg....
13  .....s.w....
14  .....
```

0123456789012345678



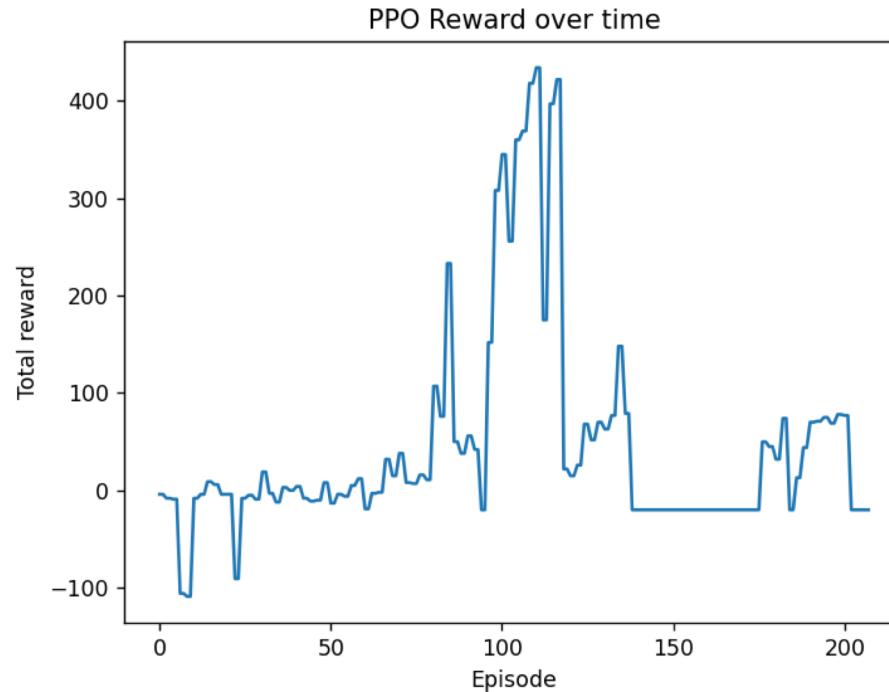
- Input is 1 x 16 vector, output is a scalar

- The learning process is fast and stable

Methodologies – RL: PPO end-to-end learning

Design the rewards (for Sheep)

- Collect grass/rhubarb : +1 / +5; Trapping enemy sheep: +100; Each step: -0.2; Got eaten: -100
- Input map is 15 x 19 matrix
output action is a scalar
- The learning process is unstable
and needs significant time



Methodologies – RL: teacher - student framework

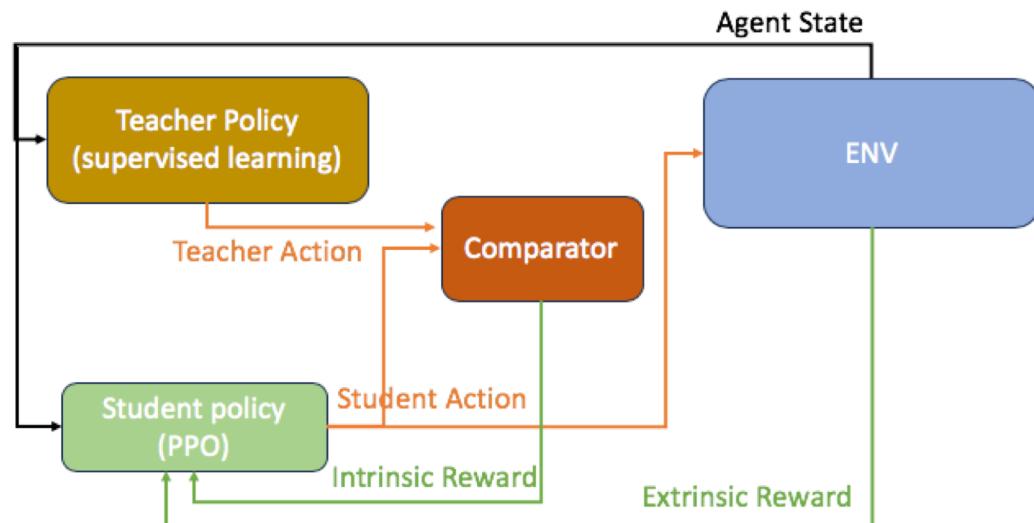
- In stead of doing everything from scratch, we can utilize the MLP model we get previously to guide our agent to learn. This is called teacher-student framework.
- Re-design the rewards (for Sheep)

Intrinsic:

- Student a == Teacher a : 0
- Student a != Teacher a : -3

Extrinsic:

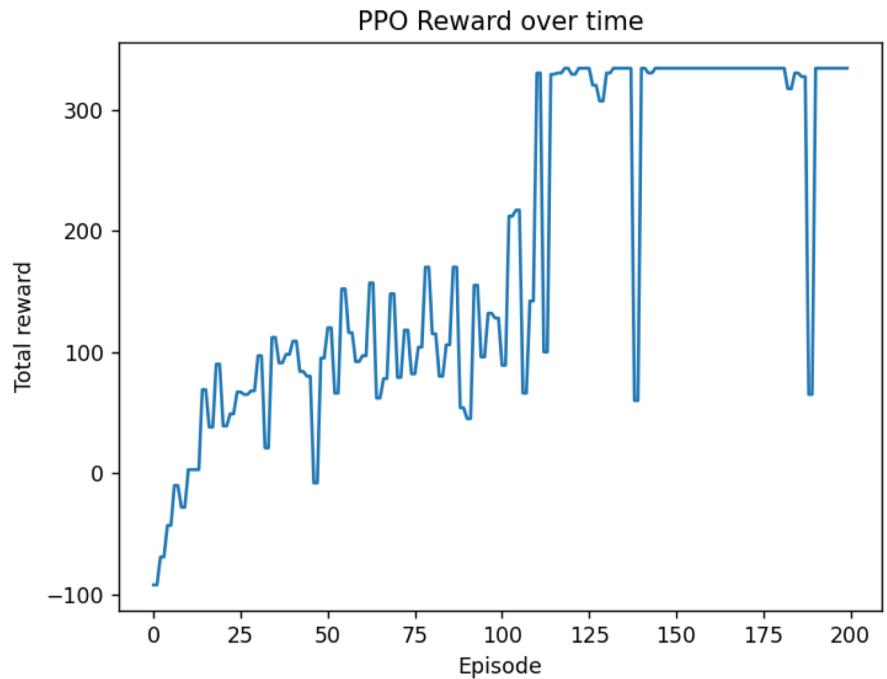
- Collect grass/rhubarb : +1 / +5;
- Trapping enemy sheep: +100;
- Each step: -0.2;
- Got eaten: -100



- Teacher student framework diagram

Methodologies – RL: teacher - student framework

- Teacher Policy
 - has access to privileged data
 - informed trajectory based on A*
- Student Policy
 - has access to map features
 - doesn't have the best trajectory, only the closeness measure
- Time efficient: fast learning process guided by teacher policy



Methodologies – RL: teacher - student framework

- Ideal student policy will replicate the teacher policy

