CS 520 - Project 2: The Circle of Life

Akanksha Reguri (ar2085), Khyati Doshi (kbd57)

November 17, 2022

1 Abstract

The problem statement is to design a probabilistic model of an environment in the presence of uncertainty and use it to inform and direct decision-making. The environment consists of 3 entities - the prey, the predator, and the agent. The agent is in search of prey and the predator is in search of the agent. Each agent built in this project comes with it's own capabilities. A capability is defined as how much information the agent knows about the environment and how it behaves in the environment.

2 The Environment

The Environment is a graph of 50 nodes, numbered 1 to 50 connected by edges. The agent,the prey, and the predator move between the nodes connected by edges. The nodes are connected in the circle and additionally, random edges are added to increase connectivity in the circle. Random edges are added based on the following rules:

1. Picking a random node with a degree less than 3
2. Adding an edge between it and one node within 5 steps forward or backward along the primary loop. (So node 10 might get connected to node 7 or node 15, but not node 16.)
3. This is repeated until no more edges can be added. Since there is already an edge from the nodes X to X-1 and X to X+1 , we won't be adding another edge to these nodes while adding an additional edge from X.

When an environment is created by calling the class The_Environment, the following entities are created which are unique to each and every environment:

- Edges - Connection between the nodes. This is a dictionary where keys are the node numbers and values are the list of neighboring node numbers. We are using adjacency list instead of adjacency matrix to save space as in our use case each vertex will only be connected to at most 3 other vertices.
- Prey location - Location of the birth of the prey
- Predator location - Location of the birth of the predator
- Agent location - Location of the birth of the agent

The agent at birth can't occupy a node that is already occupied so, the environment created makes sure that:

agent-birth-location != predator-birth-location and agent-birth-location != prey-birth-location

2.1 The Predator

Predator wants to catch the agent. Every time the Predator moves, it looks at its available neighbors, and calculates the shortest distance to the Agent for each neighbor it can move to. It then moves to the neighbor with shortest distance remaining to the Agent. If multiple neighbors have the same distance to the Agent, the Predator selects uniformly at random between them.

2.1.1 Easily Distracted Predator

This predator gets easily distracted - with probability 0.6, the predator will move to close the distance, but with probability 0.4, the predator moves to one of its neighbors uniformly at random. This means that on average the predator is moving to the neighbor that is closer to the agent, but it is not guaranteed in any round.

2.2 The Prey

The rules of the prey are simple. Prey chooses uniformly at random one of the locations from it's neighbors and including it's own location (that is if the prey has three neighbors, there is a 1/4 probability of staying where it is). This continues, regardless of the actions or locations of the others, until the game concludes.

2.3 The Agent

Agent's goal is to catch the prey, escaping from the predator. Next section describes the various types of Agents designed and implemented in the environment.

3 Agents

3.1 Agent 1

Agent 1 knows complete information about the environment. Agent 1 assigns priority to each of the choices it can take, including it's current location, and chooses the location with the highest priority. Agent 1's top priority is running away from the predator while trying to get closer to the prey. Agent 1's decision-making is a simple set of if-else conditions provided in the project write-up.
How often agent catch the prey? Survivability: Agent 1's survivability is 92 percent.
How often predator catches the agent? Death rate: Agent 1 is dying in 8 percent scenarios. The reason that Agent 1 is dying in these scenarios, even when it tries to stay away as far as possible from the predator is that Agent 1's top priority is moving away from the predator and closer to the prey. If there is no choice that is providing this benefit, it moves to a location that is the same distance as of current location to the predator. But the predator has the benefit of finding the shortest path to the agent and moving to that location. So now gradually after a few timestamps distance would be decreasing and predator would catch the agent.
How often simulation hangs? With the timeout being 50, Agent 1's hung cases are 0 percent
Average step count: It is total number of steps/number of wins. Agent 1's average step count is 11.50.

