CS 520 Project 2

The Circle of Life

Submitted by

Chandhanu Mohan Kalamani – cm1573
Sai Chaitanya Chaganti – sc2482

Instructor

Dr. Charles Cowan

November 16, 2022

Contents

1. Project Statement

2. Settings

    2.1 The Environment
        2.1.1 Implementation
    2.2 The Predator, The Prey and The Agent
        2.2.1 Implementation

3. Agents and Environments

    3.1 The Complete Information Setting
        3.1.1 Agent 1 & Implementation
        3.1.2 Agent 2 & Implementation
    3.2 The Partial Prey Information Setting
        3.2.1 Agent 3 & Implementation
        3.2.2 Agent 4 & Implementation
    3.3 The Partial Predator Information Setting
        3.3.1 Agent 5 & Implementation
        3.3.2 Agent 6 & Implementation
    3.4 The Combined Partial Information Setting
        3.4.1 Agent 7 & Implementation
        3.4.2 Agent 8 & Implementation
    3.5 Agent 9
        3.5.1 Implementation

4. Analysis

    4.1 The Complete Information Setting
    4.2 The Partial Prey Information Setting
    4.3 The Partial Predator Information Setting
    4.4 The Combined Partial Information Setting
        4.4.1 Defective Drone
    4.5 All Setting's Comparison
    4.6 All Setting's Graph Analysis
    4.7 Bonus

5 Conclusion

1 Project Statement

The goal of this project is to create a probabilistic model of an uncertain environment and use it to inform and direct decision making. You are pursuing a prey object while being pursued by a predator object, and you want to capture your target while avoiding capture yourself. This will be complicated by your inability to see where the predator and prey are - but you must use the information you have to make the best decisions possible.

2 Settings

2.1 The Environment

This project's environment is a graph of nodes connected by edges. The agent, prey, and predator can all move between nodes along the edges. A large circle is formed by 50 nodes numbered 1 to 50. In addition, we add edges at random to increase connectivity across the circle, as shown below:

- Choosing a random node with a degree less than 3
- Within 5 steps forward or backward along the primary loop, add an edge between it and one node. (Thus, node 10 may be connected to node 7 or 15, but not to node 16.)
- Repeat this process until no more edges can be added.

2.1.1 Implementation

Language: Python
Visualization: NetworkX
Math: NumPy

A cyclic graph with 50 nodes and edges that are randomly added either forward or backward by 5 node steps/node serves as the setting for this problem statement. There are a maximum of 3 degrees for each node. Node cannot contain an edge over itself. These edges are added at random using the aforementioned criteria.

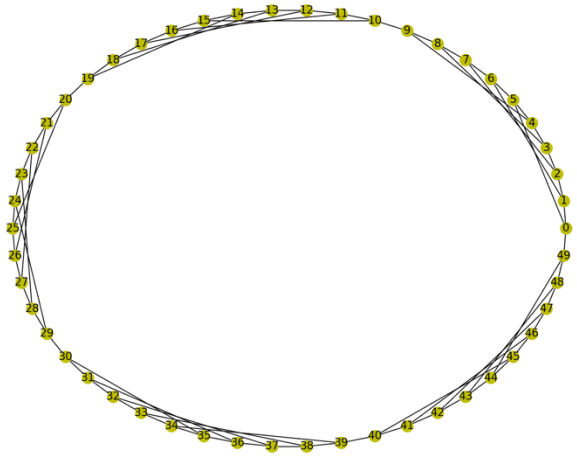


Figure 1

Graph Theory Question: With this setup, you can add at most 25 additional edges (*why?*). Are you always able to add this many? If not, what's the smallest number of edges you're always able to add?

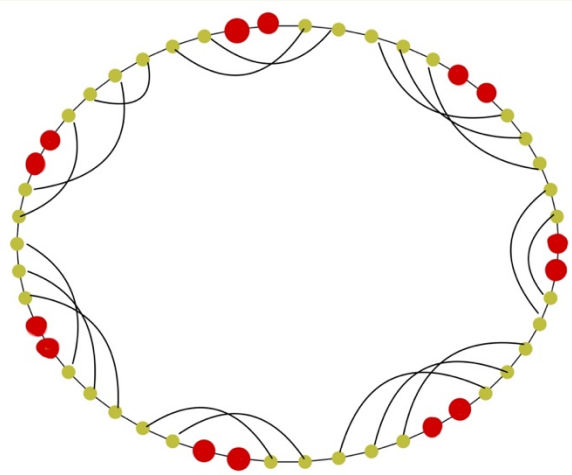


Figure 2

Answer: The most edges we can add in our environment, a cyclic network with 50 nodes, is 25. (Check Figure 1). In our configuration, we could regularly add 25 edges to our graph, with a minimum of 20. However, the smallest number of edges we could do by observation is 18.

Class: Node ()

The node is the primary data structure that is used to create the Graph space. The node contains information of the index, neighbors, current occupancy (e.g.: by Prey, Agent, or Predator) and degree.

Class: Graph_Utils (Node_count):

Type: Node ()

The collection of nodes is created by inheriting from the class Node. For our problem statement, Node count is initialized to 50 and utilized to create the graph environment. This class only accepts Node count as a variable. A cyclic graph is originally created via the CreateEnvironment () function with the aid of the NetworkX module. The function generate edges () are used to randomly add edges to the graph under the aforementioned conditions, i.e., adding edges either forward or backward by 5 nodal distances. Using Node (1) as an example, Node (0), Node (2), and Node (6) or Node (0), Node (2), and Node (46). This function can only ever add a maximum of 25 edges.

Utility functions in the Graph Utils class include shortest path (), which is an implementation of BFS. The Graph utils class has been improved to be compliant with the networkx API, which also supports the Node () class. The function all_shortest_path (graph) is used to find the shortest path from all nodes to all nodes, and it is widely used in Agent's classes to compute agent movement. Graph Utils additionally provides two visualization functions, grap.display () and graph.display_shortest_path (path) to display the graph and graph with specified path, respectively.

2.2 The Predator, The Prey and The Agent

Three entities inhabit this environment: the Predator, the Prey, and the Agent, who can move from node to node along the edges. The Agent wishes to capture the Prey, while the Predator wishes to capture the Agent. If the Agent and the Prey share a node, the Agent wins. When the Agent and the Predator share a node, the Agent loses. For the sake of simplicity, the Predator and Prey can coexist in the same node. The three players move in rounds, beginning with the Agent, then the Prey, and finally the Predator.

The Prey: The Prey's rules are simple: whenever it moves, it chooses between its neighbors or its current cell at random (i.e., if it has three neighbors, it has a ¼ chance of staying where it is). This continues until the game ends, regardless of the actions or locations of the others.

The Predator: When the Predator moves, it examines its available neighbors and calculates the shortest distance to the Agent for each neighbor to which it can move. It then moves to the neighbor with the shortest distance to the Agent. If multiple neighbors are within the same distance of the Agent, the Predator chooses one at random.

The Agent: Agent's movement is determined by the Agent's specific strategy. Before deciding where to move in a partial information setting, the Agent may choose to survey a node at a distance to determine what, if anything, is in that node. (Imagine, for example, the Agent dispatching a drone to spy on a specific location.) It should be noted that the Agent is aware of how the Predator and Prey choose the actions they will take but is unaware of the specific actions chosen.

2.2.1 Implementation

Prey:

Prey is a goal that the Agent must achieve, and hence this determines the success rate of the Agent's runs. After each time step, the prey's position changes at random. The prey can roam around at random or stay in the same node. As a result, the likelihood of prey moving to one of its potential nodes (neighbors (degree 3) + or remaining in the same node) in the following time step is always uniform. Agent is successful if it captures the prey.

