Multi-Agent Reinforcement Learning for the Wolf-Sheep Game

Xiao’ao Song Shahab Nikkhoo
Department of Electrical and Computer Engineering
UC Riverside
{xiaoao.song, shahab.nikkhoo}@email.ucr.edu

***Abstract*—This project seeks to delve into multi-agent reinforcement learning (MARL) algorithms through their application to the Wolf-Sheep game, represented in Fig. 1. This game involves two players, with each player controlling a pair of agents - a wolf and a sheep. The game concludes when either a player’s wolf captures the sheep of the opposing team or the maximum number of steps is exceeded. In either scenario, the team accruing the highest collective rewards emerges victorious. The specifics of the game environment will be further elaborated in Section II. Another key aspect of this MARL game involves enabling agents from the same team to collaborate for a better team outcome. This concept, inspired by the work of Bowen Baker et al. [1], prompts us to further investigate whether agents within the same team can effectively cooperate to dominate the game.**

Our objectives include comparing and evaluating the performance of various algorithms, spanning from search based algorithm, supervised learning algorithm, to reinforcement learning. For reinforcement learning, we adopt three distinct approaches: end-to-end learning, feature extracted learning, and the teacher-student framework. We then put these algorithms against each other to assess their learning times and costs. An additional goal is to evaluate the generalization capability of our trained models across multiple game maps.

The project’s code is accessible at https://github.com/lineojcd/Multi-Agent-RL-for-the-Wolf-Sheep-Game

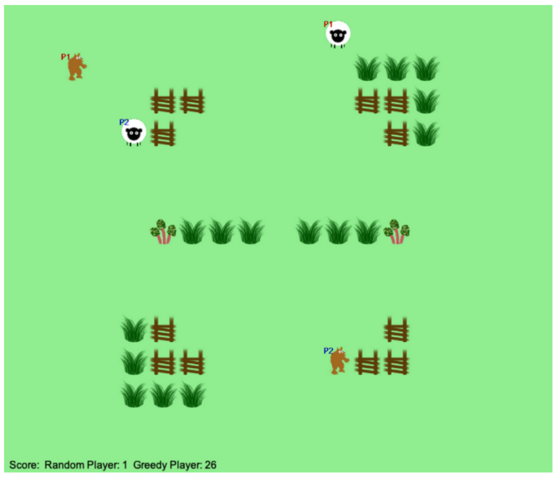


Fig. 1. Wolf-Sheep game GUI

***Index Terms*—Reinforcement Learning, Teacher-student framwork**

I. Introduction

Machine learning and artificial intelligence (AI) stand at the forefront of modern technological discussions, profoundly influencing our capacity to address complex problems. In the myriad of methodologies that researchers employ, Reinforcement Learning (RL) represents a promising and innovative approach that enables us to navigate toward optimal solutions through constant interaction with the environment.

RL has demonstrated exceptional prowess in solving an array of gaming challenges, evidenced by its ability to outperform the finest chess players [5]. Moreover, it has been integral to the significant advancements in large language models [3]. RL’s application adopts several methodologies, each with distinct characteristics and benefits.

One such methodology is end-to-end learning, where a model endeavor to maximize a reward function, learning to perform tasks more efficiently through numerous iterations. However, this process necessitates substantial computational resources and time due to high-dimensional input form the state space.

An alternative approach involves reducing the input dimension for the RL agent. This modification enables the agent to explore the solution space more effectively, increasing the likelihood of maximizing the reward function. However, each method has its strengths and weaknesses, which we seek to explore further in our project.

In this project, our primary objective involves implementing various approaches - the A* search algorithm as a traditional method, supervised learning as a machine learning technique, and different types of reinforcement learning - in the context of multiplayer games. We aim to compare the outcomes of these methodologies to gain insights into their respective advantages and effectiveness.

Moreover, we seek to leverage the teacher-student framework [6], which employs pretrained network from supervised learning algorithm to expedite and enhance the training of RL agents. By examining this methodology, we hope to shed light on its effectiveness and potential as a tool for more efficient and faster RL training.