3.2 Agent 2

Agent 2 knows the exact location of the prey and the predator. Agent 1's failure cases were analysed and Agent 2's design handles them. Thus Agent 2 improves upon Agent 1 in the following scenarios:

- Considering the case in which the Agent dies(gets caught by predator) or timeouts before catching the prey. This could happen when the Prey keeps on moving on the path away from the Agent, while agent either stays on the same location or move towards the prey, then the distance between the agent and the prey increases or remains constant respectively. To handle this, agent 2 calculates the distance not to the current location of the prey but to the future nodes where the prey can move to. So a simple analogy is instead of aiming at a dot, we are aiming at a bubble of dots(4 or 3 depending on number of neighbors). So the average distance of the shortest path from each agent neighbour to each of the node the prey can move to in the next timestamp is calculated. Thus a future region of prey is considered instead of pointing to the very location of the prey. For example - In figure 1, we consider the nodes 50,47,45 and the current prey location for calculations.

- To increase the survivability the agent would eliminate the choice of neighbor node it can move to, if the predator is present in the neighbor of that neighbor.

Figure 1: Agent and future prey location.

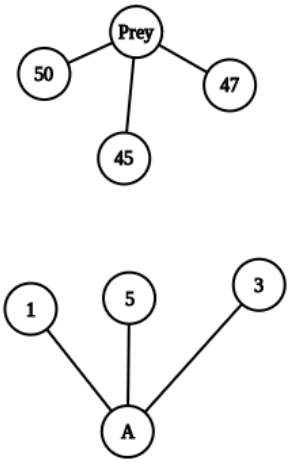
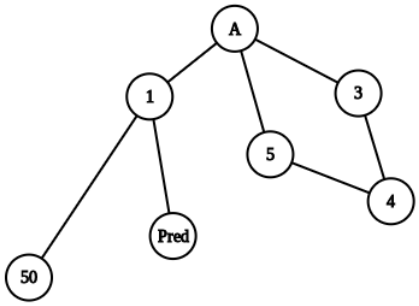


Figure 2: Agent's neighbors of neighbors has the predator.



For example - For figure 2, Agent-2's current location is node-A. It has choices of nodes - 1,A,3 and 5 ; as explained Agent-2 in all conditions will not consider choosing node 1, as the Predator is a neighbor of node 1.

Thus, Predator will only be able to catch the Agent in only the mentioned below edge case:

- If the Agent and the Predator are born in the adjacent cells (refer to figure 3), and any choice that agent can take is a neighbor to the predator, this scenario would result in predator catching the agent irrespective of agent's choice.

How often agent catches the prey? Survivability: Agent 2's survivability is 99.7 percent.
How often predator catches the agent? Death rate: Agent 2 is dying in 0.25 percent scenarios i.e. about 7 times in 3000 runs. The only scenarios in which the Predator can catch Agent 2 is explained by the edge case explained above.
How often simulation hangs? With the timeout being 50, Agent 2's hung cases are 0.05 percent i.e. approximately 1 times in 3000 runs.
Average step count: Agent 2's average step count is 11.77
When do your Agents outperform the specified Agents in the above, and why? Do you think that your Agents utilize the available information as effectively as possible? Why or why not?
Agent 2 outperforms the Agent 1 in the above mentioned scenarios. Since this agent is considering the

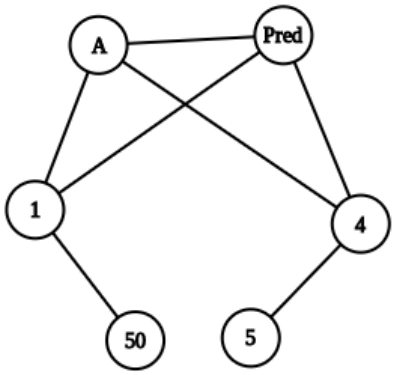


Figure 3: Edge case.

future locations of the prey instead of the very location of the prey, it moves towards a region instead of a dot point which is the primary reason behind why the survivability of Agent 2 is much higher than Agent 1 **Survivability difference** - approximately 8 percent

3.3 Agent 3

Agent 3 runs in a partial prey information setting. It always knows where the predator is, but not necessarily where the prey is. Agent 3 tracks a belief states for each of the available nodes. Every time Agent learns something about the prey and everytime prey moves, these belief states would be updated.
Initial prey probability distributions: Except for the node, where the agent is present(probability of prey = 0), there is an equal probability for the rest of the 49 cells to have the prey and thus each of those cells has an initial probability of 1/49.
Probability updates for prey: Probability updates for prey happens in the following scenarios.