Class Prey (graph, graph.Node_count):

The prey class uses the graph object from CreateEnvironment and its Node count to launch the prey class and place the prey in the network at random.
Member functions of the prey:
1. prey.get position () returns the prey's position at any given time.
2. prey.move () is used to generate motions for the prey, allowing it to move randomly to its neighbor or remain stationary. When the game is finished, the internal variable mark can be toggled to gather the prey's trail.

Predator:

Predator is likewise a goal, but the agent should avoid it; thus, the failure rate of any agent in question is defined. In the complete information setting, the predator always approaches closer to the agent, taking the shortest path feasible, and seeks to catch the agent. With only limited prey information, the predator runs directly towards the Agent, but with distraction. This means that the predator has a 40% risk of becoming sidetracked rather than following the prey in the shortest possible path. It will only have 60% of the time to pursue the agent. This creates a difficult condition for detecting the predator even after the survey claims it was successful, because we cannot determine the position of the predator in the next step even if we discovered it in the current time step. If the predator captures the agent, the agent has failed.

Class Predator (graph, graph.node_count):

The predator class also uses the graph object from CreateEnvironment and its Node count to start the predator class and put the predator at random in the network.
The predators' functions are as follows:
1. predator.get_position (): returns the predator's position.
2. predator.move (agent pos): A smart predator's action that always advances towards the agent pos in order to kill the agent by computing the shortest path between the predator and the agent. (Agents 1, 2, 3, and 4) predator.move or distract (agent pos): This is a distracted predator that only chases the target 60% of the time and is distracted 40% of the time.

Agent:

Each agent's implementation is classified from 1 to 9. The agent's success rate is defined as the number of times it can catch the prey while avoiding the predator. The Agent always tries to move toward the prey when the position is known, in the complete information setting, or it explores a node and moves toward the node with the highest probability of where the prey is, and the predator isn't.
Goals:
1. Avoid the predator; if the predator captures the Agent, it is regarded a failure.
2. If you catch the prey, you've succeeded.

Class Agent (graph, prey, predator):

To begin, the agent is given objects from the graph, prey, and predator. It will deploy the agent in the graph's surroundings at random. In a complete information setting, the agent computes the shortest path to the prey while avoiding the predator and chooses the next best node; in a partial information setting, the agent will probabilistically determine the location of the prey/predator by tracking the vector of beliefs of each node. A HashMap is used for this, with the key being the node's index and the value being its belief. The agent will assume that the prey/predator is in the index with the highest value of belief. The belief vector's total is always 1. The agent's movement is determined, and the next node is picked by the priority of the following requirements (Mentioned under 3.1.1) in both the Complete information setting and the partial/combined information configuration.

Agent's goal in complete information setting:

Agent will run in the sequence specified in run () below and verify the game's state. Because the agent knows the location of both prey and predator, it simply computes the distance between agent and prey; agent and predator; and the agent moves towards the goal based on this information. Agent.run() is the main method that makes use of agent.move, agent.status(), and/or agent.update belief(survey=True) (based on the description). The run function additionally provides an internal count (Threshold) to handle the edge situation of agents wriggling around the graph but failing to reach the goal.

Complete information setting order:

1. Agent.status()
2. agent.move()
3. Agent.status()
4. prey.move()
5. agent.status()
6. predator.move()
7. agent.status()

agent.status():

This returns the state of the run and terminates it based on the two circumstances listed below.
1. Agent is apprehended by a predator and returned Failure
2. Agent captures prey and returns it Success

Note: If both the predator and the prey occupy the same node and the agent moves into it, it will die at the hands of the predator and be considered a failure, even if the agent caught the prey.

There is an agent. Move() is used to generate motions for an agent that meet the seven conditions specified in the problem statement in order to catch the prey while avoiding the predator.

Information Settings:

In the incomplete information setting, when the exact position of a prey/predator is estimated probabilistically in the form of belief vectors, the node with the highest belief is presumed to be inhabited by a prey/predator. Based on the number of nodes in our environment, the belief vectors have 50 elements. The key is node index, and the value is beliefs. The belief represents the likelihood of a prey/predator being present in a certain node during the current timestep. Next the following steps, the beliefs are updated, and the process is repeated until we reach the goal (or halt the run and consider it as a failure in edge cases where the agent doesnt die but did not reach goal).

Once run() is called, the agent will survey a node to see whether there is a prey/predator in the survey node. The belief is updated based on the survey results using conditional probability criteria. This is true for all the predator/prey actors. The beliefs are updated by computing the P(Actors at node X/survey results at Y).

This is written as,
P(Actor at node A/ Result of Survey B) = P(Actor is at node A, Result of Survey B)/P(Result of Survey B) =P(Actor is at node A) * P(Result of Survey B | Actor is at A) / P(Result of Survey B) =P(Actor is at node A) * P(Result of Survey B | Actor is at A) / P(Result of Survey B) =P(Actor (Actor is at node A) * P(Survey B Result | Actor is at A) / (1 - removed probability)

This is accomplished through the use of the following functions:

agent.survey():

This function is called when update belief(survey=True) selects the node with the highest likelihood of containing an actor and surveys it to see if the actor is there or not. It will update the beliefs based on the outcomes.
If the survey detects the actor at node N, the belief vector will be updated with a 1 for the surveyed node and a 0 for the others. Survey results In this situation, success.
If the survey was unable to detect the presence of an actor, the surveyed node's belief becomes 0 and the others are normalized by dividing their current probability by the new sum of the belief vector. Survey results in this situation, failure. The update beliefs function will update the belief based on the survey results in both cases.

agent.update beliefs():

In the partial and combination partial information settings, this function is employed. It is used to update the belief vector and is normalized so that the belief sums to 1. Based on the issue statements, this part implements conditional probabilities. The belief is revised in order to locate the P. (prey in X next).
P(Actor in A next) = P(Actor in A next, Actor in A now) + P(Actor in A next, Actor in B now) + P(Actor in A next, Actor in C now) + P(Actor in A next, Actor in C now) +.....

= P(Actor in A now)*P(Actor in A next / Actor in A now) +P(Actor in A now)*P(Actor in A next / Actor in A now) +P(Actor in A now)*P(Actor in A next / Actor in B now) +P(Actor in A now)*P(Actor in A next / Actor in C now) +...

There are two belief vectors maintained in the incomplete information setting: believes predator and believes prey. These belief vectors are normalized and updated in accordance with conditional probability criteria.

agent.believes prey:

Before the run, the believes prey is initialized to have a 1/49 probability to all nodes except the one where the agent dwells. Because the prey moves at random or chooses to remain in the same node, the agent's probability of going to the next available node (Deg(node)+node itself = degree +1) is equally distributed. The vector is updated using 1/(degree+1).

agent.believes predator:

Once the agent is deployed, the agent first knows the location of the predator, therefore the beliefs are updated to have 1 where the predator lives and 0 for the rest of the nodes. Because the predator can be distracted by a 40% chance, the beliefs are updated accordingly. While updating beliefs, the distracted predator can be represented as prey, so that total probability = sum of probability over focused predator + sum of probability over distracted predator.

P(predator moving to node from B) = 0.6*P(Predator moves from A to B) + 0.4*P(Prey moves from A to B) = 0.6*(SumOverB - P(Predator moves from A to B)) + 0.4*(SumOverB - P(Prey moves from A to B)) + 0.4*(SumOverB - P(Prey moves from A to B))

Analysis and evaluation of the project:

Simulation of Tests:
The data is obtained over 3000 iterations by running all of the agents in a test environment that is simulated to run 100 times on 30 different paths. Different variations of the actors are examined in each experiment, and an average success rate is calculated.
Rate of success: The number of times the agent successfully captures the prey
Failure rate equals the number of times the Predator catches the Agent multiplied by the number of times it meets the threshold count.

