Project 2: Circle of Life

Course: 16:198:520

Instructor: Professor Cowan

Michael Neustater (mdn68) and Vijay Souri Maddila (vm633)

Purpose:

This project consists of a 50-node, graph-based environment, where an agent is pursuing a prey, while simultaneously being pursued by a predator object. In this environment, the agent will not always be able to see the position of the prey and predator. The purpose of this project is to build a probabilistic model, which allows the agent to make informed decisions about the next possible action, based on the position of the actors. These scenarios are first implemented by a pre-determined rule-based approach, which we attempt to improve upon with a utility approach.

Introduction

This decision-based catching and avoiding game is deployed over four information settings:

1. The complete information setting, where the agent knows the exact position of the prey and predator at every timestep in the environment.

2. The partial prey information setting, where the agent always knows the exact position of the predator but does not have information about the prey. To overcome this lack of information, the agent can look at any one of the nodes in the environment via a survey and see if the prey exists at that node, it will survey the highest belief node breaking ties at random.

3. The partial predator information setting, where the agent always knows the exact location of the prey but only has information about the starting location of the predator. Like the above setting, the agent can survey any node in the environment to know if the predator exists at that node, it will survey the node of highest belief breaking ties by proximity then random.

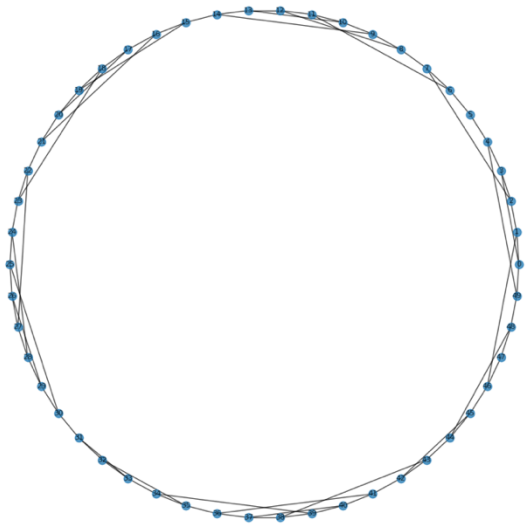
4. The combined partial information setting, where the agent does not know the location of the prey and only knows the starting location of the predator. Like the third and second information settings, the agent can survey any node in the environment, but will only be able to survey based on the belief of prey location or predator location. The agent will choose to survey based on the node of highest predator belief, unless it is certain of the predator's position, then it will survey for the node of the highest prey belief.

To implement this project, we use the following technologies:

1. Programming Language: Python
2. Mathematical Equations handling package: NumPy
3. Visualizations: 'Matplotlib'
4. Animations: 'PyGame'

The Environment

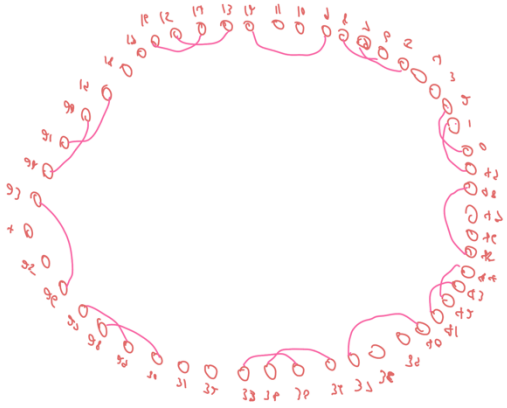
The environment for this project is a graph of fifty nodes that have edges connecting them such that it forms a circle. Additionally, this graph can have additional edges that can connect a node with another node within five backward or forward steps. The backward or forward step refers to a traversal along the immediate connected edge to another node in the original circle.



The edges conform to the constraints that the maximum degree of a node is three, a node cannot have an edge with itself, and two edges can exist between two nodes (edges are a set). These edges are generated by randomly picking a node and finding all edges that meet these criteria and randomly picking one of those edges.

Min and Max Additional Edges

Given our implementation of the graph generation, the maximum number of additional edges possible is 25. In order to explore the minimum number of edges, we generated 100 million graphs according to the rules we specified. The result of which was that 24 edge graphs were most common and the smallest generated number of edges was 19 with only 5 occurrences. We were able to draw a scenario by hand (shown right) which conformed to the above rules and included 18 edges, but while fewer than 19 edges are possible, they are extremely unlikely to occur and from this it can be concluded that any graph of less than 19 edges has an essentially zero probability of occurring.



18 Edge Graph

```
Additional Edges:  19 Occurances  5
Additional Edges:  20 Occurances  2624
Additional Edges:  21 Occurances  297533
Additional Edges:  22 Occurances  7450947
Additional Edges:  23 Occurances  42440148
Additional Edges:  24 Occurances  45581834
Additional Edges:  25 Occurances  4226909
```

Implementation of Environment

The graph space is implemented using a series of node objects which combine to make an environment.

1. **The node class:**

   This class defines the structure of the node objects that will be used throughout the project to get all the information about a node.

   A node object contains the following information:

   - The index of the node
   - The index of the node to its left (circular left).
   - The index of the node to its right (circular right)
   - The index of the node that is connected via the additional edge. This index will be the index of the current node to ensure that NULL data is not carried on to the final environment.
   - The degree of the node, which signifies the number of edges the current node has.

2. **The environment class:**

   The environment class takes an input of the number of nodes but has a default value of fifty. Based on this number, a list of nodes (node class objects) is initialized in a way that all the elements form a circle. Then additional edges are created randomly for nodes that have a degree of less than three. For simplicity, nodes are stored in a list of length 50.

   Due to the frequent need to find the shortest path between two nodes and the fact that the same environment is used for multiple simulations, in order to reduce the need for repeated searches, the shortest path between all possible nodes is calculated at the environment creation

time. These shortest paths are stored in a 2D list of length 50 by 50, such that index i,j returns the integer value distance (number of nodes traversed) from node #i to node #j. This is created using 50 full depth BFS searches, where the ith search returns the ith row. The actual path is not stored since multiple shortest paths may occur, but shortest paths can be reconstructed using this list by advancing to the next shortest possible node.

A prey transition matrix is also precomputed in the environment class. This is a fifty-by-fifty matrix that contains all the possible prey moves from one node to the other. This transition matrix will be discussed deeply in the following sections. Since the prey moves uniformly at random the transition probability is equal for all edges of a node.

Similarly, a distracted predator's transition matrix of size fifty-by-fifty is also created in the environment class which helps in calculating the next move of a distracted predator, similarly because this transition matrix is used to consider a move uniformly at random all transition probabilities for a node are equal. The transition matrixes will be discussed later in the belief subsection of Agent.

The Agent

The Agent's goal is to catch the prey while avoiding the predator. For all partial knowledge variations, the agent keeps track of the other unknown actors using belief vectors. The agent is guaranteed to spawn in a non-game-ending position (prey or predator positions).

Implementation

In this project, each agent is implemented as separate class files that contain various functions depending on the information setting, they are based on and the belief strategies they are using.

All the Agents use the following common function:

Move (): a logical procedure that begins the Agent simulation. It consists of a loop where each pass of the loop is one step (also referred to as timestamp). In the loop the agent will survey, if possible, then move into the neighboring node that is most suitable for winning the game, while simultaneously updating the positions of the prey and predator. After each update of the actors' positions, this function also performs a check to see if any of the two outcomes have been achieved (death or capture).

For the partial-belief settings, the unknown agents are kept track of using belief vectors. The prey is accounted for using the *prey_probability_array* and the predator using *predator_probabiility_array*. Each array is of length 50, with each index representing a node in the environment and the value of that node is the current belief that the actor exists there. The sum of all indexes must be equal to 1 (or 0 in the defective scenario). Each belief update has an associated function to carry out the update.