1. When Agent learns something about the prey: This happens in two cases:

   - Agent surveys a node: Agent can survey a node anywhere in the graph. Following is the description followed by a code snippet explaining how prey belief states are updated. If prey is found in the survey node then the resulting probabilities would be as follows:

     $$P(survey\text{-}node) = 1 \text{ and } P(rest \text{ of the nodes}) = 0$$

     Else if the prey is not found in the node being surveyed we capture the probability

     $$\boldsymbol{P(prey \text{ not found in survey-node}) = 1 - P(survey\text{-}node)}$$

     and update the probability of the survey node as :

     $$P(survey\text{-}node) = 0$$

     for all other nodes in the graph, based on conditional probability:

     $$P(node) = \frac{P(node)}{P(prey \text{ not found in survey} - node)}$$

     Below code snippet explains clearly on how P(prey not in survey-node) is calculated.

```
def get_failure_belief(survey_node):
    # This function returns the belief with which there is no prey in the survey node
```

```
        # Applying marginalization
        P(no prey in survey_node) = Sum(P(prey in node x, no prey in survey_node)
                                    for every x in the graph)
        # Applying conditional probability
        P(no prey in survey_node) = Sum(P(prey in node x)*
                                P(no prey in survey_node | prey in node x))
        # Splitting the summation into survey and non-survey nodes
        P(no prey in survey_node) = Sum(p(prey in node x)*
                P(no prey in survey_node | prey in node x))
                for every other node except survey_node +
                P(prey in survey_node)*P(no prey in survey_node |
                prey in survey_node)
        # The second entity would be zero since
        # P(no prey in survey_node | prey in survey_node) = 0 and
        # P(no prey in survey_node | prey in node x) = 1
        # Hence P(no prey in survey_node) is sum of all
        # other prey probabilities except itself.
        # Thus simplifying to
        P(no prey in survey_node) = 1 - P(prey in survey_node)
        return P(no prey in survey_node)

    def prey_belief_updates():
        if prey found in survey_node:
            for every node in graph:
                if node == survey_node:
                    P(node) = 1
                else:
                    P(node) = 0
        else:
            node_failure = get_failure_belief(survey_node)
            for every node in graph:
                if node == survey_node:
                    P(node) = 0
                else:
                    # Based on conditional probability, the belief of every
                    # node given that there is no node in survey node would be
                    P(node) = P(node)/node_failure
```

- Agent entering a neighboring node: If agent enters a neighbor and founds prey in that node, the agent wins. But if there is no prey in that particular node, based on marginalization

  P(prey not found in neighbor-node) = 1 - P(neighbor-node)

  P(neighbor-node) = 0

  for every node in the graph, based on conditional probability:

  $$P(node) = \frac{P(node)}{P(prey\ not\ found\ in\ neighbor-node)}$$

  These belief state updates are similar to that of how belief state updates happened in case of survey-node since it is just agent looking at a node and knowing the status of the prey location. For the above code snippet, the neighbor node would be the survey node. Except that in the case of survey-node if we found the prey, belief state updates would be 0 and 1. But in case prey is found in the neighboring node, agent would move to that node and agent wins. (Game ends).

2. When Prey moves: Every time the prey moves, the probability of the nodes needs to be updated. Prey probability updates occur according to the below pseudo code

```
new_prey_prob = deepcopy(prey_prob)
for every node:
  new_prey_prob[node] = 0
```

```
for every node:
  no_neighbors = neighbors(node) + 1
  for every neighbor:
    new_prey_prob[neighbor] += prey_prob[node]/no_neighbors
  new_preb_prob[node] += prey_prob[node]/no_neighbors
return new_prey_prob
```

How often agent catches the prey? Survivability: Agent 3's survivability is 85 percent.
How often predator catches the agent? Death rate: Agent 3 is dying in 14 percent scenarios.
How often simulation hangs? With the timeout being 100, Agent 3's hung cases are 0.4 percent
How often the Agent knows exactly where the Prey is during a simulation where that information is partial: On an average 1.5 times in a single game
Average step count: Agent 3's average step count is 25.81.