A test agent is used to track each agent's progress and compare findings. The test agent is blind to the predator and always approaches the prey greedily. This means it will not be able to avoid the predator. The test agent runs 15000 times during our simulation, with an average success rate of around 65%.

3 Agents and Environments

In each case, the Predator, Prey, and Agent begin at random nodes (though the Agent should never start on an occupied node).

3.1 The Complete Information Setting

In this setting, the Agent always knows exactly where the Predator is and where the Prey is.

3.1.1 Agent 1

Description:

When it is this Agent's turn to move, it will examine each of its available neighbors and choose from them in the order listed below (breaking ties at random).

- Neighbors that are closer to the Prey and farther from the Predator.
- Neighbors that are closer to the Prey and not closer to the Predator.
- Neighbors that are not farther from the Prey and farther from the Predator.

- Neighbors that are not farther from the Prey and not closer to the Predator.
- Neighbors that are farther from the Predator.
- Neighbors that are not closer to the Predator.
- Sit still and pray.

Implementation:

Methods are similar to the general implementations mentioned in the previous section.
Agent 1 is always informed of the predator's and prey's locations. Agent 1 is the most efficient agent, as it just computes the shortest path between itself and the prey and based its moves on the rules. When the predator is one of the neighbors and the agent's next feasible node is also a neighbor of the predator, the agent fails. If the prey and predator are in the same position, Agent 1 will fail because it will turn blind towards the predator and approach the prey.
Agent1.move() will compute the next possible node based on the aforementioned conditions; if there are more than one possible node, agent 1 will randomly break ties.

3.1.2 Agent 2

Description:

Agent 2 performs the same functions as Agent 1 in the comprehensive information environment. Agent 2 is an upgraded version of Agent 1 who follows the same guidelines. The improvement logic is used to select the agent's next feasible node by exploring where the prey might be in the future (timestep T+1), rather than going towards the prey in the present timestep T.

Implementation:

Understanding the movement of the prey, the run heuristic() function is designed to determine the heuristic to identify the ideal node for the agent 2 to choose its next best potential node. The average shortest path between the agent's next possible position to the prey's next possible nodes is the heuristic. Because Prey moves at random, its likelihood is dispersed uniformly among its neighbors and itself. This heuristic function provides a better ideal node for the agent 2 to move to and considerably enhances the agent2's performance. If there is more than one optimal node, run heuristics is called again from the prey's neighbors to the prey's neighbors, and the average shortest path cost is used to break the ties.

3.2 The Partial Prey Information Setting

Every time the Agent moves, the Agent can first select a node to scan (anywhere in the graph) to see if the Prey is there. In this scenario, the Agent always knows where the Predator is, but does not necessarily know where the Prey is. Every time the Agent enters a node, and the Prey isn't there, the Agent learns where the Prey isn't. In this scenario, the Agent must keep track of a belief state—a collection of probabilities for each node that the Prey is there—for the location of the Prey. These probabilities need to be revised each time the Agent gains new information about the Prey. This probability must be updated whenever the Prey is known to move.

In these cases, how should you be updating the probabilities of where the Prey is? Be explicit and clear. There is a right answer to this, and there is definitely a wrong answer that every year proves very popular.

Answer:
. P (Actor at node A/ Result of survey B)
= P (Actor is at node A, Result of Survey B)/P (Result of Survey B)
*=P (Actor is at node A) * P (Result of Survey B| Actor is at A) / P (Result of Survey B)*
*=P (Actor is at node A) * P (Result of Survey B | Actor is at A) / Sum of Probabilities*
*=P (Actor is at node A) * P (Result of Survey B | Actor is at A) / (1 - removed probability)*
. P(Actor in A next)
= P(Actor in A next, Actor in A now)
+P(Actor in A next, Actor in B now)
+P(Actor in A next, Actor in C now)
+ …..
*= P(Actor in A now)*P(Actor in A next / Actor in A now)*
*+P(Actor in A now)*P(Actor in A next / Actor in B now)*
*+P(Actor in A now)*P(Actor in A next / Actor in C now)*
+...

3.2.1 Agent 3

Description:

When it is this Agent's turn to move, if it is unsure of the location of the Prey at that time, it will investigate the node that has the highest likelihood of holding the Prey (breaking ties at random), and then revise its probability estimates considering the findings. After that, it will proceed according to Agent 1's rules, assuming that the Prey is in the node with the highest likelihood of now containing the Prey. Notably, there is no need to survey any node if it is known exactly where the Prey is.

Implementation:

The location of either the prey or the predator is unclear in the partial information condition. The whereabouts of the other actor, on the other hand, is always known to the agent. The conditional probabilities - P(Finding Actor in A node now), P(Finding Actor in A node next) described above are used to forecast the movement of the agents and are utilized to estimate the position of the unknown actor.

The belief vectors are changed in the Partial/Combined-Partial Information configuration as follows...

For each run:
1. agent.statement ()
2. Surveys the position of the prey/predator
3. change beliefs ()
4. agent.move ()

5. agent.status ()
6. belief update ()
7. hunt.move ()
8. agent.status ()
9. predator.action ()
10. agent.status ()
11. belief update ()

It is worth noting that the beliefs are updated twice before the prey or predator takes a move. That is, before the prey/predator moves, it will know the information of three nodes in advance (Agent's old position, Agent's current position, and surveyed node). Based on this information, the belief vector is updated, and after the predator moves, the transitional belief is updated based on the movement described for each prey and predator.
Agent 3 is a partial information setting actor who does not know the position of the prey but always knows the location of the predator. Agent 3 will begin gathering information by inspecting the node with the highest chance of where the prey is and deciding where to go next using the same rules as Agent 1 to base its moves. Agent 3 uses belief vectors to determine the location of the prey by extracting the node with the highest likelihood of locating a prey. If two or more nodes have the same maximum belief, Agent 3 will break ties at random.

The Survey () function will survey the node with maximum probability and update the belief vectors based on the results. If the survey result is positive, i.e., the survey finds the prey, it will update the node's belief to 1 and the rest of the nodes to 0. If the survey fails, i.e., there is no prey in the node, it will update the node's belief to 0 and rest by 1/49.
When the agent is moved, it now has ideas for three nodes: the agent's old position, the agent's new position, and the surveyed node, and Update belief () will update the beliefs based on this information. The transitional probabilities are updated using transition once the prey is moved, which has a uniform probability for the next feasible node ()

3.2.2 Agent 4

Description:

Agent 4 is a partial information setting actor who does not know the position of the prey but always knows the location of the predator. Agent 4 is an improved version of Agent 3 that begins collecting information by surveying the node with the highest probability of where the prey is and then decides where to go next using the same rules as Agent 1 to base its movements.

Implementation:

Agent 4 has its own run_heuristic () function, similar to Agent 2, that uses heuristics to locate the next best node for the agent to move to, taking into account the information gathered from the survey and the prey's maximum belief location. Agent 4 will apply a heuristic to the node position with the highest probability of discovering the prey. The average shortest distance between agent's next possible node and max belief(prey)'s next possible node is used as the heuristic. The node with the shortest average distance is chosen for our agent to move to. If two nodes have the same max beliefs, the same run heuristic is used on these two nodes and the prey's next position to break the tie.

3.3 The Partial Predator Information Setting

In this scenario, the Agent is always aware of the location of the Prey but may not always be aware of the location of the Predator. Before making any movement, the Agent can select any node in the network to survey in order to detect whether the Predator is there. Every time the Agent enters a node, and the Predator is not present, it learns where the Predator is not. In this scenario, the Agent must keep track of a belief state for the Predator's location, which is a set of probabilities for each node the Predator is present. This probability must be updated every time the Agent discovers new information on the Predator. This probability must be updated whenever the Predator is known to move. The Agent begins in this setting by being aware of where the Predator is.