Prior to moving, the agent must survey one node (chosen according to the setting rules), where it can see if an actor is present in that node. It then can use this information to update the belief vectors according to the rules of conditional probability. When the agent moves, if the game continues, it is also able to update the beliefs based of these same rules:

P(A is at M | Result Survey N)

=P(A is at M, Result Survey N)/P(Result Survey N)

*=P(A is at M) * P(Result Survey N | A is at M) / P(Result Survey N)*

*=P(A is at M) * P(Result Survey N | A is at M) / Sum of Probabilities*

*=P(A is at M) * P(Result Survey N | A is at M) / 1 - removed probability*

There are two functions that facilitate the above belief vector generation and manipulation:

- **survey ()** - For a successful survey of actor A at N, value at the index for N becomes 1 and all other indices become 0. For a failed survey of actor A at N, the value at the index for N becomes 0. The remaining indices must be normalized by dividing their current probability by the new sum of the belief vector (probability of the actor not being at that surveyed node), this function also returns the next highest belief of the actor(s) (depending on info setting), including tie breakers.
- **agent_moved()** - The logic for survey is extended to the second belief update that occurs when the agent moves to a new node. Each move the agent makes to a new node that does not end the game is equivalent to an unsuccessful survey of that new node, meaning that its probability is 0 and the existing beliefs will be normalized by the new sum.

The Agent understands the rules that the other actors move by and therefore is able to update its belief vectors after each movement of the actors:

P(A is at M after move)

= P(A is in M now) P(Prey in M Next | P in M now) +

P(A is in N now) P(Prey in M Next | P in N now) +

P(A is in Z now) P(Prey in M Next | P in Z now) ...

- **transition ()** - Gets the dot product of the current belief vector by a transition matrix or two in the case of the distracted predator. The transition matrix is a 50 by 50 array, where the columns represent a starting node, and the rows represent the edges to the connected nodes. The value in each cell of the matrix is the probability that the actor would travel from its current node to that connected node. Since the actors move according to different rules, their updates use different matrices. This update is only called after prey and predator move, meaning that the game has continued, so the probability an actor shares a node with the agent is 0. meaning the index of the agent is set to 0 and the belief arrays are normalized by dividing by their new sum.

  prey_probabiility_array:

  - Initialization: upon the creation of the agent, the array initializes to 1/49, since there is an equal probability that the prey is contained by any node except the initial agent node.

- Transition Matrix: Since the prey moves uniformly at random with the choice of its current node or one of 3 possible connected nodes, the transition matrix can be created at the environment creation and is a class variable of the environment. The matrix is created by zeroing out a 50 by 50 matrix, then for each node N, the edge indexes corresponding to connected nodes and itself are set to 1/(degree of N + 1). (The value is (degree of N + 1) since the edge to itself symbolizes staying still and does not actually exist, therefore it is not included in the degree).

predator_probability_array:

- Initialization: Since the agent knows the initial state of the predator, the belief vector starts with a value of 1 at the current predator position and the rest of the array is initialized to 0.
- Transition Matrix: Since the partial belief predators are "distracted" they must transition according to two transition matrices, which will be referred to as focused and distracted. The belief after transition is found by creating two copies of the belief vector. The focused vector is the dot product of one copy by the focused matrix and the distracted is the dot product of the other copy by the distracted matrix. Since there is a 0.6 probability of the predator being focused, each index of the focused vector is multiplied by 0.6 and since there is 0.4 probability of the predator being distracted, the distracted vector is multiplied by 0.4. The two vectors are then summed up by index ($i_{sumarray} = i_{focused} + i_{distracted}$)
    - Focused Matrix: The focused matrix is created by creating by finding the next possible positions of the focused predator, which are the closest nodes to the agent from a potential predator containing node, if there is more than one shortest next node for a potential predator containing node, all nodes of the same shortest path length are considered. These edges are given the probability 1/number of next possible nodes. The rest of the matrix is set to 0 (edges of nodes the predator has 0 probability of being in).
    - Distracted Matrix: The distracted matrix is also be generated at environment creation time and is a class variable of the environment since the predator will only move randomly to any of the possible edges (but not stay still). So, for each node in the environment, the probability values for its edges are 1/degree of node.

The logic and implementation of the defective updates will be discussed in a separate section in the later part of the report.

The Prey

Prey is the actor in the project that defines the success of a simulation. The prey is a goal that keeps changing its position in the environment. Prey itself has no goal or condition to satisfy, and thus moves

randomly according to a random function into any one of its neighbors or just stays in the same place. So, the probability of the prey moving into any of the possible nodes (neighboring nodes + its own node) is uniform and will depend on the number of choices it has (degree of node + 1).

Implementation

The Actor Prey is implemented as a class that only has two functions:

1. **Initialization (__init__):** This function initializes the class variable 'pos' to a random node index in an environment.
2. **Move ():** The move function takes the current environment and agent position as inputs and updates the prey position by moving uniformly at random among the options of its neighbors and itself. This function also checks for collision. If the prey moved into the node with agent, then it returns False or else it returns True.

The Predator

Predator is the actor in the project that defines the failure of a simulation. The predator is a failure node that keeps changing its position in the environment and has the goal of catching the Agent. It always has the information about the Agent's position. In the complete information setting and the partial prey information setting, the predator directly runs towards the Agent, but in the settings after these, the predator is distracted and will only move towards the Agent's position with a probability of 0.6 for any given timestep and moves to any of the neighboring nodes with the remaining probability of 0.4. This makes it hard to keep track of the predator location even though its initial position is known in the partial predator information setting and combined partial information setting.

Implementation

The predator is implemented as a class that has three functions:

3. **Initialization (__init__):** This function initializes the class variable 'pos' to a random node index in an environment.
4. **Move ():** The move function takes the current environment and agent position as inputs and updates the predator position based on the shortest distance to the agent position. This position update is moving to one of the neighboring nodes that has the shortest distance to the agent's position, if there is more than one possible shortest distance node, the predator will pick between them uniformly at random. Like the other move functions, this function also checks for collision. If the predator moved into the node with agent, then it returns False or else it returns True.
5. **Move_distractable ():** This function is like the above function with a slight modification. With a probability of 0.6, the predator will function the same as regular move. With the remaining probability of 0.4, it will choose to move uniformly at random to one of the adjacent nodes. Once it has moved to a new node, it will check if it has collided with the agent, if it has it returns False else True.

Project Design

Testing:

All testing consists of 100 runs on 30 different environments (for this case the environment referred to is only the collection of edges and nodes not actor positions), for a total of 3000 test iterations. For some run R in environment E, all agent variations will be tested using the same initial starting environment. This means for each R in E, all agents will start in the same initial position, with the same starting position for prey and predator. After this initial state, the movement will deviate due to randomness driving the prey movement and the dependency of predator movement on the agent position.

Steps:

A step is a loop inside of the move function for all Agents (steps start at 1 not 0), for the complete information settings, this consists of the agent move, followed by a check for collision with predator, then prey, then prey move, then predator move. For all incomplete information settings, a step consists of a survey prior to agent move.

End Game States:

For all agents, after the agent moves, a collision with the predator is checked, then prey, meaning if the agent moves into a node containing the predator it will always die even if the prey is also present. If only the prey is present in the new node, the game will end as a win. Once these collisions are checked the prey is allowed to move and a collision is checked for if the prey has moved into the agent's position (ending immediately if this occurs), then the predator is allowed to move, and a collision is checked for a death.

An additional end game state is timeout, which occurs if the agent has not caught the prey or been killed by the predator in 5000 steps. This end state is to prevent edge cases in which the agent never dies (wiggles between states where it attempts to avoid death but cannot confidently approach the prey). While our agents did not encounter this issue and therefore never timeout, it was still included for thoroughness.

Animation:

The code features the option to animate an instance of a setting by running **animation_main.py.** This is done using pygame and runs through each agent sequentially.

https://drive.google.com/drive/folders/19hlHVPGoX66Lz-b9HjVGQN83FZTeJZjs?usp=sharing

Complete Information Setting

Agent 1

Agent 1 is always aware of both actors' positions and bases its movement off these set conditions, where a lower number condition is prioritized.