3.4 Agent 4

Agent 4 runs in a partial prey information settings. This agent is an improvement over Agent 3. Agent 4 has the same information of what Agent 3 (knows the location of the predator but not necessarily of the prey). This agent is a hybrid model of Agent 2 and Agent 3. Belief state updates of Agent 4 is same as Agent 3 in all the scenarios. The only thing that Agent 4 is different from Agent 3 is how it chooses it's next step in the next timestamp provided the belief states. For every choice the agent can take, it calculates the weighted average distance from the choice to all the prey locations that the prey can take in future. Here weight is the probability of those future prey locations. And similar to Agent 2, it doesn't consider moving to node, where predator is present in the neighbor of that node. Below code snippet gives a clear picture on how agent 4 works.

```
If uncertain of where the prey is:
    distances = {}
    for every neighbor node of agent_location:
        for every prey future location:
            distances[neighbor_node] += distance(neighbor_node,
            prey_future_loc)*P[prey_future_loc]
    run on priority rules of Agent 1 on how to choose the agent_loc
else if certain of prey location:
    # It follows Agent 2 since it is now a complete
    # information setting
# Clearly the neighbor node with least distance to the prey and
# farthest from the predator would be chosen by the agent
# Note: Prey future locations include it's current location
# and all it's neighbors
# We skipped normalizing with weights since it is equal to all the
# choices agent can take and our goal is just comparison
```

How often agent catches the prey? Survivability: Agent 4's survivability is 96.8 percent.
How often predator catches the agent? Death rate: Agent 4 is dying in 0.2 percent scenarios.
How often simulation hangs? With the timeout being 100, Agent 4's hung cases are 3 percent
How often the Agent knows exactly where the Prey is during a simulation where that information is partial: On an average 2 times in a single game
Average step count: Agent 4's average step count is 27.95. **When do your Agents outperform the specified Agents in the above, and why? Do you think that your Agents utilize the available information as effectively as possible? Why or why not?**
Yes, Agent 4 is clearly outperforming the Agent 3 because of the following reasons:

1. It considers the future locations of the prey instead of prey location(considering a bubble, instead of a dot).
2. These future locations are again weighted by the belief states of those locations instead of the average, thus utilizing the belief states better than Agent 3 and outperforming it.

Now we can see the prey is being caught more often relative to Agent 3 as well.

3.5 Agent 5

Agent 5 runs in partial predator information settings. It always knows where the prey is, but not necessarily where the predator is. When the agent starts it knows the location of the predator. Agent 5 tracks the belief states for each of the available nodes. Every time Agent learns something about the predator and every time the predator moves, these belief states would be updated. This Agent would have been pretty straightforward if the predator always moves to one of the nodes, leading to the shortest distance to the agent. In such a case, the agent almost never loses the location of the predator. But the predator here is an easily distracted predator. It moves to the node leading to the shortest distance with a probability of 0.6 and to one of it's neighboring nodes with a probability of 0.4

Initial predator distributions: Since the agent starts knowing the location of the predator, P(node) where the predator is present equals 1, and the probability of the rest of the nodes equals zero.
Probability updates for predator:

1. Agent learns something about the predator. This happens in two scenarios:

   - Agent surveys a node: Agent can survey a node anywhere in the graph and if the predator is found in that particular node then

     P(survey-node) = 1 and P(rest of the nodes) = 0

     Else if the predator is not found in that particular node, based on marginalization

     P(predator not found in survey-node) = 1 - P(survey-node)
     P(survey-node) = 0
     for every node belonging to the rest of the nodes, based on conditional probability*:*
     P(node) = P(node)/P(predator not found in survey-node)

   - Agent entering a node: If the agent enters a node and founds the predator in that node, the agent dies. But if there is no predator in that particular node.

     Based on marginalization
     P(predator not found in neighbor-node) = 1 - P(neighbor-node)
     P(neighbor-node) = 0
     for every node belonging to the rest of the nodes, based on conditional probability*:*
     P(node) = P(node)/P(predator not found in neighbor-node)

   Predator probability updates are similar to that of prey, when surveying and moving to the next node. The only difference is that which node to survey is chosen based on maximum belief in case of prey(breaking ties at random), but in case of predator survey ties are broken initially based on distance and then at random.