```
0 .....  
1 .W..S.....  
2 .....  
3 .....##..r..##.....  
4 .....#...g...#....  
5 .....g.....  
6 .....g.....  
7 .....g.....  
8 .....g.....  
9 .....g.....  
10 .....#.....#....  
11 .....##.....##....  
12 .....  
13 .....S.W.  
14 .....
```

- Teacher policy

0
1	.W..S.....
2
3##..r..##.....
4#...g...#....
5g.....
6g.....
7g.....
8g.....
9g.....
10#.....#....
11##.....##....
12
13S.W.
14

- Student policy

Final demonstration

Map 1

- Full of food

Opponent agent:

- Greedy player

Our agent:

- RL player

Order:

- Second move



Final demonstration

Map 2

- Fence in the middle

Opponent agent:

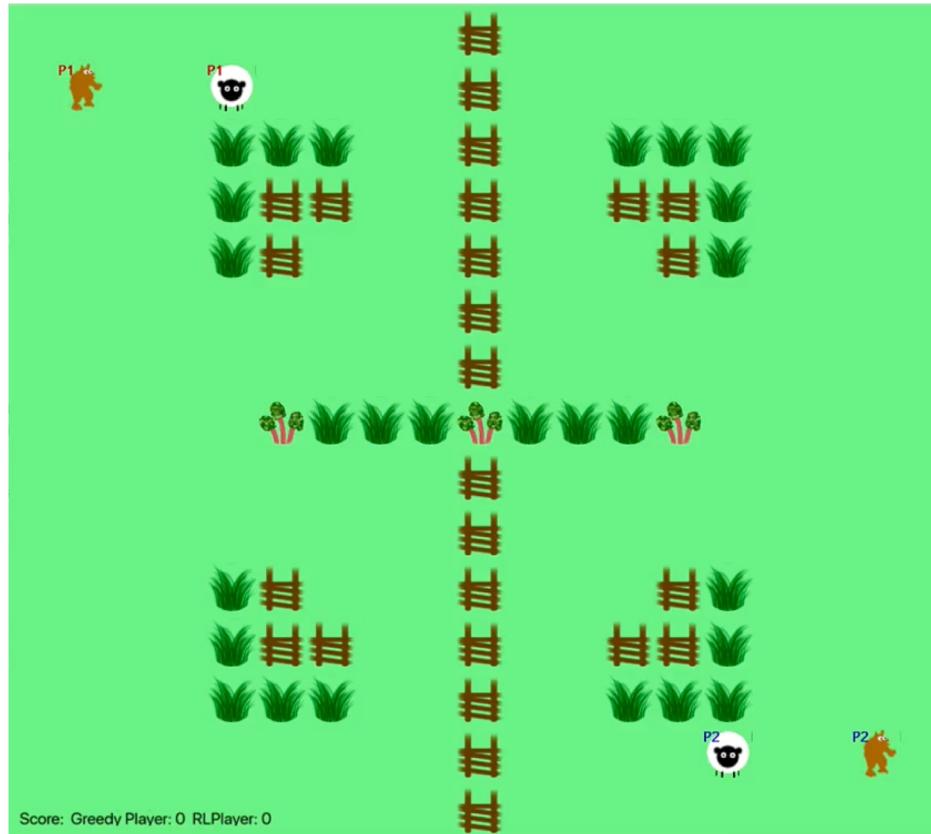
- Greedy player

Our agent:

- RL player

Order:

- Second move



Summary

Search-based

- **Pros:** rule-based, flexible
- **Cons:** needs hand craft

