LAB GUIDE 02

In the last lab, we experimented with single- and multi-agent virtual environments and compared the performance of random agents with greedy agents. In this lab, we will see the limitations of these types of agents in multi-agent scenarios with higher complexity and understand how we can incorporate coordination mechanisms to create better-performing agents. The code for this laboratory can be found in the following repo: https://github.com/GAIPS/aasma-spring-23/tree/lab2

1. Motivation

Last time, we compared teams with random and greedy agents for scenarios with two required captors (N=2) and observed how simple greedy agents could perform much better than random agents for these scenarios. Now we will add complexity by increasing the number of required captors to N=4. This means a prey is considered captured if and only if it is cornered by all predators, as shown in **Fig. 1**. Note that, with this restriction, if the prey goes to a corner or a wall it cannot be cornered by the predators.

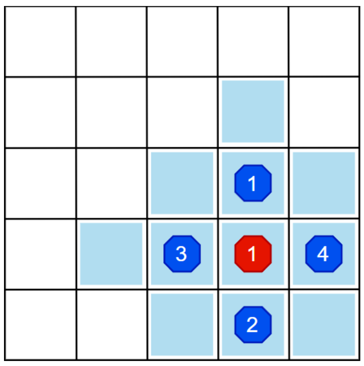


Fig. 1: The predator prey environment where four predators must catch the prey.

The prey must be cornered on all four sides to be considered captured.

1.1 EXERCISE 1: NUMBER OF REQUIRED PREDATORS

Open file *exercise_1_no_coordination.py*. The file contains code, similar to that implemented in lab 1, that compares the performance of the teams of random and greedy agents. Below, you can find an image showing the performance of the random and greedy teams when two captors are required to capture the prey. This scenario corresponds to the one considered in the previous lab.

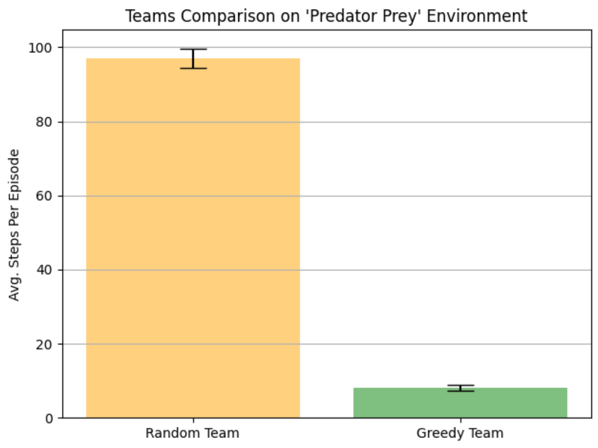


Fig. 2: Results of running *exercise_1_no_coordination.py* with *required_captors=2*.

As opposed to lab 1, where only two predators were required to capture the prey, we will now change the code so that we increase the number of required predators to capture the prey to four. Implement line 23 in *exercise_1_no_coordination.py* so that four predators are required to capture the prey.

Afterwards, run the python file. You should be able to see a plot similar to the one below, which depicts each team's average number of time steps to capture the prey (alongside a 95% confidence interval).

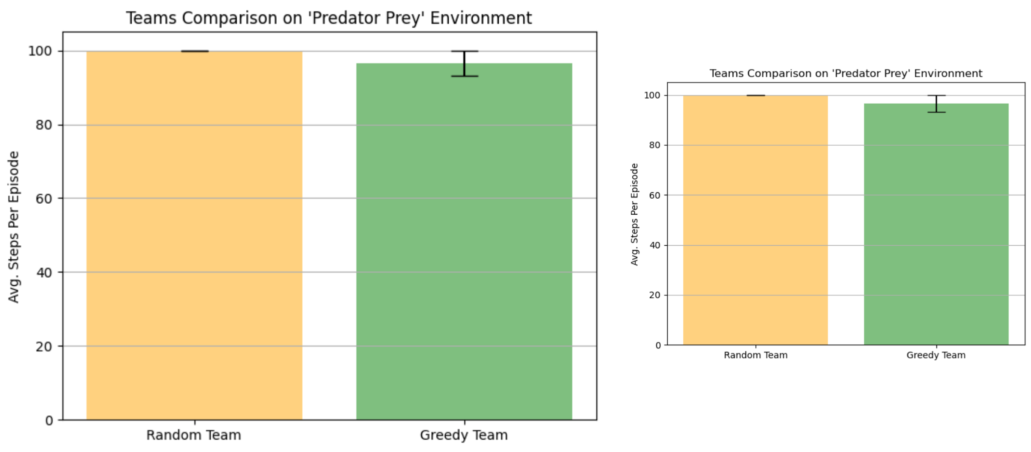


Fig. 3: Results of running *exercise_1_no_coordination.py* with *required_captors=4*.