How do the probability updates for the Predator differ from the probability updates for the Prey? Be explicit and clear.

Answer: Once the agent is deployed, the agent first knows the location of the predator, therefore the beliefs are updated to have 1 where the predator lives and 0 for the rest of the nodes. Because the predator can be distracted by a 40% chance, the beliefs are updated accordingly. While updating beliefs, the distracted predator can be represented as prey, so that total probability = sum of probability over focused predator + sum of probability over distracted predator.

*P(predator moving to node from B) = 0.6*P(Predator moves from A to B) + 0.4*P(Prey moves from A to B) = 0.6*(SumOverB - P(Predator moves from A to B)) + 0.4*(SumOverB - P(Prey moves from A to B)) + 0.4*(SumOverB - P(Prey moves from A to B))*

3.3.1 Agent 5

Description:

This Agent will inspect the node with the highest likelihood of holding the Predator whenever it is its time to move, breaking ties first based on proximity to the Agent, then at random, and updating the probabilities based on the outcome. After then, it will act in line with Agent 1's rules, assuming the Predator is positioned at the node with the highest likelihood of now doing so (breaking ties first based on proximity to the Agent, then at random). Notably, there is no need to survey any node if it is known exactly where the Prey is.

Implementation:

Agent 5 is a partial information setting actor who does not know the position of the predator but always knows the location of the prey. One key feature of a predator is that it can be distracted; that is, the predator will only follow the target with a probability of 0.6 and will be diverted with a probability of 0.4 at any time step. Agent 5 is also originally aware of the predator's position. Agent 5 will begin gathering information by inspecting the node with the highest chance of where the prey is and deciding where to

travel next based on the information. The same rules for movement as Agent 1 are employed here to base its moves. Agent 5 uses belief vectors to determine the predator's location by extracting the node with the highest probability of finding a predator. If two or more nodes have the same maximum belief, Agent 5 will break ties at random.

Because agent 5 initially knows the location of the predator, the beliefs are initialized to have 1 at the predator's pos and the rest are all zero, before the run (). The agent 5 will then proceed in the same manner as any other partial information environment, surveying the node of maximum belief position and updating the belief based on the information gathered. Just like agent 3, it will update the belief after a survey and after the agent moves. Once the predator has been transported, the transition() function will normalize the belief based on the predator's distraction probability distribution.

3.3.2 Agent 6

Description:

Agent 6 is a partial information setting actor that does not know the position of the predator but always knows the location of the prey. Only Agent 5, the predator, has a 0.6 chance of becoming distracted. Apart from assuming that the point of maximum belief is where the predator is and breaking ties at random, the improvement logic is used in selecting the next node for the agent to travel to. For any other circumstances, the same movement logic as agent 5 is used.

Implementation:

Agent 6 has its own run heuristic () function, which uses heuristics to locate the next best node for the agent to move to, considering the information gathered from the survey and the prey's maximum belief location. Agent 4 will conduct a heuristic on the node position with the highest probability of finding the predator. The average longest distance between agent's next possible node and max belief (preynext)'s possible node is used as the heuristic. The node with the longest distance average is chosen for our agent to migrate to. If two nodes have the identical max beliefs, the same run heuristic is used on these two nodes and the predator's next position to break the tie. This has significantly improved agent 6's performance over agent 5.

3.4 The Combined Partial Information Setting

In this scenario, the Agent may not always be aware of the whereabouts of the Predator or Prey. Every time the Agent moves, it can first select any node in the graph to survey in order to find out who is currently occupying that node. In this scenario, the Agent must monitor the belief states of both the Predator and the Prey, updating them in light of new data and knowledge of the players' actions. The Agent begins in this setting by being aware of where the Predator is.

3.4.1 Agent 7

Description:

If this Agent is not currently certain where the predator is, it will survey in accordance with Agent 5 whenever it is its turn to move. If it is aware of the location of the Predator but not the Prey, it will conduct the survey in accordance with Agent 3. However, this Agent continues to only do one survey per round. The Agent acts by assuming the Prey is at the node with the highest probability of containing the Prey once probabilities are updated in light of the survey's findings (breaking ties at random) and by assuming the Predator is at the node with the highest probability of confining the Predator (breaking ties by proximity to the Agent, then at random). Then, Agent 1's actions are implemented.

Implementation:

Agent 7 is a hybrid of agents 3 and 5, with the agent 7 being unaware of the whereabouts of either the prey or the predator. Agent 7 follows the same regulations as agents 3 and 5. So the agent 7 will begin by knowing the location of the predator and then forget the location of the predator in the next time step. Two belief vectors, one for prey and one for predator, are simultaneously updated in accordance with the rules of agents 3 and 5 and the partial information setting execution order.

Agent 7 will begin the predator's belief vector since, like agent 5, it knows the predator's initial location. Then, before the agent moves, it surveys for the predator and the prey (survey prey(), survey predator()). The same movement rules for agents are used here, with prey and predator positions assumed probabilistically. When there is more than one appropriate node, the ties are broken at random. Once the agent has moved, the beliefs of both actors are modified. When the actors move, their beliefs are updated individually, taking into consideration the movement of the prey and distracted predator.

3.4.2 Agent 8

Description:

Agent 8 is an upgraded variant of Agent 7 and an actor of mixed partial information setup. The same improvement reasoning as in agents 4 and 6 is used here to select the next best node for agent 8 and break ties at random. The sole difference is that survivability is prioritized over catching the prey when breaking bonds, i.e., taking the average longest distance from the predator's estimated location to the agent's neighbors. In all other acts, agent 8 will act like agent 7, following the norms of agents 3 and 5. Run heuristic () is used on both prey and predator to compute the average shortest and longest distances from the agent's next possible nodes to the actor's next possible nodes.

Implementation:

Agent 8 initially knows the predator's location, hence only the belief predator is changed before the run, as opposed to agent 7. Instead of assuming the location of prey and predator based on the location of the maximum expected belief of both prey and predator, Heuristic is used to find the next best location for the agent to move to by applying the same run heuristics for each prey and predator to extract the

average shortest distance and average longest distance. Only the run heuristic for predator is employed when breaking ties, prioritizing survivability over goal-winning danger.

Some questions to consider, when contemplating your own Agents:

- What node should the Agent move towards, given its current information?
- What node should the Agent survey, given its current information?
- How best can the Agent use all the knowledge it has available to it to make a decision?

Answer:

- *Odd agents identify the position of the prey/predator probabilistically, assuming the actor's existence in the node at time (t) and following the rule for agent moves. It does not consider how the prey/predator might behave in (t+1). And directly selects the next node for the agent in order to avoid/catch the actor while keeping the actor in the position of maximum belief. Even agents consider the movement of the prey/predator when determining the motion. This improves the efficiency of the even agent's movement and raises its chances of survival.*
- *All agents survey the node with the highest level of belief. Based on the survey results, this belief is updated when initialized before run (), after the agent's move, and after the prey/ predator's move.*
- *The prey/predator is assumed to be in the highest belief by odd agents, while even agents perform additional heuristics from agent's neighbor to actor's neighbor, determining the average shortest and longest distance to discover the ideal node all the time. When there is more than one optimal node, the same heuristic and movements are used to break ties.*

3.5 Agent 9

Agent is thought to operate in the same combined partial information environment as Agents 7 and 8. Agent 9 can be bootstrapped with agent 8 to deal with the random tie breaker and select the best node.
Problem with a faulty drone:
If survey () fails to locate an actor,
Normally, P (finding an Actor at X/survey result X) = 0.
However, n defective drone P (finding an Actor at X/survey result X) = 0.1 P (Actor at A/False survey result) = 0.1 * the belief that the actor was at the surveyed node divided by the new sum of beliefs

Improvement Logic:

To deal with the malfunctioning drone, Agent 9 can be programmed to act only when the survey results are successful. The belief system would, in principle, stay constant.
Because when a faulty drone surveys a node and certifies the presence of an actor, the information is 100% accurate. In the false negative condition, however, the malfunctioning drone returns false, which equals 0.1*P (actor's presence).