In the context of this project, our focus will be on the ’Wolf-Sheep’ game. This multiplayer game is structured such that each team has one wolf and one sheep. The primary objective of the game is to maximize each team’s score, achievable either by the sheep consuming grass or the wolf preying on the sheep. As we progress in this report, we will delve deeper into the intricacies of the game and elucidate its various aspects.

II. Problem Description

In this section we dig deeper into the game to explain the environment, rules, objectives, and agents.

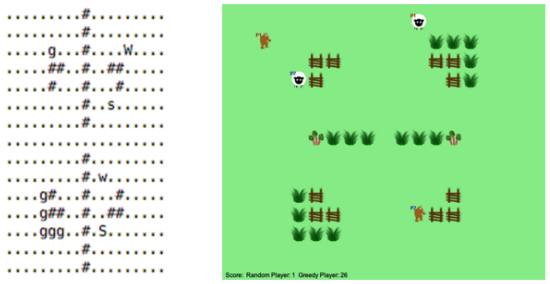
A. Game Description

The Fig. 2 is the graphical user interface(GUI) and matrix form of this Wolf-Sheep game. Each game is played by two players (P1 and P2) that play against each other. A player has to operate two agents: the sheep and the wolf. In the game, we can have different maps: each map should consists of 15x19 squares and each square can contain one of four elements, represented by the following symbols:

- Empty .
- Grass g
- Rhubarb r
- Fence #

The start position of the players are represented by these letters:

- S (Sheep player 1)
- W (Wolf player 1)
- s (Sheep player 2)
- w (Wolf player 2)



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

Fig. 2. Gmae Map in GUI form and matric form

B. Rules and Objectives

Both, the wolf and the sheep have 5 movement options for each call: {up, down, left, right, none}. The wolf is allowed to make a step for every second step of the sheep. In other words the sheep can move twice as fast as the wolf.

- Wolves cannot step on squares occupied by the opponent's wolf (wolves block each other).
- Wolves cannot step on squares occupied by the sheep of the same player.
- Sheep cannot step on squares occupied by the wolf of the same player.
- Sheep cannot step on squares occupied by the opposite sheep.
- Neither the sheep nor the wolf can enter a square with a fence on.
- Neither the sheep nor the wolf can step on a square outside the map. Imagine the map is surrounded by fences.
- If the sheep steps on a food object the food object is consumed (removed from the map) and a score is awarded.
- If the wolf steps on a food object the food object gets removed but no score is awarded.

Each game lasts for a maximum of 100 turns or unless a sheep gets caught by the wolf, then the game ends immediately. If the wolf of player1 catches the sheep of player2 the game ends immediately and player1 wins and is awarded all the points for the current run. This is a fully observable environment meaning that you can have all the information in each step including where the food is and the enemy sheep or wolf is.

The wolf's objective is to sabotage the other player, by removing food items and trying to catch the opposite player's sheep. The sheep's objective is to avoid the wolf of the opposite player and score as many points as possible by eating food items. Collect the rhubarb object gives you five score points and collect the grass object gives you one score point.

C. Default Agents

The Agents are the key of this project as different agents have different capability to win the game. These are the default agents come up with the game:
Random player: A preset agent that moves randomly every turn.
Keyboard player: A preset agent that receives your keyboard command rather than controlled by an algorithm.
Passive player: A preset agent that simply stays in place every turn.
Greedy player: A preset agent that uses a simply greedy approach. It includes the following excerpted functions:

- *get_player_position* gets the current position of the player's or enemy player's sheep or wolf.
- *food_present* tells you if the food is still available in the map.
- *valid_move* checks if the action you are going to take is valid or not.
- *closest_goal* checks if the action you are going to take is valid or not.
- *wolf_close* returns True/False is the enemy wolf is close to your sheep.
- *run_from_wolf* takes action to run from enemy wolf.
- *gather_closest_goal* guides the agent towards its target.

III. Methodology

We propose three methods to solve this problem: Search-based algorithm, Supervised learning-based and RL-based algorithm.