1. Choose neighbors that are closer to the Prey and farther from the Predator.
2. Choose neighbors that are closer to the Prey and not closer to the Predator.
3. Choose neighbors that are not farther from the Prey and farther from the Predator.
4. Choose neighbors that are not farther from the Prey and not closer to the Predator.

5. Choose neighbors that are farther from the Predator.
6. Choose neighbors that are not close to the Predator.

When there is more than one neighbor that satisfies the highest possible condition, then the agent must choose an optimal one among them. These conditions are implemented in:

Get_optimal_Node (): which returns the optimal neighbor to move into based on these rules. If there is more than one next node that satisfies a condition, a tiebreak is created by again comparing the eight optimality conditions in accordance with the prey and predator distances.

Governing logic

The 'get' function inside 'get_optimal_node.py' gives the most optimal node from the available nodes by iterating over all the available nodes and checking their distances to prey and predator. Within each of the six conditions outlined above, if another neighboring node has the same priority, then the same six conditions are used to break ties between the new node and the already found node. The most optimal node among them is stored in a new 'optimal_node' variable, and after iterating over all the neighboring node, this 'optimal_node' is assigned to the new position of Agent 1.

The Agent moves towards the prey while distancing itself from the predator. In the cases where such a move is not possible, the Agent prefers to move closer to the prey while not being concerned about moving away from the predator.

Results

```
Agent 1:
Caught (including timeout): 90.3% | Died (including timeout): 9.7% | Timed Out %: 0.0%
Caught (excluding timeout): 90.3% | Died (exlcuding) timeout): 9.7% | Avg Steps: 12.481666666666667
```

Agent 2

Agent 2 is an actor in the complete information setting that attempts to assess the utility of each next possible move it can take by considering the combined positive utilities of moving towards the prey and negative utility of moving towards the predator and takes the move of highest combined utility. The logic for determining belief is Bellman inspired and will be further touched upon in the improvement logic section.

Improvement logic

The bellman equation can be defined as $V(s) = max\, a(R(s,a,s') + \gamma \sum_{s'} P(s,a,s') + V(s')$, where γ denotes some decay value, R denotes the reward of taking action a to go from s to s' and P(s,a,s') is the probability of traveling from s by action a to s'. The agent considers the prey to have a reward of 1 and the predator to have a reward of -1 and uses a constant decay factor of 0.9. For simplicity, these values are used to create two separate utility arrays, one for the reward of going near the prey and one for the penalty of going towards the predator. Initially the utility array is set to 0 except for the position of the prey at node N which is 1. According to the bellman equation, the utility this provides to all adjacent nodes is max of the set {0, 0.9 * (1)}. Furthermore, for all nodes adjacent to those nodes are the utility provided is max of the set {nodes current values, 0.9 * 0.9} and so on. Since an agent can only pick

adjacent nodes or itself as an action, the calculations for utility are only considered for adjacent cells. Furthermore, nodes part of the shortest path will always have the highest utility, since they will be multiplied by the decay value fewer times. Therefore, the utility of each adjacent node can be easily simplified as $0.9^{distance\ from\ prey\ to\ action\ node}$. Such that 0, results in a direct reward of 1, and each increase in distance has a decay of 0.9. This same logic can be used for the predator however these utilities are detrimental (negative values). The predator utility for each action is then subtracted from the prey utility so that a new combined utility is created. Whichever combined utility has the largest value is the next action chosen by the agent. The resulting utilities from these calculations are only an approximation since they create the decay based of the assumption the actors stay still and do not consider the potential of the rewards moving at each timestep. This method is much faster and since the utility is calculated so frequently (which becomes more important in partial information settings). This approximation is still very effective since it provides a fairly accurate utility for actors far from the agent which increases in accuracy when they are closer to the agent (since they have less potential to move from their current position in the few steps the utility is calculated). Ultimately accurate utility is most important when the reward (prey) and penalty (predator) are nearby since these actions have a higher chance of ending the game.

Results

```
Agent 2:
Caught (including timeout): 97.467% | Died (including timeout): 2.533% | Timed Out %: 0.0%
Caught (excluding timeout): 97.467% | Died (exlcuding) timeout): 2.533% | Avg Steps: 10.015666666666666
```

Partial Prey Information setting

Agent 3

Agent 3 is an actor set in the partial prey information setting, where the Agent is always aware of the predator's position but does not have any information about the prey's location. This agent can survey any node in the environment for a timestamp and then move according to the new information. Agent 3 makes use of a belief vector (described in the product design section), to keep track of the probability that the prey is at a certain node. The agent surveys the node with the highest expected belief of prey's position (breaking ties at random) and updates the belief vector based on the survey result. It then attempts to move towards the node with the current highest belief, using the **Get_optimal_Node()** function from Agent 1.

Governing logic

The main logic that governs Agent 3 is the prediction of Prey. Initially, the belief of prey at each node is initialized to 1/49 since, there are 49 nodes after removing the node where the Agent is present. For each timestamp, the agent will call **survey(),** which will survey the node of the highest belief (breaking ties at random) and update beliefs based on the probability rules mentioned previously. It will then attempt to move using agent 1's logic to the new node of highest belief and call the **agent_moved()** function, which updates the belief vector to 0 for the new position of the agent and normalizes the probability. If the game has not ended by a collision with an actor it will call the **transition()** function, which updates the belief vector based on the next possible movements of the prey.

Results

```
Agent 3:
Caught (including timeout): 87.233% | Died (including timeout): 12.767% | Timed Out %: 0.0%
Caught (excluding timeout): 87.233% | Died (exlcuding) timeout): 12.767% | Avg Steps: 21.837333333333333
Steps where certain of prey pos (and correct): 6.185%
```

Agent 4

Agent 4 abides by the same rules as Agent 3 and follows the same belief logic to keep track of the prey. Instead of using **Get_optimal_Node(),** Agent 4 builds upon the ideas from Agent 2 but adapts them for the partial information setting of the prey.

Improvement logic

Since the exact value of the prey is unknown, for each possible next action, the agent finds the hypothetical utility of the prey from each position in the environment, using $0.9^{distance\ from\ node\ to\ action\ node}$ then multiplies it by the belief vector value that the prey is in that node. The agent multiplies by the belief in order to weight the utility the prey would give by being in that node, such that highly probable nodes have a much larger influence on utility. These utilities are then combined to create a composite prey utility for a given next action node. Since the predator locations are known, the predator utility is calculated in the same way as agent 2 and is subtracted from the new composite prey utility for each next possible action node. The next action node with the highest utility is chosen as the next agent position.

Results

```
Agent 4:
Caught (including timeout): 98.533% | Died (including timeout): 1.467% | Timed Out %: 0.0%
Caught (excluding timeout): 98.533% | Died (exlcuding) timeout): 1.467% | Avg Steps: 22.561333333333334
Steps where certain of prey pos (and correct): 5.953%
```

Partial Predator Information setting

Agent 5

Agent 5 is set in the partial predator information setting, where it has information about the position of prey at time step in the simulation but only has information about the predator's initial position. Like the previous partial information setting, the Agent can survey a node in the environment and get information. Using this information, the Agent can maintain and update a belief vector for the predator's position. Like the previous odd Agents, it uses the **get_optimal_node()** function, breaking ties off proximity.

Governing logic

The approach for this Agent is like the previous partial information setting agents. The survey function and transition update functions differ a bit due to the logic that governs the predator's movement.

The survey function does not just take the node with highest predator belief, but also considers how close the chosen node is to the current position of the agent. If there are multiple nodes with the same

distance to the agent and belief value, then the ties are broken at random. Once the choice for which node to survey is made, the same logic of survey and belief update for *predator_probability_array* discussed in project design applies here.

The next highest belief, closest predator position is then assigned to the predator position variable, and from there Agent 1's logic takes over till the Agent's new position is assigned. Immediately after assigning the new Agent's position, the predator belief is updated according to the **Agent_moved()** function to account for the latest information obtained after moving the agent.