P (Finding an actor at X| survey result at X is failure) = 1 for some nodes.

The false negative condition significantly reduced agent 7's chances of survival; removing this false negative from the belief system would greatly improve his chances of survival. In circumstances where both the prey's and predator's surveys are false, we can make the agent operate as agent 7/8.

As a result, the agent relies heavily on
P (finding an actor at A) =
P (Finding an actor at A) *P (finding an actor in A| Actor in A) = P (Finding an actor at A)*0.9.

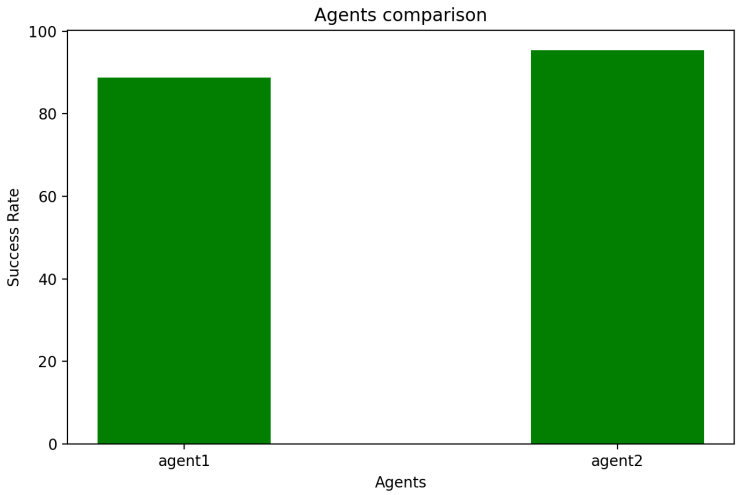
4 Analysis

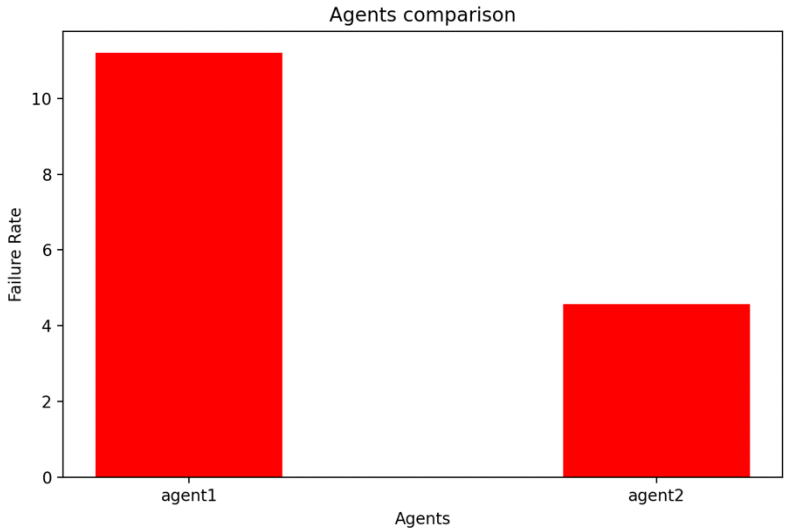
```
testagent  :
success_rate: 67.45,
variance: 84.41416666666663,
std deviation : 9.187718251375944
```

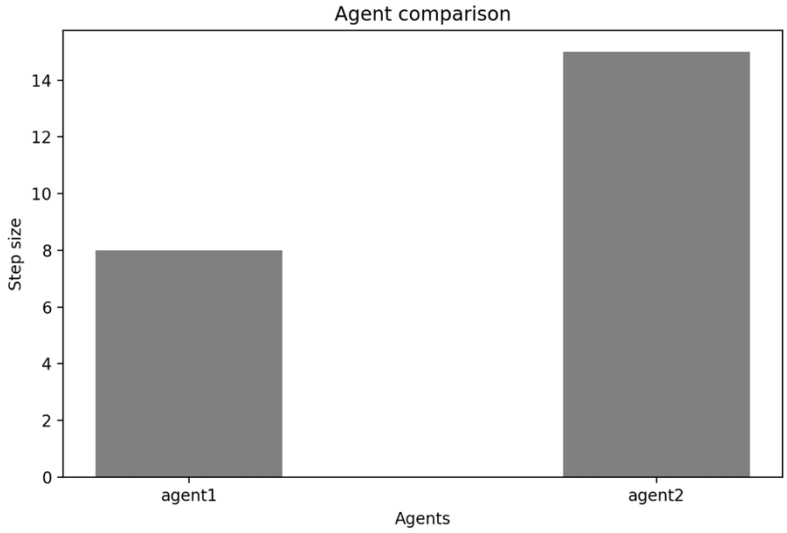
4.1 The Complete Information Setting

```
agent1  :
success_rate: 88.79333333333331,
variance: 36.11951111111111,
std deviation : 6.009951007380269

agent2  :
success_rate: 92.43333333333332,
variance: 13.047777777777785,
std deviation : 3.6121707846913584
```