2. When Predator moves: Every time the predator moves, the probability of the nodes needs to be updated. For every choice, the predator can take, neighbor and optimal choices are calculated. That is for every node where the probability of the predator is greater than 0, neighbor and optimal choices are calculated. 0.4 weightage is given to nieghbor-choices and 0.6 weightage is given to optimal choices. Below is the pseudo-code explaining the predator probability updates.

```
# pred-prob is a dictionary with keys being node numbers and
# values being probability of predator being present in that node
pred-prob-copy = deepcopy(pred-prob)
clear(pred-prob) # clearing all the probabilities to zero. Since predator always moves,
# the vital part of this step is making belief states zero for the
# current nodes where it is believed to have a predator
for every node where pred-prob-copy greater than 0:
    neighbor-choices = get-neighbor-choices(node)
    optimal-choices = get-optimal-choices(node)
    optimal-choice-len = len(optimal-choices)
    neighbor-choice-len = len(neighbor-choices)
    for every node in pred-prob:
```

```
            if node in optimal-choices:
                pred-prob[node] = pred-prob[node] +
                (0.6/optimal-choice-len)*pred-prob-copy[node]
            if node in neighbor-choices:
                pred-prob[node] = pred-prob[node] +
                (0.4/neighbor-choice-len)*pred-prob-copy[node]
    return pred-prob
```

How often agent catches the prey? Survivability: Agent 5's survivability is 83 percent.
How often predator catches the agent? Death rate: Agent 5 is dying in 16 percent scenarios.
How often simulation hangs? With the timeout being 100, Agent 5's hung cases are 0.03 percent
How often the Agent knows exactly where the Predator is during a simulation where that information is partial: On an average, 9 times in a game.
Average step count: Agent 5's average step count is 18.22

3.6 Agent 6

Agent 6 runs in a partial predator information settings. This agent is an improvement over Agent 5. Agent 6 has the same information of what Agent 5 (knows the location of the prey but not necessarily of the predator). This agent is an improvement over the hybrid model of Agent 2 and Agent 5. Belief state updates of Agent 6 is same as Agent 5 in all the scenarios. The only thing that Agent 6 is different from Agent 5 is how it chooses it's next step in the next timestamp. Similar to Agent 2, for every choice the agent can take, it calculates the average distance from the choice to all the prey locations that the prey can take in future and it doesn't consider moving to node, if the neighbor of the node has the highest belief state of predator. And it also calculates predator weighted average distance from the every location predator can take in future to the every agent neighbor. Here weight is probability of predator choosing a location in future. Agent 6 then assigns priority(priority rules of Agent 1) to each of the choices it can take and chooses the node with highest priority. Below code snippet elaborates more on how weighted predator distances are calculated.

```
distances = {}
agent_loc - current agent location
# As of agent 5 compute optimal and neighbor choices,
# predator can take based on distance to agent location
predator - predator location with highest probability
neighbor-choices = get-neighbor-choices(predator)
optimal-choices = get-optimal-choices(predator)
optimal-choice-len = len(optimal-choices)
neighbor-choice-len = len(neighbor-choices)
for every node in pred-neighbors:
    P[node] = 0
    if node in optimal-choices:
        P[node] = P[node] +
        (0.6/optimal-choice-len)
    if node in neighbor-choices:
        P[node] = P[node] +
        (0.4/neighbor-choice-len)
# Since this is how belief states would be distributed
# when predator moves, weighted average distance
# is calculated from all these neighbor nodes
# of predator to all the neighbors of agent
for neigh in agent_neighbors:
    if neigh of neigh has predator:
        continue
    else:
        distance = 0
```

```
        for every choice predator has:
            distance += distance(predator_choice, neigh)*P[predator_choice]
            # P[predator_choice] is the probability with
            # which predator can enter the node
            # in next timestamp
    distances[neigh] = distance
```