Then checks for the end game states are performed while simultaneously updating the positions of prey and predator. Finally, the transition matrix for the predator must be updated for each time step since the predator moves according to the agent's position. The logic behind the two transition matrices is outlined in the project design section.

Results

```
Agent 5:
Caught (including timeout): 83.533% | Died (including timeout): 16.467% | Timed Out %: 0.0%
Caught (excluding timeout): 83.533% | Died (exlcuding) timeout): 16.467% | Avg Steps: 17.441
Steps where certain of predator pos (and correct): 59.593%
```

Agent 6

Agent 6 functions according to the same rules and belief updates as Agent 5, but like agent 4, agent 6 builds upon the ideas from Agent 2 but adapts them for the partial information setting of the predator.

Improvement logic

For each possible next action node, the agent finds the hypothetical utility of the predator from each position in the environment, using $0.9^{distance\ from\ node\ to\ action\ node}$ then multiplies it by the belief vector value that the predator is in that node. The agent multiplies by the belief in order to weight the utility the predator would give by being in that node, such that highly probable nodes have a much larger influence on utility. These utilities are then combined to create a composite predator utility for a given next action node. Since the prey locations are known, the prey utility is calculated in the same way as agent 2 and the composite predator utility is subtracted from this prey utility for each next possible action node. The next action node with the highest utility is chosen as the next agent position.

Results

```
Agent 6:
Caught (including timeout): 96.1% | Died (including timeout): 3.9% | Timed Out %: 0.0%
Caught (excluding timeout): 96.1% | Died (exlcuding) timeout): 3.9% | Avg Steps: 11.870333333333333
Steps where certain of predator pos (and correct): 62.576%
```

The Combined Partial Information setting

Agent 7

Agent 7 only has information about the initial position of the predator and no information on the prey. So, Agent 7 must maintain belief vectors for prey and predator. The belief updates happen according to

the logic outlined for Agents three and five. But for each timestep, the Agent is only allowed to survey one node, and the choice of which node to survey depends on the belief vectors. If the location of the predator is known, that is the belief of predator for a node is one, then the node with the highest prey belief is chosen for surveying.

Also, the survey function for this Agent returns an index for both the most likely predator position and most likely prey position.

Governing Logic

After each survey, both the predator belief vector and the prey belief vector are updated based on the new information. These updates follow the same logic as Agent 3 (Prey update) and Agent 5 (Predator update).

Like the other previous Agents three and five, after surveying and updating the belief vectors, Agent 1 logic prevails until the next Agent position is ascertained. Here for the **agent_moved()** function, both the prey and predator belief vectors are updated. And the transition function calculates the new predator transition matrix and simultaneously calculates the next state of both the belief vectors.

Results

```
Agent 7:
Caught (including timeout): 78.433% | Died (including timeout): 21.567% | Timed Out %: 0.0%
Caught (excluding timeout): 78.433% | Died (exlcuding) timeout): 21.567% | Avg Steps: 29.709
Steps where certain of prey pos (and correct): 1.735
Steps where certain of predator pos (and correct): 57.141%
```

Agent 8

Agent 8 abides by the belief updates and setting rules of Agent 7. Since the exact location of the prey and predator are not always known, Agent 8 serves as a combination of Agents 4 and 6.

Improvement logic

Agent 8 finds two composite utilities for each possible next action node, one for the prey calculated the same way as in agent 4 and one for the predator, calculated the same way as in agent 6. The new composite predator utility is then subtracted from the new composite prey utility for each next action node. The next action node with the highest utility is chosen as the next agent position.

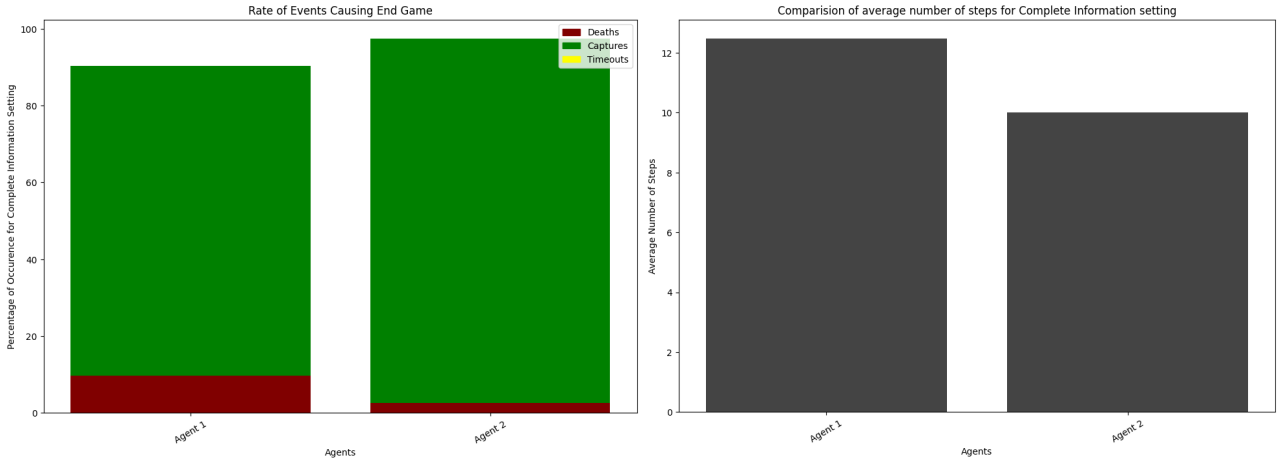
Results

```
Agent 8:
Caught (including timeout): 88.7% | Died (including timeout): 11.3% | Timed Out %: 0.0%
Caught (excluding timeout): 88.7% | Died (exlcuding) timeout): 11.3% | Avg Steps: 32.39933333333333
Steps where certain of prey pos (and correct): 2.234%
Steps where certain of predator pos (and correct): 53.667%
```

Comparative Analysis

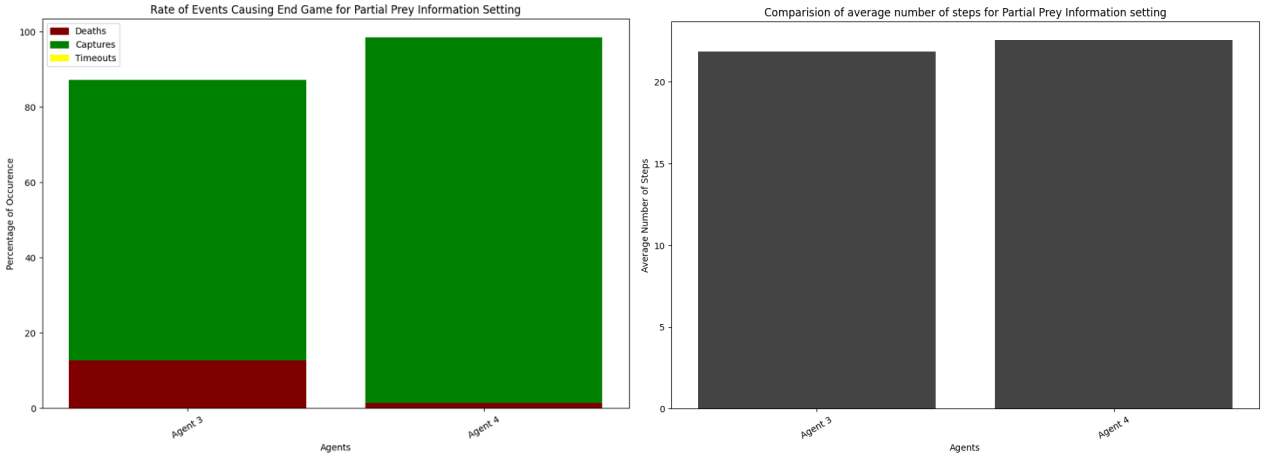
All the above agents are simulated over a hundred times on thirty different graphs. The following are the obtained results and a comparative analysis of these results.