A. Search-based Algorithm

A* search algorithm [2] is a graph traversal and path search algorithm. Through A*, the algorithm will guide the agent to move towards the goal. This agent extends all the functions from Greedy player but replacing the *gather_closest_goal* function by A* path finding algorithm and decreasing the conservative level in *wolf_close* function. In addition, it also adds the following functions:

- *getManhDistance* computes manhattan distance between two spots on the map.
- *huntSheep* takes enemy sheep as the goal and finds out the path for my wolf.
- *getObjbyPosition* takes input position and returns what figure stays on this position.
- *catchableSheep* computes manhattan distance between my wolf position and enemy sheep position, if the distance is 1, my wolf is nearby the enemy sheep and return enemy sheep's position.
- *trapableSheep* computes manhattan distance between my wolf and enemy sheep, my sheep and enemy sheep, my sheep and my wolf, and then returns the wolf action position. Basically, it tells you if my sheep and wolf are trapping the enemy sheep together.
- *stuckableSheep* returns a FLAG that tells you if the enemy sheep is stuck by my wolf at the fence or the boundry of the map. In this case, my sheep might need to come over instantly and help my wolf to trap the enemy sheep together.
- *getFoodsPosition* returns a list of available food and its ABS location and relatively location to my current figure position or another position. The returned list is ordered by relatively distance to me. By default the relative distance is referring w.r.t my sheep. This function also take consideration of the award level of food and its relative distance. For example, if my relative distance towards grass and rhubarb are 1 and 4, the function will select the rhubarb prior to grass.
- *checkSafeFromWolf_by_Manh* returns True value when the Manh Dist from the given spot to enemy wolf is lower than 2 otherwise retuns False value.
- *sheepfencing* is designed for my sheep. Basically, it tells the sheep don't move and work as a fence when my wolf and sheep are trapping the enemy sheep together.

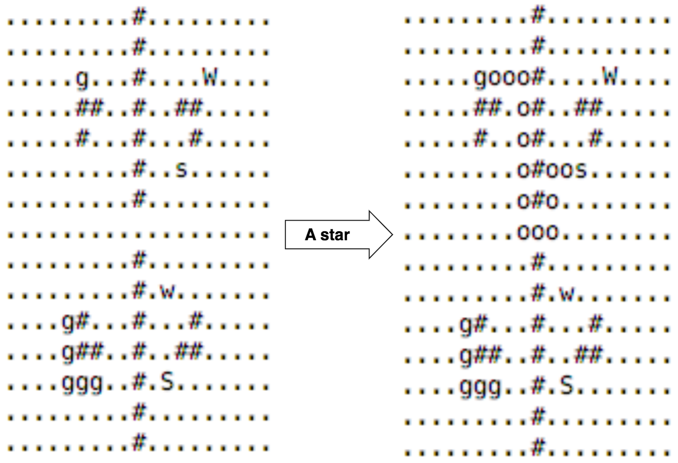
A visualization of A* path finding algorithm is shown in Fig. 3. On the left, the sheep (s) is finding the path towards its closest grass (g). After runing A* algorithm, it has found the shortest pathfrom sheep to grass as shown on the right hand side.

B. Supervised learning Algorithm

Unlike the Search-based algorithm, in this approach, one needs to learn the strategy from the past experience as known as training data.

1) Training Data: The training data will be stored in .csv files. It was gathered from previous game experience. Each file is a game that is run, where the following data is gathered for each move:

- *field_before*: this is the field before the move was made.

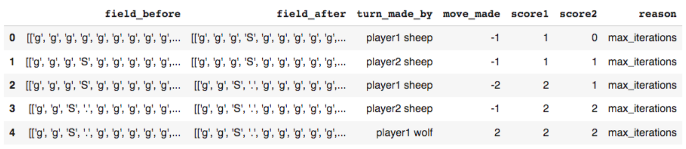

```
.........#.........
.........#.........
.....g...#....W....
.....##..#..##.....
.....#...#...#.....
.........#..s......
.........#.........
...................
.........#.........
.........#.w.......
....g#...#...#.....
....g##..#..##.....
....ggg..#.S.......
.........#.........
.........#.........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