How often agent catches the prey? Survivability: Agent 6's survivability is 92.4 percent.
How often predator catches the agent? Death rate: Agent 6 is dying in 7 percent scenarios.
How often simulation hangs? With the timeout being 100, Agent 6's hung cases are 0.8 percent
How often the Agent knows exactly where the Predator is during a simulation where that information is partial: On an average, 12 times in a single game
Average step count: Agent 6's average step count is 22.4
Survivability difference: 10 percent approximately. **When do your Agents outperform the specified Agents in the above, and why? Do you think that your Agents utilize the available information as effectively as possible? Why or why not?**
Yes, Agent 6 is clearly outperforming the Agent 5 because of the following reasons:

1. It considers the future locations of the prey instead of prey location(considering a bubble, instead of a dot).

2. It considers the future locations of the predator, and again these future locations are weighted by the probability with which predator moves to that future location. Thus utilizing the available information in the most effective way.

Now we can see we are more certain about the predator relative to Agent 5.

3.7 Agent 7

Agent 7 runs in a combined partial information settings(both partial predator information and partial prey information). It doesn't necessarily know where the prey and predator is. When the agent starts, it knows the location of the predator but not of prey. Agent 7 tracks belief states for both predator and prey for each of the available nodes. If the agent 7 is not certain where the predator is, it survey similar to Agent 5. If the agent is certain where the predator is, it surveys in accordance to Agent 3. Once probabilities are updated based on the results of the survey, the Agent acts by assuming the Prey is at the node of highest probability of containing the Prey (breaking ties at random) and assuming the Predator is at the node of highest probability of containing the Predator (breaking ties by proximity to the Agent, then at random) and then acts according to rules of agent 1.Since Agent 7 is a hybrid(combining both agent 3 and agent 5), there is not much of difference in updating probabilities. Probability updates of prey (due to prey movement) happens similar to Agent 3 and probability updates of predator(due to predator movement) happens similar to Agent 5.
How often agent catches the prey? Survivability: Agent 7's survivability is 75 percent.
How often predator catches the agent? Death rate: Agent 7 is dying in 24.5 percent scenarios.
How often simulation hangs? With the timeout being 150, Agent 7's hung cases are 0.4 percent
How often the Agent knows exactly where the Predator is during a simulation where that information is partial: On an average 17 times in a game
How often the Agent knows exactly where the Prey is during a simulation where that information is partial: On an average 0.01 times in a game. Almost zero, which is fair since we survey for the prey only when certain of predator
Average step count: Agent 7's average step count is 40.00.

3.8 Agent 8

Agent 8 runs in a combined partial information setting. Agent 8 has the same information of what Agent 7 (both partial predator information and partial prey information). This agent is a hybrid model of Agent 7, Agent 6 and Agent 4. Belief state updates of Agent 8 is same as Agent 7 in all the scenarios. The only thing that Agent 8 is different from Agent 7 is how it chooses it's next step in the next timestamp provided the belief states. For every choice the agent can take, it calculates

the weighted average prey distance from the choice to all the prey locations that the prey can take in future(similar to Agent 4). Here weight is the probability of those future prey locations. And similar to Agent 6, it also calculates the weighted average predator distance from all the choices predator can take in future to all the agent's neighbor locations. Here weight is probability of predator choosing a location in future. It also doesn't consider moving to node, if the neighbor of the node has the highest belief state of predator.
How often agent catches the prey? Survivability: Agent 8's survivability is 86.4 percent.
How often predator catches the agent? Death rate: Agent 8 is dying in 12.5 percent scenarios.
How often simulation hangs? With the timeout being 150, Agent 8's hung cases are 1 percent
How often the Agent knows exactly where the Predator is during a simulation where that information is partial: On an average, 18 times per game
How often the Agent knows exactly where the Prey is during a simulation where that information is partial: On an average, 0.02 times per game, which is zero
Average step count: Agent 8's average step count is 37.8.
Survivability difference: 10 percent approximately. **When do your Agents outperform the specified Agents in the above, and why? Do you think that your Agents utilize the available information as effectively as possible? Why or why not?**
Yes, Agent 8 is clearly outperforming the Agent 7 because of the following reasons:

1. It considers the future locations of the prey instead of prey location(considering a bubble, instead of a dot) and again these future locations are weighted by the probability of prey in that location.