Complete Information setting



The set of graphs above compare the survival and death rates (left), and average number of steps (right) for the rule-based Agent 1 and the utility-based Agent 2 in 3000 complete information simulated scenarios. There is a moderate increase in survivability (about 7%) for the utility-based agent and an even more notable decrease in the number of steps by the utility-based agent. The cause of the higher survivability and lower death rate is that while Agent 1, while good at moving to the prey node, it is not good a balancing between approaching the prey and moving away from the predator, meaning that in some scenarios when attempting to get closer to the prey, it may allow the predator to come too close. The result is that it will have to strictly retreat and may die trying or will travel relatively far and incidentally collide with the prey. Agent 2 is attempts to find a balance of closeness to prey and farness from predator meaning that most steps will advance towards a "reward", reducing the rate of dire retreating and therefore the number of steps.

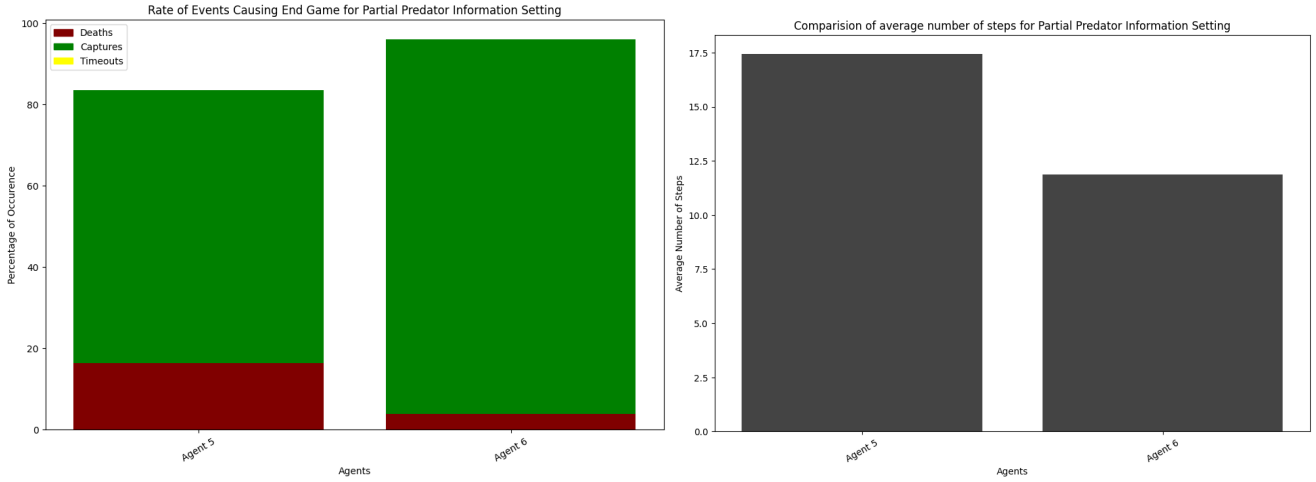
Partial Prey Information setting



The set of graphs above compare the survival and death rates (left), and average number of steps (right) for the rule-based Agent 3 and the utility-based Agent 4 in 3000 partial prey information simulated scenarios. The utility-based approach of Agent 4 results in an approximate 10% improvement in survivability, however the average number of steps remains similar in both Agents. Agent 3 assumes that the prey is contained by the highest belief node, however Agent 4 considers all positions of possible

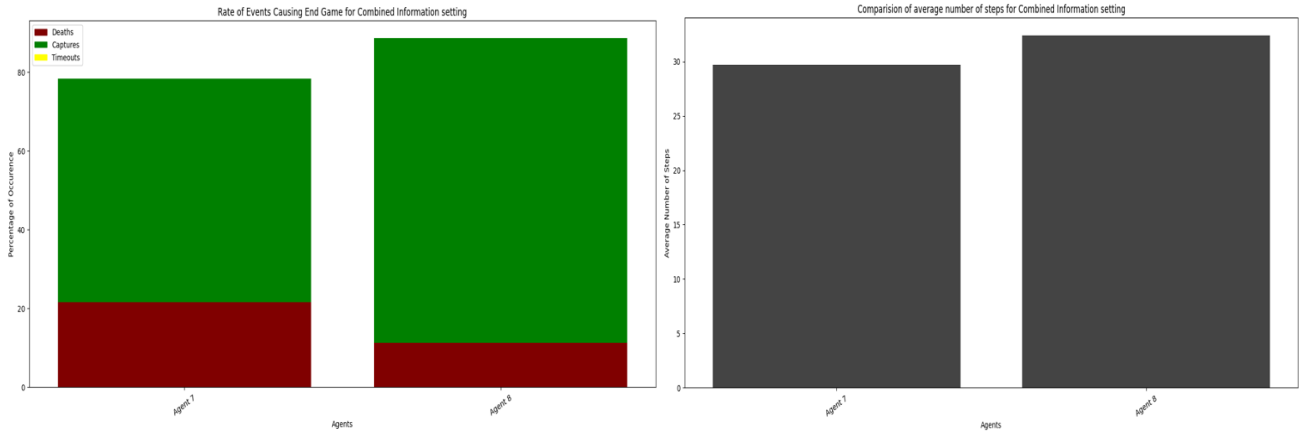
preys in its next decision, decreasing their weight based on belief and distance. The result is that Agent 3 will sometimes overcommit to node of higher probability (even if only marginal higher belief) that is far away, whereas Agent 4 will attempt to find a balance. This is reflected in the number of steps taken, where Agent 3 takes many steps (because it will attempt to travel to far nodes) and Agent 4 takes many steps because it will not take direct paths to nodes in favor of a position that is best for all node beliefs (since the certainty of prey position is usually not high).

Partial Predator Information setting



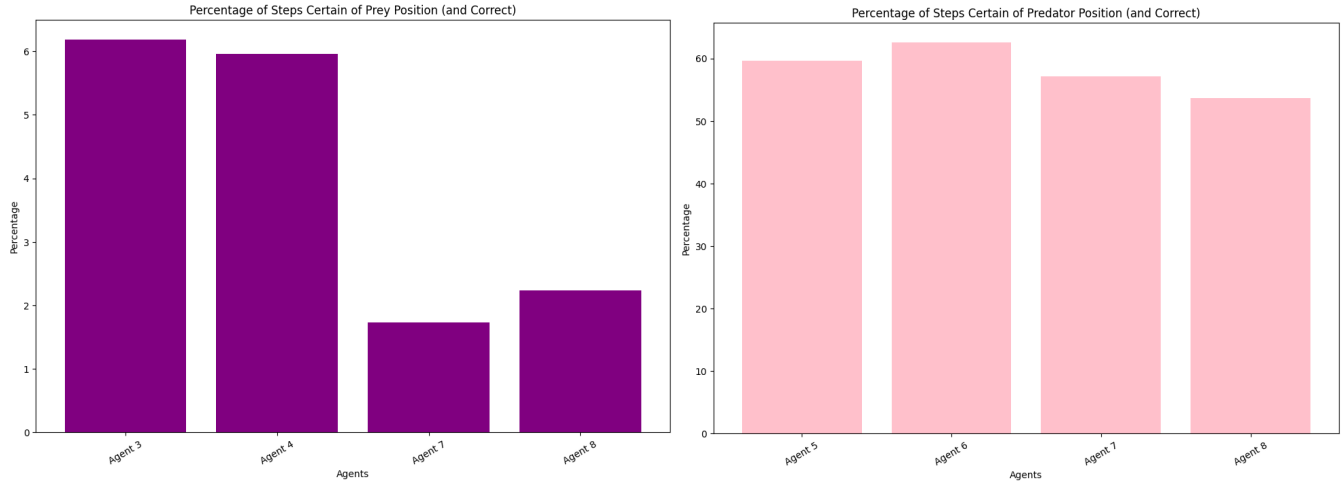
The set of graphs above compare the survival and death rates (left), and average number of steps (right) for the rule-based Agent 5 and the utility-based Agent 6 in 3000 partial predator information simulated scenarios. There is a notable improvement of Agent 6 over agent 5 (approximately 13%) and like in the complete information scenario the utility-based agent takes significantly fewer steps. The reduced number of steps taken by Agent 6 occurs for the same reason as in Agent 2, which is that the utility-based approach is better able balance prey and predator position. Additionally, Agent 6's survivability is much higher than Agent 5's because it considers the potential that the Predator is at any state with a belief value, rather than just the state of highest belief. This means that is better able to avoid collisions with the predator by finding a node that is on average best, especially since the distracted predator's position is less predictable.