Now try to answer the following questions:

1. On average, with *required_captors=4*, how many time steps does it take for the episode to terminate for each team? 100 for team random, 98 for greedy
2. When we increase the number of required captors, we can see that the performance of the greedy team deteriorates. Can you think of some limitations that could explain why the Greedy Team is performing more poorly compared to the previous scenarios? Because there is no communication between captors and it is harder to coordinate multiple agents at once.
3. How could we improve the performance of the greedy team?
   Add some sort of communication mechanism or social rules to improve cooperation.

2. Coordination Games

We will implement coordination mechanisms using the framework of normal-form games. Remember that a game in normal form is a game where (among other things) there is more than one agent that has a set of actions from which they can choose from. For each joint action selected by the agents, a payoff function dictates the utility of a joint set of actions (i.e., how good is this joint action). Having this in mind, we can define our predator-prey scenario as a normal-form game as follows:

$$P = \{P_1, P_2, P_3, P_4\},\ n = 4$$

$$A_i = \{GoN, GoS, GoW, GoE\},\ i \text{ is the index of the predator in } P$$

$$u_1(a) = u_2(a) = u_3(a) = u_4(a)\ \ u(a) = \begin{cases} 1,\ if\ \ a_1 \neq a_2, a_1 \neq a_3,\ a_1 \neq a_4, a_2 \neq a_3, a_2 \neq a_4, a_3 \neq a_4 \\ 0,\ otherwise \end{cases}$$

Where P corresponds to the set of predator agents, and A_i defines the set of actions available to agent i. Note that the set of actions is the same for all predator agents. The actions GoN, GoS, GoW, and GoE, describe the action of going to the North (N), South (S), West (W), or East (E) side of the prey, respectively (as shown in **Fig. 4.**). The payoff function u is shared by all agents, reflecting their preference for the joint action a. The agents have a payoff of 1 if all of them choose different actions from each other; otherwise, they receive a payoff of 0. We note that, since this is a pure coordination game, the payoff function is shared between the agents.

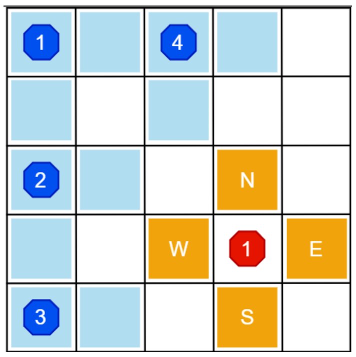


Fig. 4: Predator-prey problem illustration.

Now try to answer the following questions:

1. How many joint actions does the game have? Justify your answer.
   Each agent can choose 4 actions, hence we have 4x4x4x4 joint actions
2. How many Pareto optimal Nash Equilibria are in the game? Justify your answer.
   If the first agent chooses a given action, the next to choose can only choose 3 actions and so on for us to have a payoff of 1 (where all the actions chosen are different)...
   In total, there 4! = 4 x 3 x 2 x 1 Pareto optimal Nash Equilibria since these joint actions aren't Pareto dominated.

3. Social Conventions

We have defined that coordination in normal-form games is the process in which a group of agents choose a single Pareto optimal Nash equilibrium. Hence, in the predator-prey environment, a coordination mechanism should make the predators choose the same Pareto optimal Nash Equilibrium. One mechanism that can be used to solve this problem is *social conventions*. Remember that a social convention is a rule that dictates how the agents should choose their actions in a coordination game. In addition, social conventions are common knowledge among agents (i.e., every agent is aware of them) and no agent can benefit from not abiding by this rule.