```
.........#.........
.........#.........
.....gooo#....W....
.....##.o#..##.....
.....#..o#...#.....
........o#oos......
........o#o........
........ooo........
.........#.........
.........#.w.......
....g#...#...#.....
....g##..#..##.....
....ggg..#.S.......
.........#.........
.........#.........
```

Fig. 3. A visualization of A star path finding algorithm

- *field_after*: this is the field after the move was made.
- *turn_made_by*: which player made the move (e.g. 'player1 sheep').
- *move_made*: the actual move the agent made (e.g. -1).
- *score1*: the score of player 1 axer the move.
- *score2*: the score of player 2 axer the move.
- *reason*: on the final line of the file, the reason will be given for the termination of the game, if it was not the maximum number of iterations.

Some sample data is shown in Fig. 4:



	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

Fig. 4. Sample training data

In the context of supervised learning, we just need these two fields: **field_before** and **move_made** to form (X, Y) pairs.

2) Feature Extraction: Due to the limited computing power, we preprocess the input data (a 15x19 matrix) into a 1-D vector with the following features as elements below. Compared with the fashion of end-to-end learning, this method sacrifices generalization ability but it is a practical way to deal with our case.

For Sheep:

- *sheepfencing*: Is my sheep blocking the enemy sheep as the fence?
- *stuckableSheep*: Is my sheep stucking the enemy sheep to assit my wolf?
- *stayable*: Is my sheep better staying at current spot?
- *food_present*: Is the food still available in the map?
- *mypos_2_new_tar_has_way*: Is there a way towards the target?
- *mypos_up_2_new_tar_real_dist_smaller*: Is it starting from my up grid to the target better?

- *mypos_down_2_new_tar_real_dist_smaller*: Is it starting from my down grid to the target better?
- *mypos_left_2_new_tar_real_dist_smaller*: Is it starting from my left grid to the target better?
- *mypos_right_2_new_tar_real_dist_smaller*: Is it starting from my right grid to the target better?
- *new_tar_neighboring*: Is the target neighboring?
- *my_pos_up_walkable*: Is my up grid walkable?
- *my_pos_down_walkable*: Is my down grid walkable?
- *my_pos_left_walkable*: Is my left grid walkable?
- *my_pos_right_walkable*: Is my right grid walkable?
- *runfromwolf*: Does my sheep need to run from wolf?

For Wolf:

- *stuckableSheep*: Is my sheep stucking the enemy sheep to assit my wolf?
- *stayable*: Is my sheep better staying at current spot?
- *mypos_2_new_tar_has_way*: Is there a way towards the target?
- *mypos_up_2_new_tar_real_dist_smaller*: Is it starting from my up grid to the target better?
- *mypos_down_2_new_tar_real_dist_smaller*: Is it starting from my down grid to the target better?
- *mypos_left_2_new_tar_real_dist_smaller*: Is it starting from my left grid to the target better?
- *mypos_right_2_new_tar_real_dist_smaller*: Is it starting from my right grid to the target better?
- *my_pos_up_walkable*: Is my up grid walkable?
- *my_pos_down_walkable*: Is my down grid walkable?
- *my_pos_left_walkable*: Is my left grid walkable?
- *my_pos_right_walkable*: Is my right grid walkable?