Combined Information Setting



The set of graphs above compare the survival and death rates (above left), and average number of steps (above right) for the rule-based Agent 7 and the utility-based Agent 8 in 3000 combined partial information simulated scenarios. Agent 8 performs about 10% better than Agent 7 while also taking more steps. Due to the distracted (less predictable) movement of the predator, after the first step where they are certain of the predator position, both agents tend to only survey for the predator. Since the agents are only generally aware of the region of the predator, but not the prey, the most important factor for survival is avoiding the predator. Agent 8 like Agent 6 is very good at doing this since it factors in the several positions the predator may exist. Furthermore, given the few surveys of the prey that occur in this scenario, the Agent's are generally uncertain of the prey position. However, Agent 8, like Agent 4 is good at finding a "best" node, meaning that in this uncertainty it tends to do well by eliminating closer nodes first. Agent 8 ultimately takes more steps than Agent 7 because it is able to survive longer and therefore will take more steps while it attempts to locate the prey.

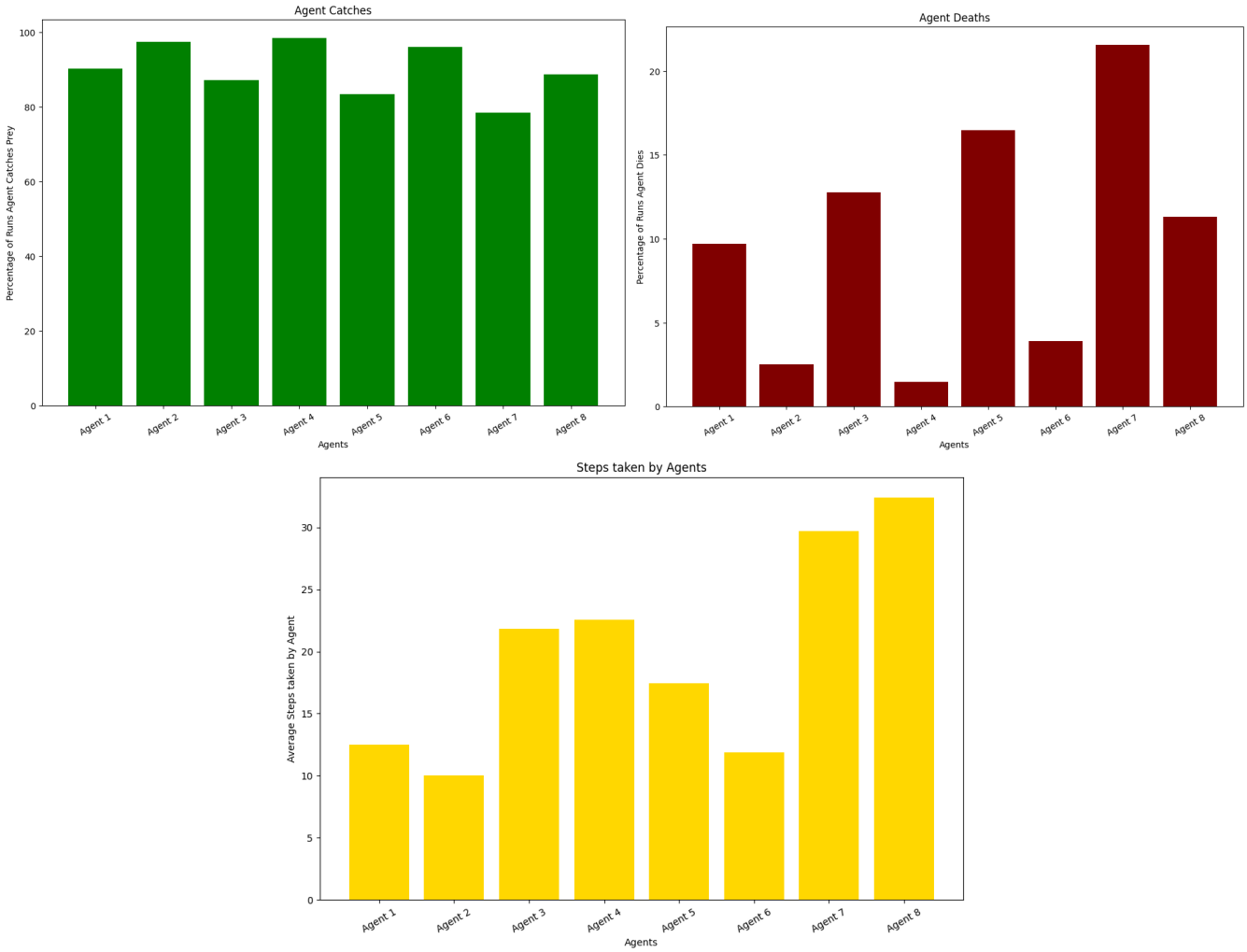
Rates of Correct Prediction



Since the agents in the combined partial information setting frequently are not certain of the predator position (which is because the predator has a chance of moving randomly) and will only intentionally survey for prey if they are certain of the predator position, they survey for the prey infrequently. Since the prey begins in any node with a 1/49 probability and is not usually being surveyed for, improvement of the probabilities comes incidentally from movement and from surveys intended for the predator finding the prey. This is different to the partial prey information settings which focus only on improving the beliefs of the prey. Since the prey still begins with a 1/49 probability of starting in any node, it is still difficult to narrow down the prey location. As can be seen in the left graph it still manages to be about twice as effective.

As stated previously, the agents in the combined partial information setting frequently are not certain of the predator position (since transitions can be random), meaning that it focuses on surveying the predator more often. Additionally, since the predator has a higher probability of moving towards the agent, rather than randomly, a lost predator can be found relatively quickly since it will always move generally towards the agent, however it will also frequently be lost. The result is that the partial predator and partial combined certainty of predator position are very similar, with the combined certainty being slightly lower since it will use some surveys to look for the prey.

Comparing Agents of all settings



There is a clear trend among the information settings for the rule-based (odd) Agents, where settings of more information allow for higher levels of survivability and take fewer steps. Partial prey tends to be more survivable than partial predator for odds because it is always sure of the predator position and therefore more confidently able to avoid it, which can be seen in a higher number of steps for Agent 3 over Agent 5, since it will retreat more successfully. For utility-based (even) Agents however this trend is less clear. Agent 4 has a higher survivability rate (and lower corresponding death rate), when compared to Agent 2. This is because Agent 4 is fairly uncertain of the prey position and finds higher utility in avoiding the predator, whereas Agent 2 is certain of the prey position and may occasionally risk getting closer to the predator in attempt to catch the prey. This is evident in the step graph, where Agent 4 takes more than double the steps of Agent 2. Besides Agent 4, the trend of decreased success with less info for the even agents remains like the odd agents, with a corresponding increase in average steps taken. Overall utility-based agents tend to have better survivability over rule-based agents, with the down side of potentially more steps.

Defective Drone Setting

The defective drone setting is a modification of the combined partial information setting (agents 7 and 8), where each survey of a node where one or more actors is present has a 0.1 probability of returning false instead of true. The original agents 7 and 8 are first run against this setting with no modifications to their belief updates, each implemented in agent_7_defective.py and agent_8_defective.py respectively. They then have their belief systems updated in, agent_7_defective_updated.py and agent_8_defective_updated.py order to account for the potential of a false negative. These updated agents contain only updates to their belief systems, but still function in the same way as the original agents 7 and 8.