An easy way of doing this is to define a unique ordering scheme that all agents should follow when deciding which action to select. With this ordering, each agent first computes all Pareto optimal Nash Equilibria of the game and then selects the Nash Equilibrium according to the ordering scheme. Having this in mind, we can define the following social convention:

$$Order\ of\ Agents: P_1 \succ P_2 \succ P_3 \succ P_4$$

$$Order\ of\ Actions: GoN \succ GoS \succ GoW \succ GoE$$

Using this social convention, agent P_1 has priority over all other agents and chooses an action first. When choosing an action, P_1 will prefer GoN, because it comes first in the action ordering. Agent P_2 is next in the order of agents, so knowing that P_1 has priority and will choose GoN, P_2 will choose the GoS action, and so on for P_3 and P_4. Note that since the social conventions are common knowledge, the agents don't need to communicate with each other. They can simply follow the ordering scheme to come up with their actions that will yield the same Pareto optimal Nash Equilibrium for all agents.

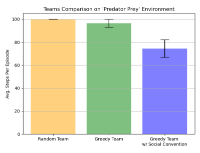
Now try to answer the following questions:

1. Which Pareto optimal Nash equilibrium will be selected by the agents given the social convention above?
   (a1, a2, a3, a4) = (GoN, GoS, GoW, GoE)

3.1 Exercise 2: Social Conventions

Open the file *exercise_2_social_conventions.py*. The file contains a class named *ConventionAgent*, which implements an agent that uses a coordination mechanism based on a social convention, as well as some code to run experiments. Your task is to complete the code and implement the social convention-based agent. The agent should, given access to the social convention (shared among agents), decide on which side it should aim to corner the prey (N,S,E, or W) and move in the corresponding direction. Implement line 40 on *exercise_2_social_conventions.py*.

After running the code, you should be able to see the following plot, which depicts each team's average number of time steps to capture the prey (with a 95% confidence interval):



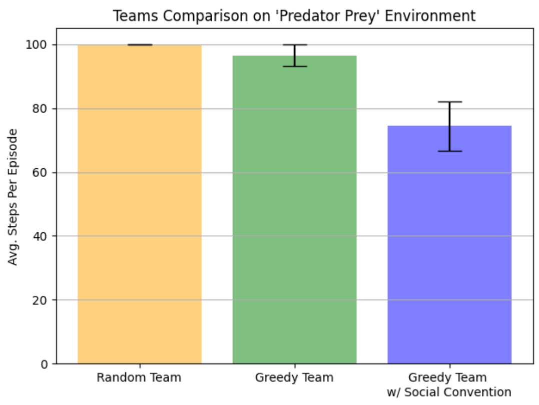


Fig. 5: Results of running *exercise_2_social_conventions.py*.

Now try to answer the following questions:

1. Which team has the best performance? Why do you think this happened?
   Greedy team with social conventions. Because they could better coordinate...
2. Can you think of any limitations to using social conventions?
   They are highly restrictive in terms of what actions agents choose to perform. Aside from that, this method also assumes agents compute all the equilibria in a game before choosing a single one, which can be computationally expensive.

4. Roles

An alternative to the coordination mechanism with social conventions is to assign roles to predator agents. A role restricts the number of actions that an agent can select, i.e., if an agent is given a role at a particular state, some of its actions might be unavailable (e.g., a football player in the defense role cannot choose to score a goal). Since roles restrict some actions, this also simplifies the computation of the Pareto optimal Nash Equilibria (because we only need to calculate the Pareto optimal Nash Equilibria of a smaller game that does not include the restricted actions). For our game, let us define the following roles and the order by which the roles must be assigned:

$$Roles\ of\ Agents:\ R\ =\ \{N,\ S,\ W,\ E\}$$

$$N:\ Agent\ goes\ to\ the\ north\ (N)\ square\ adjacent\ to\ the\ prey$$

$$S:\ Agent\ goes\ to\ the\ south\ (S)\ square\ adjacent\ to\ the\ prey$$

$$W:\ Agent\ goes\ to\ the\ west\ (W)\ square\ adjacent\ to\ the\ prey$$

$$E:\ Agent\ goes\ to\ the\ east\ (E)\ square\ adjacent\ to\ the\ prey$$

$$Order\ of\ Roles: N \succ S \succ W \succ E$$

We also define the following potential function (i.e., function that computes how appropriate an agent is for a specific role given the current state of the world):

$$potential_{ij} = \ -\ distance(p,\ r),\ where\ p \in P\ =\ \{P_1, P_2, P_3, P_4\},\ r \in R,$$

Where *distance* calculates the Manhattan distance between the predator's position and the target position of one the prey's adjacent squares (i.e., north square, south square, west square, or east square). Assuming that each agent can only have one role, the potential function above assigns a role to an agent that is closest to the corresponding square, e.g., if P_2 is closest to the north square then he receives the N role. Note that every agent knows the potential function and is able to assign a role for every agent. Therefore, the agent-role assignment can occur in parallel and without communication for each of the agents.