2. It considers the future locations of the predator, and again these future locations are weighted by the probability with which predator moves to that future location. Thus available information is utilized in the most effective way.

Now we can see the we are more certain about the predator and prey, relative to Agent 7

3.9 Defective survey drone

This is a scenario introduced for Agents 7 and 8, here the drone is defective and even if there is prey/predator in the node, with 0.1 probability it says that there is nothing in the node (a False negative).

3.10 Not accounting for defective drone scenario

If belief update rules are not accounted for this, then every false is assumed as there is no prey/predator in that scenario, and Agent 7 and 8 survivabilities are reduced relative to a perfect drone scenario.

3.10.1 Agent 7

For Agent 7 relying on the defective survey drone and not knowing it is defective, below are the survivabilities and death rates: **How often agent catches the prey?** Survivability: Agent 7's survivability is 55.5 percent.
How often predator catches the agent? Death rate: Agent 7 is dying in 44 percent scenarios.
How often simulation hangs? With the timeout being 150, Agent 7's hung cases are 0.07 percent
How often the Agent knows exactly where the Predator is during a simulation where that information is partial: On an average 9 times in a game
How often the Agent knows exactly where the Prey is during a simulation where that information is partial: On an average 0.02 times in a game which is zero
Average step count: Agent 7's average step count is 42.

3.10.2 Agent 8

For Agent 8 relying on the defective survey drone and not knowing it is defective, below are the survivabilities and death rates: **How often agent catches the prey?** Survivability: Agent 8's survivability is 60.4 percent.

How often predator catches the agent? Death rate: Agent 8 is dying in 39 percent scenarios.
How often simulation hangs? With the timeout being 150, Agent 8's hung cases are 0.01 percent
How often the Agent knows exactly where the Predator is during a simulation where that information is partial: On an average 9.3 times in a game
How often the Agent knows exactly where the Prey is during a simulation where that information is partial: On an average 0.01 times in a game, which is zero
Average step count: Agent 8's average step count is 43.09

3.11 Updating belief rules to account for defective drone scenario

Say Node x is surveyed and drone output is false. If drone output is true, there is for sure a prey/predator in the node. But if drone output is false, there might be a prey/predator, or truly the node is empty. Below snippet explains the belief update rules in case of a defective survey drone:

```
# Here x is the survey node and so the survey drone being defective
# By marginalization
P(node x empty) = P(node x empty | node is empty)*P(node is empty) +
                  P(node x empty | prey in node)*P(prey in node)
P(node x empty | node is empty) = 1
P(node x empty | prey in node) = 0.1
P(node x empty) is computed as above where P(node is empty)
and P(prey in node) is the beliefs of that particular node before surveying
def update_belief_rules():
    x = survey_node
    if x is not empty:
        P(x) = 1
        P(all other nodes except x) = 0
    else:
        for every y in graph:
            # By conditional probability
            P(y) = P(prey in y|node x empty)
                 = P(prey in y)*P(node x empty|prey in y)/P(node x empty)
        if y == x:
            # In case of survey node, belief of survey node
            # being empty such that there is prey present is 0.1
            P(node x empty|prey in y) = 0.1
        else:
            P(node x empty|prey in y) = 1
```

3.11.1 Agent 7

For Agent 7 relying on the defective survey drone and knowing it is defective, below are the survivabilities and death rates: **How often agent catches the prey?** Survivability: Agent 7's survivability is 64 percent.
How often predator catches the agent? Death rate: Agent 7 is dying in 35.5 percent scenarios.
How often simulation hangs? With the timeout being 150, Agent 7's hung cases are 0.2 percent
How often the Agent knows exactly where the Predator is during a simulation where that information is partial: 10 times in a game
How often the Agent knows exactly where the Prey is during a simulation where that information is partial: 0.01 times in a game
Average step count: Agent 7's average step count is 39.46

3.11.2 Agent 8

For Agent 8 relying on the defective survey drone and knowing it is defective(accounting to it), below are the survivabilities and death rates: **How often agent catches the prey?** Survivability: Agent 8's survivability is 75.76 percent.
How often predator catches the agent? Death rate: Agent 8 is dying in 23.5 percent scenarios.