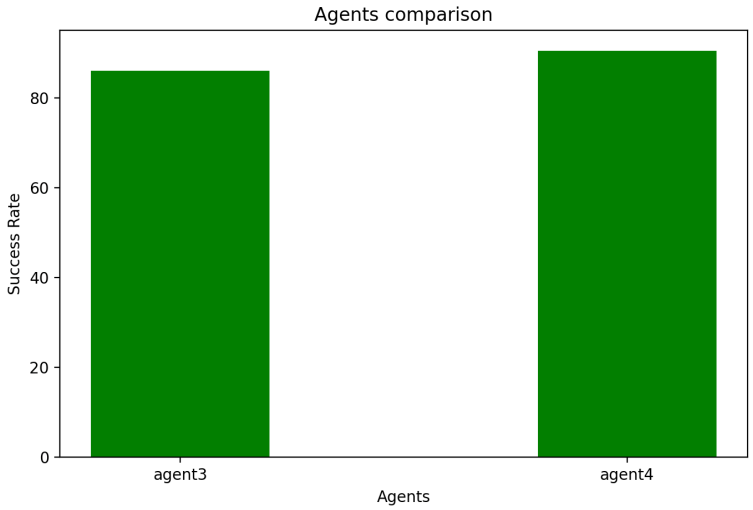


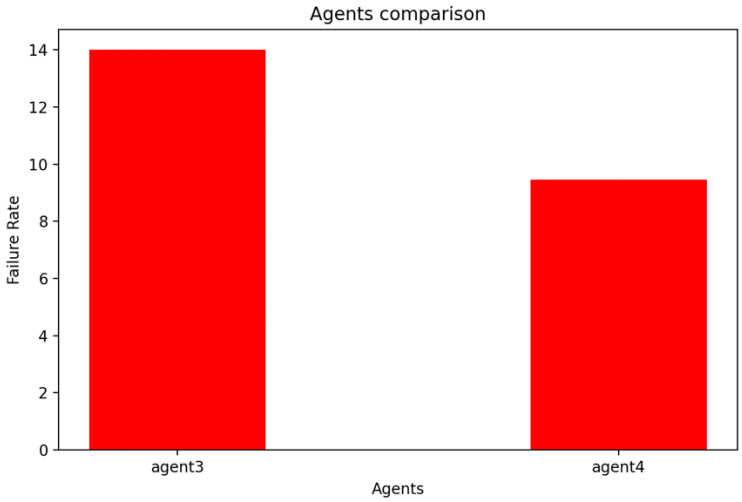


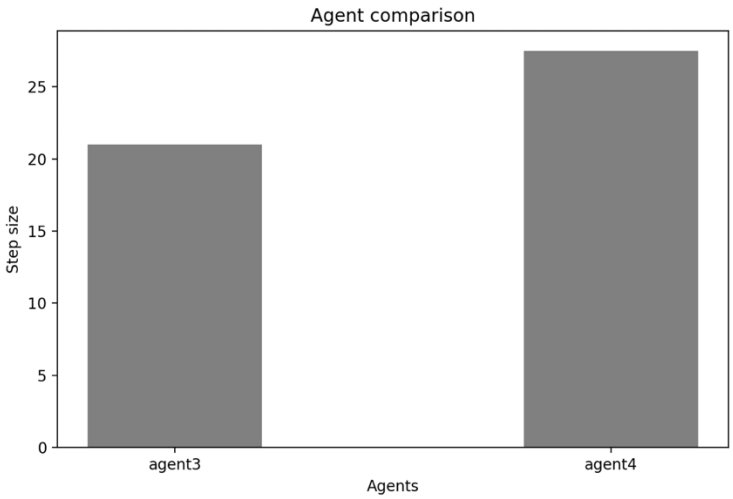
4.2 The Partial Prey Information Setting

```
agent3  :
success_rate: 85.98902000000001,
variance: 37.47183613737776,
std deviation : 6.121424355276945

agent4  :
success_rate: 88.53333333333335,
variance: 11.048888888888886,
std deviation : 3.3239868966181088
```


Agents comparison
Success Rate
0
20
40
60
80
agent3
agent4
Agents

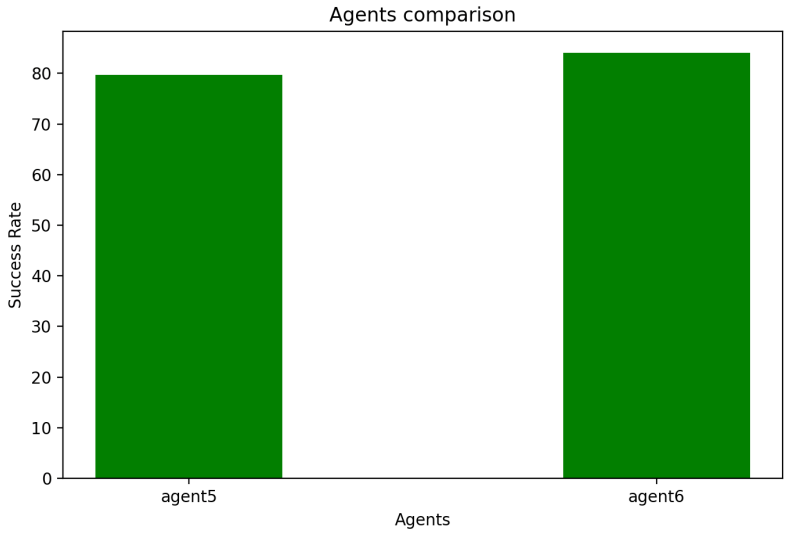

Agents comparison
Failure Rate
0
2
4
6
8
10
12
14
agent3
agent4
Agents

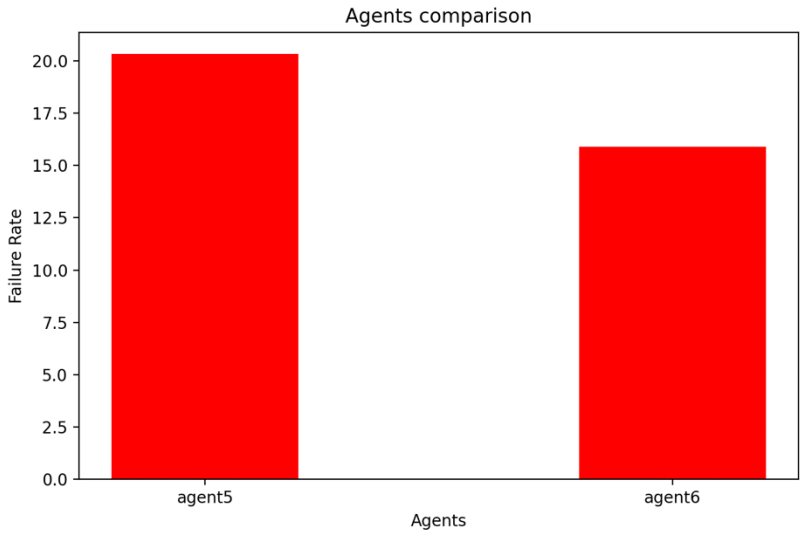

Agent comparison
Step size
0
5
10
15
20
25
agent3
agent4
Agents

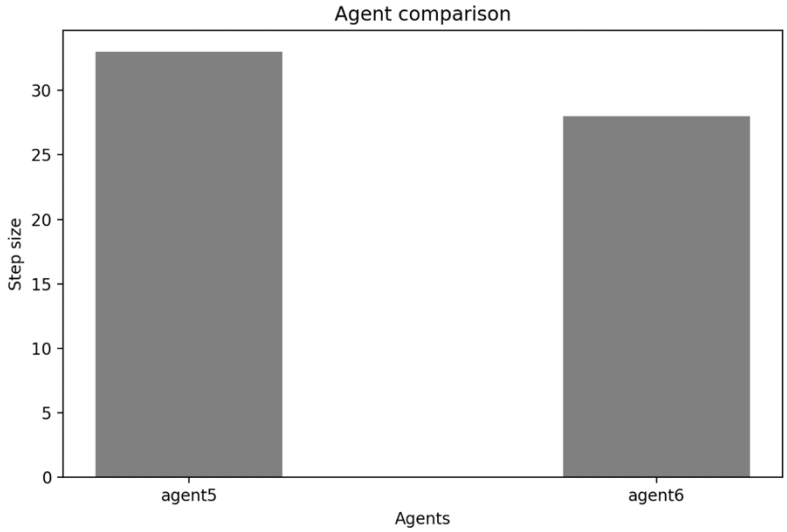
4.3 The Partial Predator Information Setting

```
agent5  :
success_rate: 79.65333333333335,
variance: 94.39537777777792,
std deviation : 9.715728370934313

agent6  :
success_rate: 82.10666666666658,
variance: 63.92640000000003,
std deviation : 7.9953986767390175
```