All of the above features only have value 1 or 0. 1 means True and 0 means False. The dimension of a sample feature vector for sheep is 1 x 16, looks like: [0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 0, 0, 0, 1, 1]. Given this input, the sample action it took might be -1 (up). The dimension of a sample feature vector for wolf is 1 x 11, looks like: [0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1]. Given this input, the sample action it took might be 0 (none).

3) Multilayer Perceptron (MLP): We hand-crafted the features from the input image and hence reduced the input data from high-dimension to relatively low-dimension. With the (X, Y) pairs we collected before, we can simply train a MLP model to learn the strategy for our agents. However, all of the hand-crafted features are learnable. This can be exemplified by using a Convolutional Neural Network (CNN). For example, we can design a CNN to extract the position of the target from the input image and the prediction accuracy will be 100% in this context. The other features can be also extracted similarly by using one or multiple CNNs. This approach of extracting learnable features through end-to-end learning is graphically represented in Fig. 5.

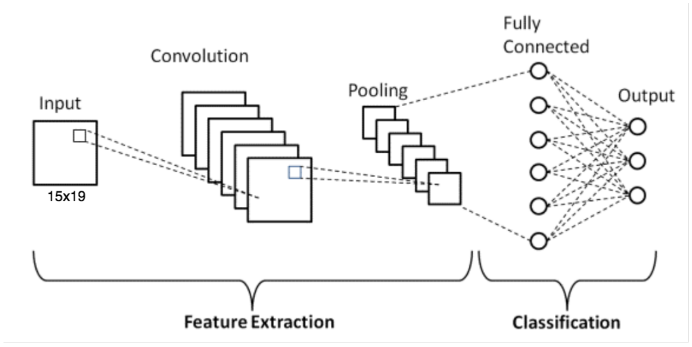


Fig. 5. CNN feature extractor

C. Reinforcement Learning

Reinforcement Learning (RL) differentiates itself from the other two methods we discussed earlier primarily by its ability to learn a policy without reliance on pre-existing data. An RL agent, interacting directly with its environment, learns from the feedback it receives in the form of rewards or penalties as shown in Fig. 6. Through this iterative process, it navigates its way to an optimal policy. We have conducted an evaluation of three different approaches within the realm of RL, each of which will be detailed in this section.

a) Proximal Policy Optimization (PPO): Irrespective of the approach, we employ Proximal Policy Optimization (PPO) as the algorithm of choice [4]. This selection is influenced by the structure of the game, which necessitates discrete input and thus requires an algorithm suitable for discrete action methods, such as A2C, DQN, or PPO. The decision to opt for PPO over the others is primarily due to its superior convergence characteristics. To summarize, the advantages of Proximal Policy Optimization (PPO) can be enumerated as follows:

- It operates on policy, meaning it learns directly from the interactions it experiences.
- It strikes an effective balance between sample efficiency and stability.
- It proves robust in the face of hyperparameter selection, often ensuring reliable convergence.
- It employs a surrogate objective function, complemented by a clipping mechanism, to ensure the stability of updates.

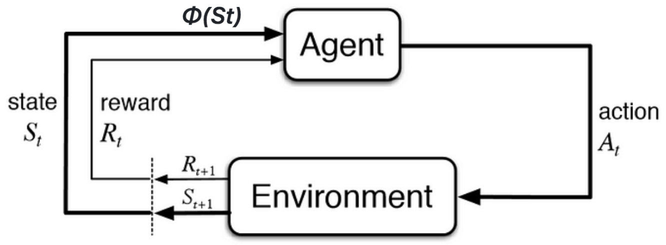


Fig. 6. Use extracted features as a state for RL

b) General Reward Function: In reinforcement learning, the learning process is not guided by labeled data, as in supervised learning, but by reward mechanisms. We have tailored a reward system specific to the Wolf-Sheep game as follows:

For the Sheep:

- The agent loses 0.2 points each turn.
- Consuming grass results in a gain of 1 point.
- Consuming rhubarb results in a gain of 5 points.
- Trapping the enemy sheep with the player's wolf results in a gain of 10 points per turn.

For the Wolf:

- The agent loses 0.2 points each turn.
- Consuming the enemy's sheep results in a gain of 100 points.
- Trapping the enemy sheep with the player's sheep results in a gain of 5 points per turn.