```

2736     badSpot_1, badSpot_2 = alterI
2737
2738     if badSpot_1 < badSpot_2:
2739         post_move = _move1
2740     elif badSpot_1 > badSpot_2:
2741         post_move = _move2
2742     else:
2743         print("the two ways are :")
2744         post_move = _move1
2745
2746     if post_move == 'up':
2747         suggestedAction = Node(Pi)
2748     elif post_move == 'down':
2749         suggestedAction = Node(Pd)
2750     elif post_move == 'left':
2751         suggestedAction = Node(Pl)
2752     elif post_move == 'right':
2753         suggestedAction = Node(Pr)
2754     else:
2755         suggestedAction = Node(Pi)
2756
2757     return suggestedAction

```

Nearly 3000 lines of code!

Learning-based

- **Pros:** no hand crafting
- **Cons:** needs data

```

my_map_p1_v2_gready_scraping_results.csv',
map2_p1_v2_jcd_scraping_results.csv',
test_p1_v2_gready_scraping_results.csv',
map2_p1_v2_gready_scraping_results.csv',
test3_p2_v2_gready_scraping_results.csv',
test_p2_v2_jcd_scraping_results.csv',
test3_p2_v2_jcd_scraping_results.csv',
map3_p1_v2_jcd_scraping_results.csv',
map1_p1_v2_gready_scraping_results.csv',
map3_p1_v2_gready_scraping_results.csv',
my_map_2_p2_v2_gready_scraping_results.csv',
my_map_p1_v2_jcd_scraping_results.csv',
map1_p2_v2_gready_scraping_results.csv',
my_map_2_p1_v2_gready_scraping_results.csv',
map3_p2_v2_gready_scraping_results.csv',
map3_p2_v2_jcd_scraping_results.csv',
test_p2_v2_gready_scraping_results.csv',
map1_p1_v2_jcd_scraping_results.csv',
my_map_p2_v2_gready_scraping_results.csv',
test3_p1_v2_gready_scraping_results.csv',
map2_p2_v2_gready_scraping_results.csv']

```

At least 3~4k data samples!

RL-based

- **Pros:** no need to collect data
- **Cons:** needs huge time to learn

At least 3h per training



FREELUNCH

Contribution and future work

- Wrote the **gym** environment for the Wolf-sheep game.
- Developed **Search**-based, Supervised **learning**-based and **RL**-based agents.
- Tested our agents in multiple **unseen maps**.

In future work:

- Learn end-to-end model to process high-dimensional data directly.
- Learn one model for multi-agents (depends on the situation).
- Test on more maps that contain some corner cases.