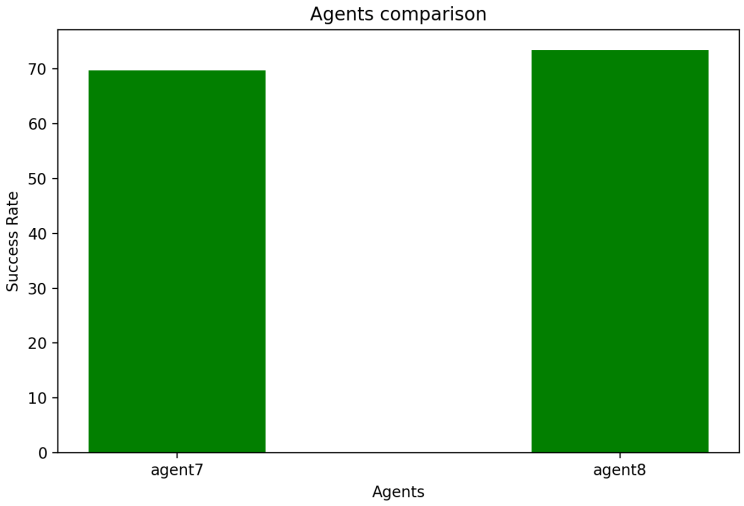


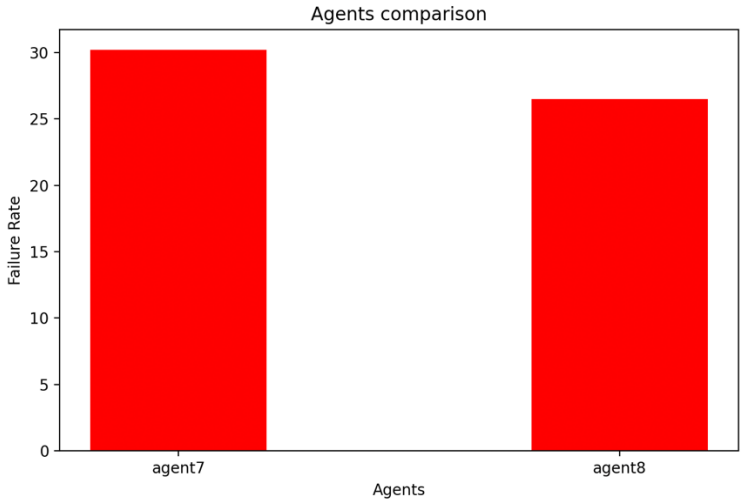


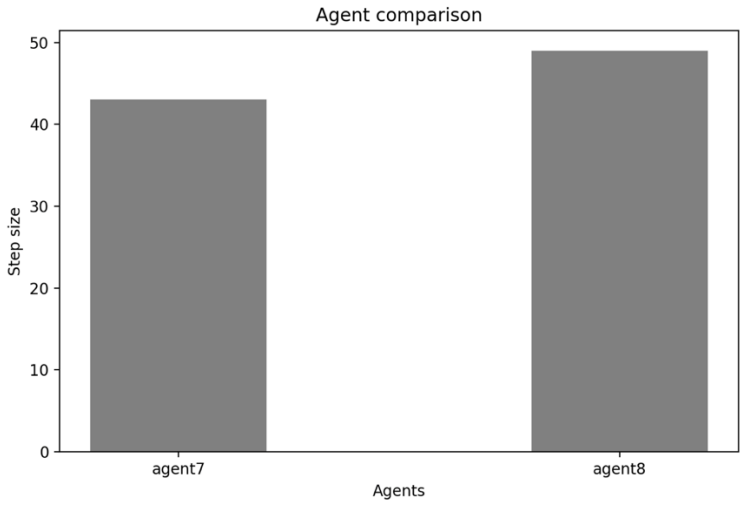
4.4 The Combined Partial Information Setting

```
agent7  :
success_rate: 72.77999999999994,
variance: 34.59159999999999,
std deviation : 5.881462403178311

agent8  :
success_rate: 75.48666666666668,
variance: 44.28760000000009,
std deviation : 6.654892936779682
```


Agents comparison
Success Rate
0
10
20
30
40
50
60
70
agent7
agent8
Agents


Agents comparison
Failure Rate
0
5
10
15
20
25
30
agent7
agent8
Agents


Agent comparison
Step size
0
10
20
30
40
50
agent7
agent8
Agents

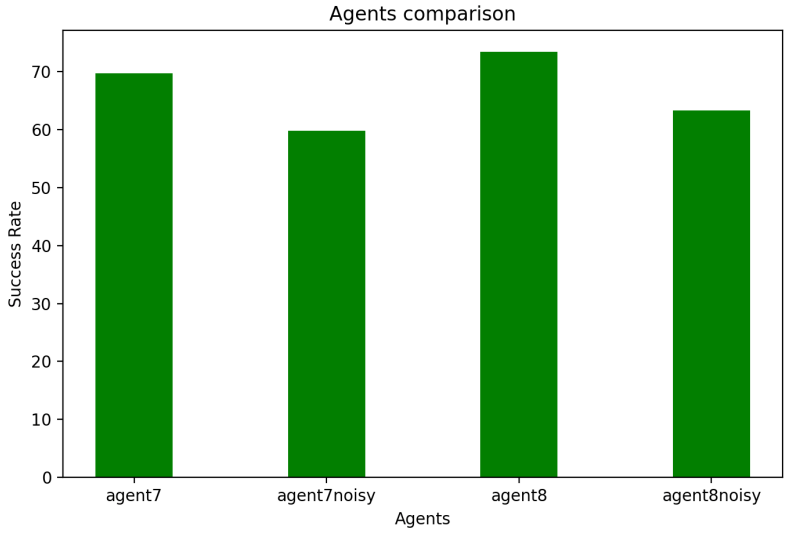
4.4.1 Defective Drone

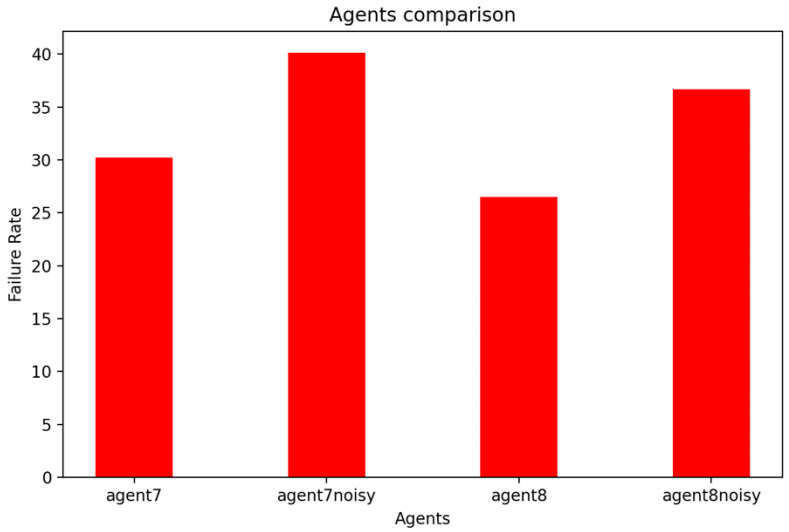
- How do Agents 7 and 8 compare in this setting, if you do not update your belief update rules to account for this?
- How should you update your belief update rules to account for this?
- How do Agents 7 and 8 compare in this setting, once belief update rules have been updated to account for this?
- Can you build an Agent 9 to do better?

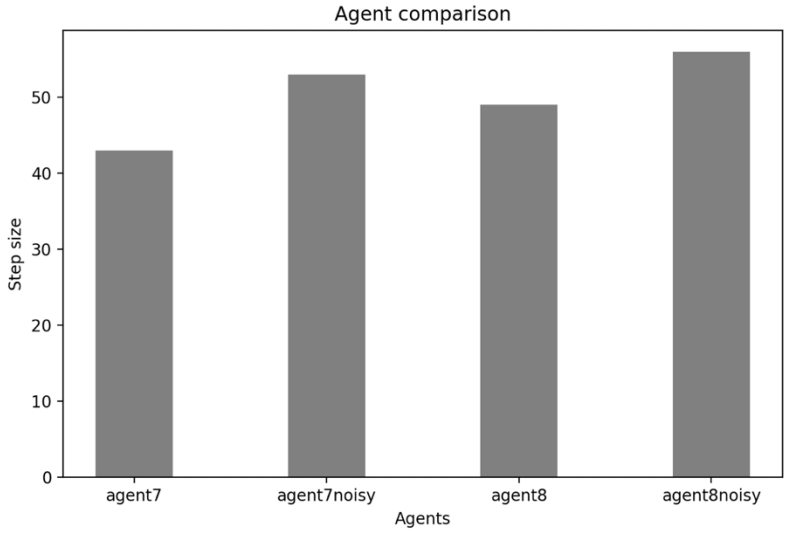
Insights:

This establishes the combined partial information settings as agents 7 and 8. The defective drone describes a survey that returns a false negative of 0.1 probability even when an actor is present. The survey's result is 0.9 probability correct and 0.1 error correct. Agent7 noisy.py and Agent 8 noisy.py are mentioned in the description.