1) Feature Vector learning: In this method, rather than using the complete 15x19 map as our state, we leverage the extracted features derived from the supervised learning algorithm in section III-B2. We apply the PPO algorithm to these states with the objective of maximizing the rewards provided by the environment.

2) End-to-End Learning: In scenarios where computational resources are not a limiting factor, end-to-end learning can be employed without necessitating data preprocessing.

In the Wolf-Sheep game, we take the 15x19 map as out input and use this as our state representatives. We apply the PPO algorithm to these states with the objective of maximizing the rewards provided by the environment.

3) Teacher-Student Framework: The Teacher-Student framework in machine learning involves a knowledge transfer process from a trained model (teacher) to a less experienced model (student). The teacher, being complex and highly performing, imparts its learned behavior to the student model. This distillation process allows the student model, usually smaller and less resource-intensive, to mimic the teacher's performance. In reinforcement learning, this framework aids in bootstrapping the learning process, enabling faster and more effective learning. [6] In our approach, we leverage the pretrained MLP model as a guide for our agent, instead of starting the learning process from scratch. This strategy is encapsulated in the Teacher-Student framework, as illustrated in Fig. 7. To adapt to this framework, we design an intrinsic reward function on top of the general rewards as follows:

- The agent loses 3 points for deviating from the teacher's action.
- The agent gains 0.05 points for performing an action identical to the teacher's.

In the Teacher-Student Framework, the Teacher Policy benefits from privileged access to data, thus facilitating an informed trajectory based on A*. The Student, on the other hand, only has access to map features, so it doesn't have the optimal trajectory but merely a closeness measure.

Some similiar methods such as, Behavior Cloning and Imitation Learning are also employed for knowledge transfer from a source, the 'teacher', to a target model, the 'student'. However, significant differences exist.

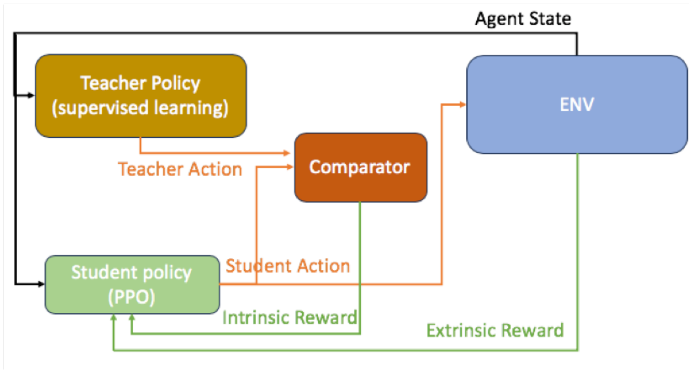


Fig. 7. Teacher-Student Framework

Behavior Cloning is a direct method where the student model mimics the actions of the teacher. This is a form of supervised learning where the teacher's actions function as labels. The student learns a deterministic policy that strives to replicate the teacher's actions given identical observations.

Imitation Learning, conversely, is a broader term encompassing various methods, including Behavior Cloning. Unlike Behavior Cloning, certain types of Imitation Learning, such as inverse reinforcement learning, extend beyond directly copying the teacher's actions. They aim to comprehend the underlying objectives or rewards that the teacher is optimizing for, subsequently employing reinforcement learning to discover an optimal policy given these inferred rewards.

The Teacher-Student Framework, however, is a universal framework for knowledge distillation in which a more sophisticated teacher model assists in training a simpler student model. This framework may incorporate methods like Behavior Cloning or Imitation Learning but can also include other methods. The primary objective here is the transfer of knowledge from the teacher to the student, enabling the student to learn more efficiently and effectively. In some instances, the student model might outperform the teacher post substantial training, as it learns from its successes and errors, in addition to the teacher's guidance.