Updated Belief System

The updated belief system (used in agent_7_defective_updated.py and agent_8_defective_updated.py) functions based on the same dependent probability as before, where a true survey has the same update, and a false survey has a minor modification for the surveyed node:

$$P(A \text{ is at } N \mid \text{False Survey } N)$$

$$= P(A \text{ is at } N, \text{ False Survey } N) / P(\text{False Survey } N)$$

$$= P(A \text{ is at } N) * P(\text{False Survey } N \mid A \text{ is at } N) / P(\text{False Survey } N)$$

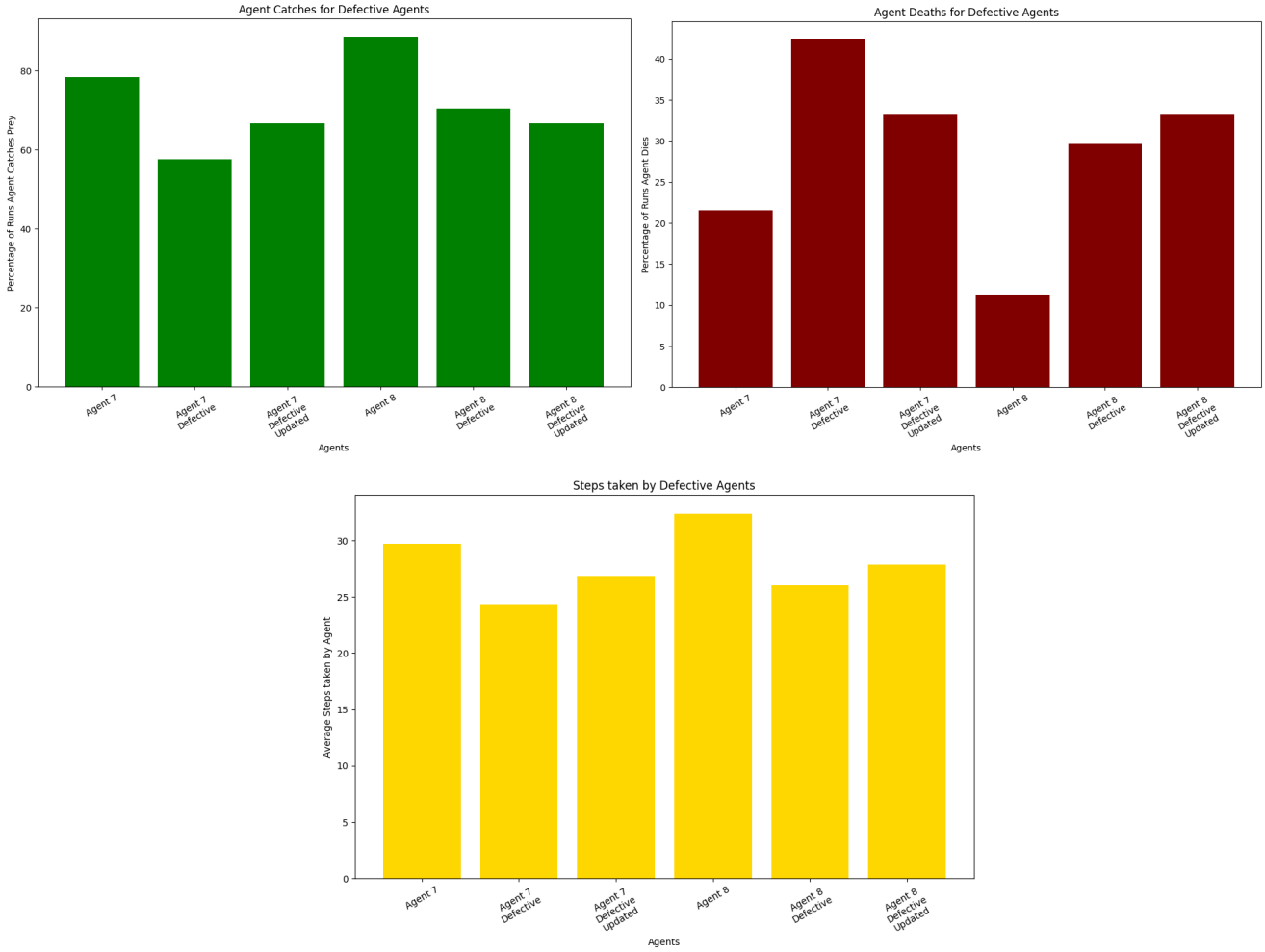
$$= P(A \text{ is at } N) * P(\text{False Survey } N \mid A \text{ is at } N) / \text{Sum of probabilities}$$

However, previously the term P(False Survey N | A is at N) always was equal to 0, meaning that the belief of a node after a false survey could always be set to 0. Instead, this probability is now 0.1 meaning the new probability is 0.1 * the belief that the actor was at the surveyed node divided by the new sum of beliefs. The remaining node beliefs are updated in the same way as previously, by dividing by the new sum of probabilities, since for some node M, P(False Survey N | A is at M) = 1.

Results:

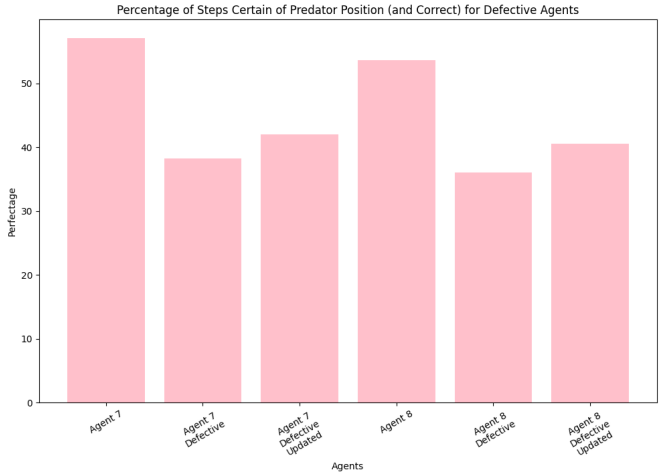
```
Agent 7 Defective:
Caught (including timeout): 57.567% | Died (including timeout): 42.433% | Timed Out %: 0.0%
Caught (excluding timeout): 57.567% | Died (exlcuding) timeout): 42.433% | Avg Steps: 24.342333333333332
Steps where certain of prey pos (and correct): 1.594%
Steps where certain of predator pos (and correct): 38.271%

Agent 7 Defective Updated:
Caught (including timeout): 66.733% | Died (including timeout): 33.267% | Timed Out %: 0.0%
Caught (excluding timeout): 66.733% | Died (exlcuding) timeout): 33.267% | Avg Steps: 26.879333333333335
Steps where certain of prey pos (and correct): 1.63%
Steps where certain of predator pos (and correct): 41.995%

Agent 8:
Caught (including timeout): 88.7% | Died (including timeout): 11.3% | Timed Out %: 0.0%
Caught (excluding timeout): 88.7% | Died (exlcuding) timeout): 11.3% | Avg Steps: 32.39933333333333
Steps where certain of prey pos (and correct): 2.234%
Steps where certain of predator pos (and correct): 53.667%

Agent 8 Defective:
Caught (including timeout): 70.367% | Died (including timeout): 29.633% | Timed Out %: 0.0%
Caught (excluding timeout): 70.367% | Died (exlcuding) timeout): 29.633% | Avg Steps: 26.018
Steps where certain of prey pos (and correct): 1.983%
Steps where certain of predator pos (and correct): 36.02%

Agent 8 Defective Updated:
Caught (including timeout): 81.233% | Died (including timeout): 18.767% | Timed Out %: 0.0%
Caught (excluding timeout): 81.233% | Died (exlcuding) timeout): 18.767% | Avg Steps: 27.866
Steps where certain of prey pos (and correct): 1.922%
Steps where certain of predator pos (and correct): 40.734%
```