4.1 Exercise 3: Roles

Open the file *exercise_3_roles.py*. The file contains a class named *RoleAgent* that implements a role-based agent, as well as code to run some experiments. Your task is to complete the code and implement the role-assignment process. Implement:

- the *potential_function* method, which calculate the potential function given by the Manhattan distance between the agent position and the target position given the role;
- the *role_assignment* method, which first calculates the potential function for each agent-role pair. Then, iteratively assigns, given the fixed role order, a role to each of the agents. Make sure that, in case more than one agent has the same potential value for a given role, the agent selection is done in a consistent (deterministic) manner.

After running the code, you should be able to see the following plot, which depicts each team's average number of time steps to capture the prey (with a 95% confidence interval):

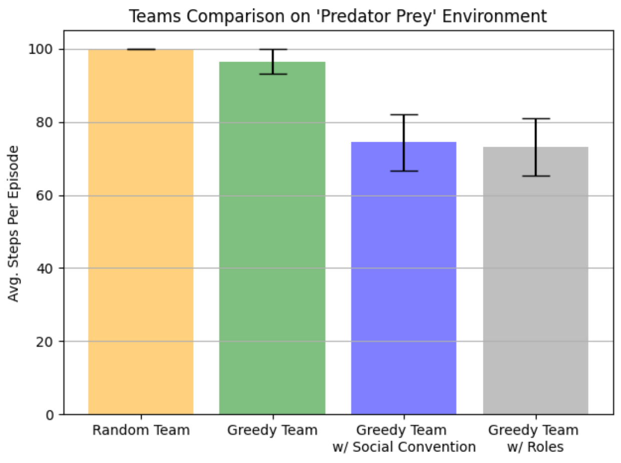


Fig. 6: Results of running *exercise_3_roles.py*.

Now try to answer the following questions:

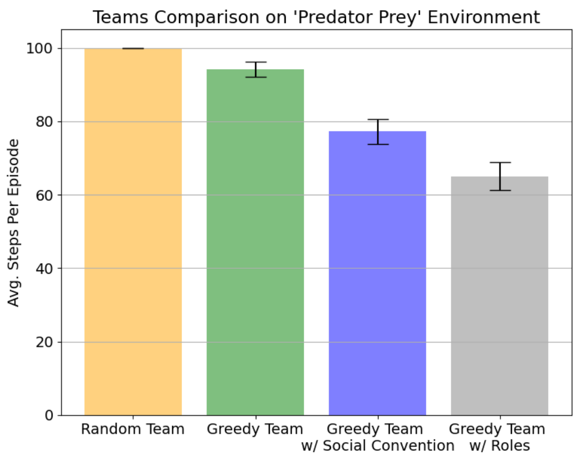
1. Which team has the best performance? Why do you think these results happened?

Greedy team with roles -> their behavior takes into account their current position and they can change goal destinations as the simulation evolves, leading to a more strategic behavior.

Bonus question:

1. Run again the code but now change the grid shape to (25, 25) and the number of episodes to 400. You should obtain a plot similar to Fig. 7. Which team has the best performance? Justify.

Greedy team with roles. Having more time to coordinate, having roles proves to be better than simple social conventions because it allows the agents to adapt their goal position considering their manhattan distance to the possible targets, facilitating a more dynamic and strategic answer.



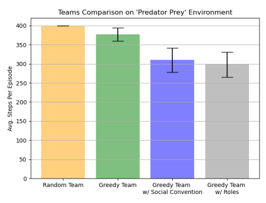


Fig. 7: Results of running *exercise_3_roles.py, when setting grid_shape=(25, 25) and episodes=400*.

The End