Each method carries its unique strengths and weaknesses, and the selection among them often relies on the specific context or problem at hand.

IV. Experiment Result

A. Experiment Setup

In our experiments, we conduct training on a laptop equipped with an Intel Core i7 vPro 10th Generation processor, operating at a speed of 3.9GHz. In the context of our game, the RL player takes the first move during the training phase. For evaluation, the RL player takes the second move. During the training sequence, the RL player's opponent is a Random player. The map we used is Fig. 1.

B. Comparative Evaluation of RL Algorithms

We compare the RL algorithms based on the recurring reward per episode. The ideal algorithm would exhibit stable results and learn effectively with a limited number of episodes.

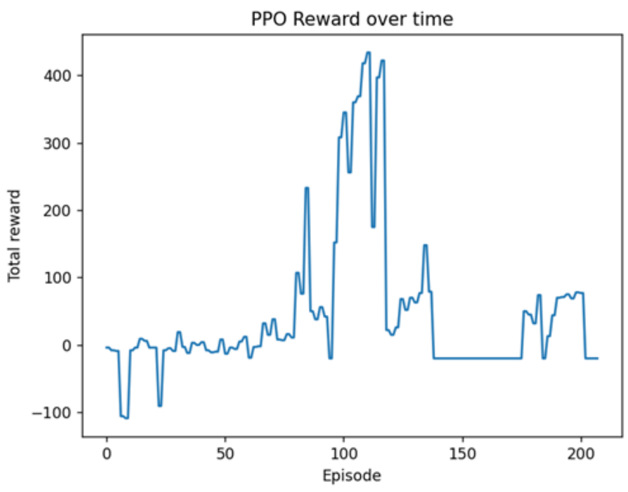


Fig. 8. Recurring reward for PPO algorithm based on end-to-end learning for the sheep player

a) End-to-End Learning: In this method, we trained a PPO agent with the entire map serving as the input state. We trained sheep and wolf separately while the other agent on the same team used the pre-trained MLP strategy. We conducted the training in this environment for 200 episodes. Figure 8 shows the recurring rewards per episode for the sheep player. As observed, the learning process lacks stability and did not converge within this episode limit. Although continued training might yield better results, our objective here is to identify which algorithm can achieve optimal results within limited episodes.

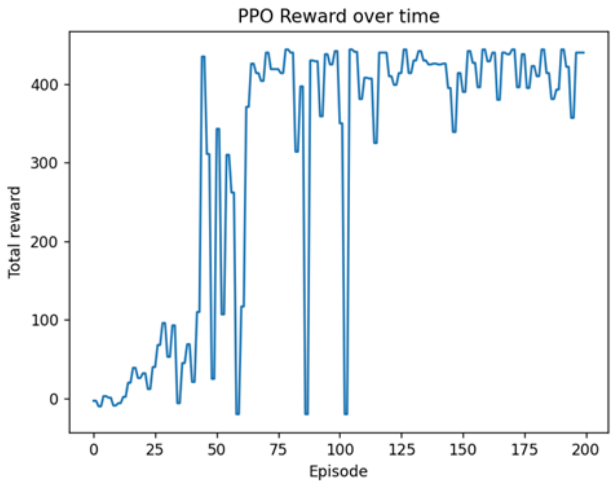


Fig. 9. Recurring reward for PPO algorithm based on feature vector learning for the sheep player

b) Feature Vector Learning: The implementation of this approach closely resembles the end-to-end method, with the difference being the reduced size of the input state due to feature extraction. Like before, we trained the model for 200 episodes. Figure 9 depicts the recurring reward per episode. Evidently, this method of learning on extracted features offers greater stability compared to the end-to-end learning approach, as it starts converging around 430 after 125 episodes.

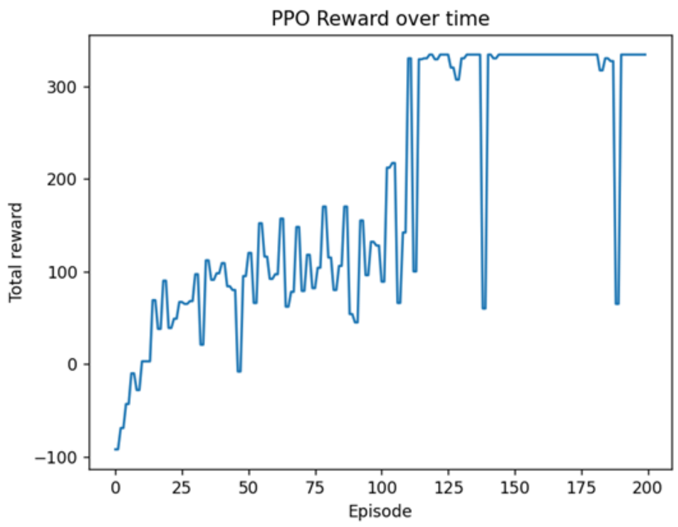


Fig. 10. Recurring reward for PPO algorithm based on teacher-student framework for the sheep player

c) Teacher-Student Framework: From the above comparison, we trained the student policy with the PPO algorithm for 200 episodes, using extracted features as the input state and an additional intrinsic reward from the teacher policy that operated by the pre-trained MLP model. The player's opponent used is Greedy player. The results are shown in Figure 10. It's worth mentioning that the rewards in this graph are not directly comparable to the other two methods due to the distinct reward function design and the different opponent player. Under the teacher-student framework, the result converge fast and stable after 110 episodes, given training in a more complicated environment with a stronger opponent.

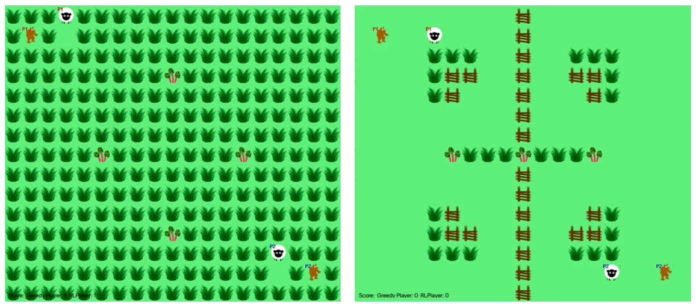


Fig. 11. New test maps

C. Generalization Evaluation

Finally, we tested our RL agent (teacher-student based) on two unseen maps without any further fine-tuning, see Fig. 11.