If the survey () fails to identify an actor, the P (finding an Actor at X/result of survey X) is normally equal to zero.
However, n faulty drone P (finding an Actor at X/survey result X) = 0.1 P (Actor at A/False survey result) = 0.1 * the belief that the actor was at the surveyed node divided by the updated sum of beliefs



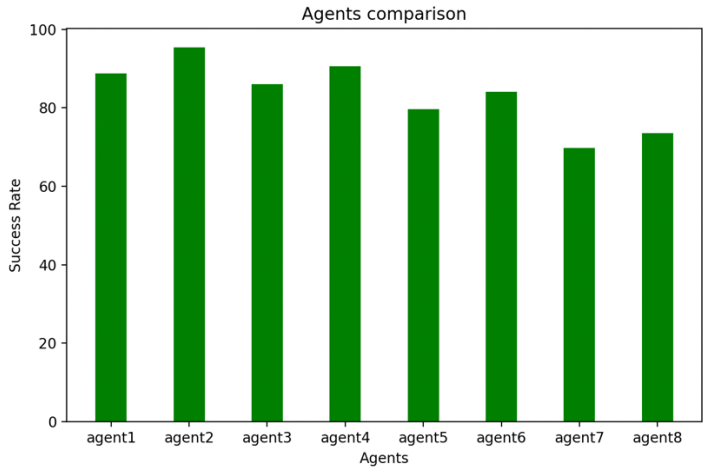


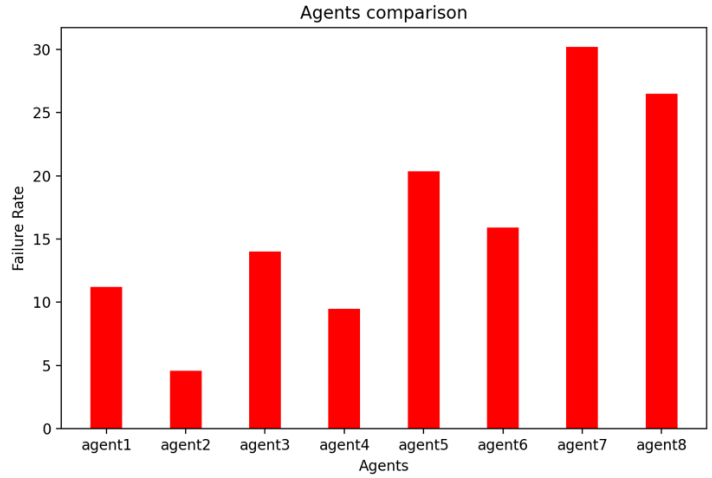


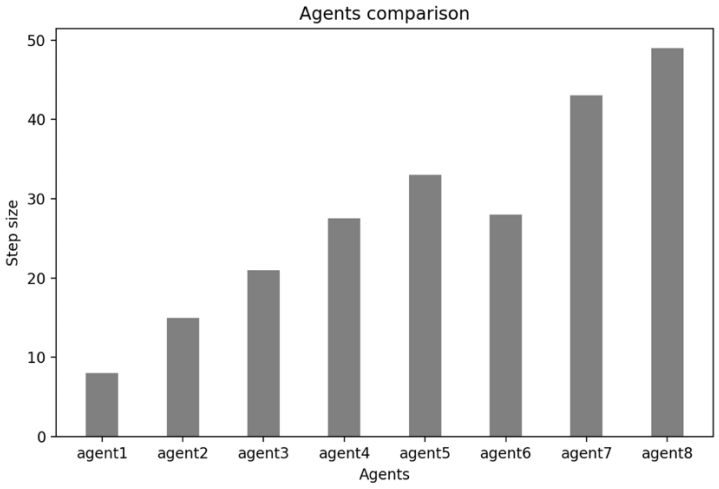

```
agent7noisy  :
success_rate: 59.85333333333331,
variance: 129.19848888888893,
std deviation : 11.366551319062829

agent8noisy  :
success_rate: 63.29333333333331,
variance: 159.22728888888875,
std deviation : 12.618529585054224
```

4.5 All Settings' Comparison:







4.6 All Settings' Graph Analysis:

Complete information setup:

Because agents 1 and 2 are aware of the whereabouts of prey and predator, the survivorship is naturally high. (Case: 1) When both the agent and the predator share the same neighbor, Agent 1 suffers. (Case: 2) In circumstances where the prey and predator are in the same position, the agent will attempt to go for the prey while ignoring the predator, causing the agent to perish in vain. Agent 2 outperforms agent 1 by 5% by using heuristics to select the node and break ties.

Partial info setup:

Agents 3 and 4 are unaware of the position of the prey, but they both probabilistically presume it and move appropriately. Because both are aware of the predator's location, it is simple to avoid the victim. Agent 3 fails for the same reason as agent 1, and agent 4 is developed, which uses heuristics to avoid cases 1 and 2. Agent 4 almost always provides a greater survival rate. Agent 4 outperforms agents 1 and 3 on average.

Agents 5, 6, 7, and 8 have a lower survival percentage than Agents 1–4 since they are unaware of the predator, yet the predator always draws closer to the agent despite the diversion. The target can still be pursued by the distracted predator. This requires the agent to constantly monitor the predator, even if the animal's location is frequently surveyed true. However, given our habitat, practically all nodes can be reached in 5 timesteps, and knowing the location of prey assists 5 and 6 in catching the prey and has a greater survival probability than 7 and 8.

Agents 7 and 8 are unaware of each other's whereabouts. When compared to the other agents, Agent 7 performs the worst. The combined heuristic in agent 8 significantly improves agent 8's performance and keeps it alive by 4-5%.

Odd agents can locate prey around 10-15% of the time. Even agents can locate the target around 15-20% of the time. Almost 5% of the time, unusual agents can locate the predator. Agents can only locate the predator 5-10% of the time.

Simulation:

The count was set to 5000, but it was never reached; instead, one of the two goals was always met. This is included to address the edge case of an agent scurrying across the environment without getting caught or catching the prey. The likelihood of hanging was significantly reduced when the count was set to 500, but at 300, we could see the hanging 10% of the time.

4.7 Bonus

Bonus: In the Partial Information Settings - suppose that whenever it is the Agent's turn, the Agent must make a choice to move or to survey, instead of doing both. How should the Agent decide which to do, and once the decision is made, how should the Agent decide what to do (move to take or node to survey)? Implement this and compare its performance to the above.

Answer: Bonus can be viewed as the bootstrapped version of Agents 4 through 6 in the incomplete information setting. With all the heuristics of the even agents and breaking ties by applying the same heuristics to the agent's neighbor pos to the actor's neighbor pos. Furthermore, the bonus agent may survey only if the number of max belief count is larger than or equal to node count/2. If the max belied count is less than the node count/2, it can be more certain of the prey's location and can advance towards it. However, all these conditions apply only when the agent is placed far enough away from the predator. However, the simulation shows that the average survival rate for the bonus agent is almost identical to that of the test agent, which is 65%.