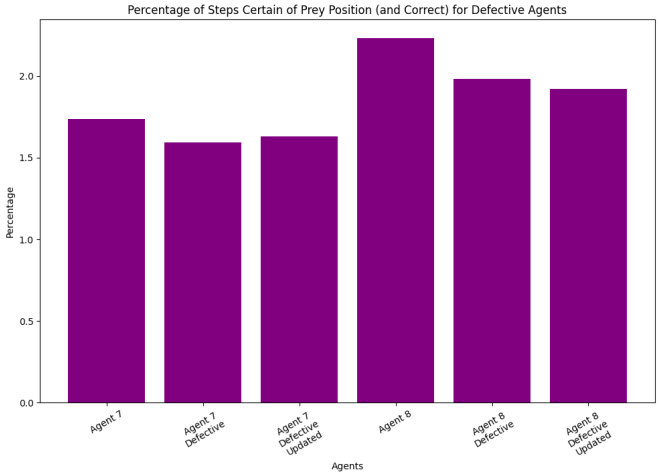
Defective Comparative Analysis



The above plots show Agents 7 and 8 compared to their defective and defective updated counterparts. Using the original belief updates for defective Agents results in the potential of a false negative survey, where the Agent believes the actors are not present in a node that they are present in. This is especially detrimental during an incorrect predator survey, since the agent may accidentally move into node the predator is present, or worse become confident that the predator is in a cell that it is not (in some rare cases this can even result in a belief the predator is not in the graph at all, caused by a survey of a node with a belief of 1). The result is the predator not only dies more frequently (about 20% more) but also dies in fewer steps. The updated belief system however results in an approximate 10% increase of survivability for both Agent 7 and 8. The death rate is still higher than the original Agents 7 and 8 because there is an additional factor of uncertainty in considering a false negative, meaning it is never fully certain that the actors are not contained by a specific node. This can be seen in the graphs below, for predator certainty (and correct), the defective agents always have the lowest rates, followed by updated defective, then regular agents. The inability to be certain of an empty node ultimately makes knowing the predator position more difficult. The percentage of steps where the agent knows the prey's position and correct does not show the same obvious trend, all defective agents performing similarly, with agent 8 defective slightly out performing its updated counterpart. This is mostly due to the

infrequency of survey with the purpose of finding the predator. Since the defective agent often holds incorrect beliefs, it tends to survey more nodes and occasionally find the prey more often however this behavior is incidental.





Agent 9 (a hypothetically improved defective drone agent):

All previous even numbered agents all attempt to improve upon their pre-determined choice counterparts using a bellman-like algorithm, which simplifies the concept of decaying utilities for further rewards. While this works very well, increased levels of uncertainty make the approximation less accurate. This is because the current implementation does not consider the fact the actors move, meaning that for each step, not only does the decay have to be applied to the previous reward, the potential new positions of the rewards must also be considered.

An additional improvement that can be made for this hypothetical agent is to implement some degree of stickiness around node with very high belief. For example, if there was a node with a 0.9 believed probability of containing an actor and all remaining probability nodes were relatively low and spread apart. If the 0.9 belief node surveys as false, the agent could continue to survey in the area near the 0.9 belief node for several moves, since it is likely that the survey was a false negative. This is because a node with a relatively high chance of containing an actor means there is a high chance of finding the actor in that node:

Given P(actor is in N)

P(finding actor in N)

= P(finding actor in N AND actor is in N) + P(finding actor in n AND actor is not in N)

= P(finding actor in N AND actor is in N) + 0

*= P(finding actor in N) * P(finding actor in N | actor is in N)*

*= P(finding actor in N) * 0.9*

This means that nodes for node with much higher probabilities of containing the actor, the probability of finding the actor there is much higher. The stickiness could be tied into the belief, so that the lower the belief of finding an actor in N, the less prone it will be to assuming a false negative. Furthermore, the agent will continue to consider the predator to be in the general area until the sticky surveys finish.

Realizations

During the whole development process of this project, we realized quite a few important things that completely changed our approach and results. Some of the most notable ones are discussed below:

Although we were choosing the most optimal node based on the priorities given in the writeup, there were no guidelines to choosing the most optimal ones. If there was more than one adjacent node with the highest priority, then our initial implementation was not choosing the optimal one. So, to break ties and find the most optimal node, a function was written which chose the best node from the available nodes by checking the same six priorities. This change improved our odd agents and brought a gain of approximately 5%-6%. This can be seen in the sub-folder **og_odd_agents**, which also contains its own main that compares the original to the current odd agents, the result of which can be seen below.

```
Agent 1_og:
Caught (including timeout): 85.367% | Died (including timeout): 14.633% | Timed Out %: 0.0%
Caught (excluding timeout): 85.367% | Died (exlcuding) timeout): 14.633% | Avg Steps: 10.945666666666666

Agent 1:
Caught (including timeout): 90.133% | Died (including timeout): 9.867% | Timed Out %: 0.0%
Caught (excluding timeout): 90.133% | Died (exlcuding) timeout): 9.867% | Avg Steps: 12.753666666666666

Agent 3_og:
Caught (including timeout): 81.6% | Died (including timeout): 18.4% | Timed Out %: 0.0%
Caught (excluding timeout): 81.6% | Died (exlcuding) timeout): 18.4% | Avg Steps: 20.505666666666666
Steps where certain of prey pos (and correct): 5.844%

Agent 3:
Caught (including timeout): 88.1% | Died (including timeout): 11.9% | Timed Out %: 0.0%
Caught (excluding timeout): 88.1% | Died (exlcuding) timeout): 11.9% | Avg Steps: 21.964333333333332
Steps where certain of prey pos (and correct): 6.335%

Agent 5_og:
Caught (including timeout): 79.267% | Died (including timeout): 20.733% | Timed Out %: 0.0%
Caught (excluding timeout): 79.267% | Died (exlcuding) timeout): 20.733% | Avg Steps: 15.014666666666667
Steps where certain of predator pos (and correct): 60.905%

Agent 5:
Caught (including timeout): 83.7% | Died (including timeout): 16.3% | Timed Out %: 0.0%
Caught (excluding timeout): 83.7% | Died (exlcuding) timeout): 16.3% | Avg Steps: 18.171
Steps where certain of predator pos (and correct): 58.276%

Agent 7_og:
Caught (including timeout): 73.967% | Died (including timeout): 26.033% | Timed Out %: 0.0%
Caught (excluding timeout): 73.967% | Died (exlcuding) timeout): 26.033% | Avg Steps: 27.397666666666666
Steps where certain of prey pos (and correct): 1.768%
Steps where certain of predator pos (and correct): 57.05%

Agent 7:
Caught (including timeout): 76.333% | Died (including timeout): 23.667% | Timed Out %: 0.0%
Caught (excluding timeout): 76.333% | Died (exlcuding) timeout): 23.667% | Avg Steps: 29.754333333333335
Steps where certain of prey pos (and correct): 1.864%
Steps where certain of predator pos (and correct): 56.027%
```

Prior to using utilities, we attempted to use the transition models we created to predict where prey and predator would be before they arrive at those positions. Ultimately the predator predictions were not extremely helpful since it could always be assumed the predator will generally move toward the agent, even in distracted settings. Additionally predicting the prey often resulted in a lot of noise since the prey moves so randomly, and usually we found that the highest expected position was the same node as the highest belief. Using this method with the original rule-based approach resulted in a 2% increase in performance at best and occasionally performed worse. We also attempted to find steady states of the prey and to move there but found it was often not successful.

Since the current state of utilities does not consider the future movements of the prey and or predator, these agents could potentially be improved further by considering these factors, the tradeoff would be much higher complexity. This is because the transitions would have to be considered in an infinite time horizon type setting where the horizon would have to be recalculated each time the beliefs change. discounting does a fairly good job and increasing preference for closer nodes meaning the agent will attempt to check nodes of higher probability nearby since the result is more useful. If the prey is at the nearby node the game ends and if not, the far away node of high certainty becomes even higher. Additionally, since the predator's movement is somewhat predictable even in the distracted setting, utility helps the agent avoid the general area of the predator all together.