The first test map has plenty of food and doesn't have any fence. The second test map has several food and a vertical line of fences with a small gap in the middle of the map. In both map, our opponent agent is Greedy player and we take the second move which is disadvantage to the game. The results is shown in Table I.

TABLE I
RL AGENT VS GREEDY AGENT RESULT

	Game Result	**End Reasonn Result**	**Final reward**
Map1 RL	Win	Reach 100 turns	100
Map2 RL	Win	Eat enemy sheep	124
Map1 A*	Win	Eat enemy sheep	100
Map2 A*	Win	Eat enemy sheep	124
Map1 MLP	Win	Eat enemy sheep	100
Map2 MLP	Win	Eat enemy sheep	124

D. Discussion

We also run A* agent and MLP agent under the same conditions, see the results in Table I. Our RL(teacher-student based) agent win the game on both new maps. Specifically on Map2, the wolf eat enemy sheep with the help of sheep from the same team. Furthermore, our RL agent perform as good as the A* and MLP agent.

It is worth noting that solving the game in A* is flexible as it is rule-based. One can add or remove some rules based on some domain knowledge. However, composing this agent takes more than a thousand lines of code. There is no need to hand craft the rules in Supervised Learning algorithm, but we collected more than 4,000 entries of data to train the MLP model. Thanks to the RL algorithm, data collection and domain knowledge is no longer needed to play this game and we can finally perform as good as an expert.

CONCLUSION

In this project, we wrote the gym environment for the game so that it can be used directly for many existing RL algorithm packages. We developed search-based, supervised learning-based and RL-based agents to play this game. We tested our algorithms on different maps to demonstrate its generalizability.

In the future, we will extend the game into partially observable environment. and learn an end-to-end model that the input is the image and output is the action distribution. All the features will be extracted along the way instead of handcrafted separately. Furthermore, the algorithm will be tested on more maps that contain some corner cases.

ACKNOWLEDGEMENT

This project used the GUI from the DDIS group (led by Prof. Abraham Bernstein) of the University of Zurich. We thank them for their support.

REFERENCES

[1] BAKER, B., KANITSCHEIDER, I., MARKOV, T. M., WU, Y., POWELL, G., MCGREW, B., AND MORDATCH, I. Emergent tool use from multi-agent autocurricula. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020* (2020), OpenReview.net.

[2] BOTEA, A., MÜLLER, M., AND SCHAEFFER, J. Near optimal hierarchical path-finding. *J. Game Dev. 1*, 1 (2004), 1–30.

[3] CARTA, T., ROMAC, C., WOLF, T., LAMPRIER, S., SIGAUD, O., AND OUDEYER, P.-Y. Grounding large language models in interactive environments with online reinforcement learning. *arXiv preprint arXiv:2302.02662* (2023).

[4] SCHULMAN, J., WOLSKI, F., DHARIWAL, P., RADFORD, A., AND KLIMOV, O. Proximal policy optimization algorithms. *CoRR abs/1707.06347* (2017).

[5] SILVER, D., HUBERT, T., SCHRITTWIESER, J., ANTONOGLOU, I., LAI, M., GUEZ, A., LANCTOT, M., SIFRE, L., KUMARAN, D., GRAEPEL, T., ET AL. A general reinforcement learning algorithm that masters chess, shogi, and go through self-play. *Science 362*, 6419 (2018), 1140–1144.

[6] ZIMMER, M., VIAPPIANI, P., AND WENG, P. Teacher-student framework: a reinforcement learning approach. In *AAMAS Workshop Autonomous Robots and Multirobot Systems* (2014).