How often simulation hangs? With the timeout being 150, Agent 8's hung cases are 0.4 percent
How often the Agent knows exactly where the Predator is during a simulation where that information is partial: On an average 12 times in a game.
How often the Agent knows exactly where the Prey is during a simulation where that information is partial: On an average 0.01 times in a game.
Average step count: Agent 8's average step count is 38.7.

4 Agent 9

4.1 Design

Agent 9 runs in combined partial information setting. It has the same information about the prey and predator as Agent 7 and agent 8. For the defective drone survey, on surveying a node the result might be a false negative with 0.1 probability. Agent 9 is designed to outperform Agent 7 and Agent 8 in this scenario. The design of Agent 9 is based on the concept of continuity of rational choices. If the survey drone results in positive, this would mean that the node being surveyed is truly occupied. When the survey drone results in negative, Agent 9 would consider the current probability of node being surveyed. If this probability is above a set threshold then it can assumed that the node is actually occupied and the negative result from the survey drone is a false negative. If this probability is below the set threshold then it is assumed that the node is truly not occupied.

4.2 Implementation

The threshold being used here is taken by running multiple experiments on the setup. The ideal threshold that we figured out from data of Agent 7 and 8 in case of defective survey drone is 0.2. **How often agent catches the prey?** Survivability: Agent 9's survivability is 77 percent.
How often predator catches the agent? Death rate: Agent 9 is dying in 20 percent scenarios.
How often simulation hangs? With the timeout being 100, Agent 9's hung cases are 2.2 percent
How often the Agent knows exactly where the Predator is during a simulation where that information is partial: 11 times in a game
How often the Agent knows exactly where the Prey is during a simulation where that information is partial: 0.02 times in a game
Average step count: Agent 9's average step count is 27.2

5 Bonus Agent

5.1 Design

Bonus Agent runs in combined partial information settings. This agent can either survey or move, but cannot perform both operations. At every timestamp, agent computes the utility of the survey and the utility of movement. Based on the utilities of both these actions, the agent takes a call on what to do. In the scenario where both utilities are equal, the movement would be preferred since movement gives the agent a choice to stay away from the predator, whereas survey would reduce the distance between predator and agent since predator has the benefit of moving towards the agent and agent is not going to take a step in case of survey scenario.
The utility of an action is defined as the cost of survivability. The higher the cost of survivability, the least the utility.

$$\text{for every node:}$$
$$\textit{prey-distance} = \Sigma(\textit{distance(agent-loc, node)*prey-belief[node]})$$
$$\textit{predator-distance} = \Sigma(\textit{distance(agent-loc, node)*pred-belief[node]})$$
$$\textit{cost} = \textit{prey-distance/predator-distance}$$
$$\textit{utility} = 1/\textit{cost}$$

5.2 Implementation

Below code snippet explains how the agent computes utility.

```
def get_utility(by=None):
    agent_choice = current_agent_loc
    if by == 'survey':
        # surveys the node
        if certain of pred:
            if certain of prey:
                return get_utility(by = movement)
            else:
                survey for prey
                update_prey_beliefs()
        else:
            survey for predator
            update_predator_beliefs()
    else:
        # moves to the neighbor
        prey_loc = node with highest prey belief
        pred_loc = node with highest predator belief
        # breaking ties according to the rules of agents in
        # combined partial info settings
        # For the given prey and predator location,
        # acts in terms of rules of Agent 1(just like other agents)
        agent_choice = agent1()
        update_predator_beliefs()
        update_prey_beliefs()
    pred_weight = 0
    prey_weight = 0
    for every node:
        pred_weight += distance(agent_choice, node)*pred_belief[node]
        prey_weight += distance(agent_choice, node)*prey_belief[node]
    cost = prey_weight/pred_weight
    utility = 1/cost
    return utility

def bonus_agent():
    u1 = get_utility(by = survey)
    u2 = get_utility(by = movement)
    if u1 > u2:
        perform_survey
    else:
        perform_movement
```

How often agent catches the prey? Survivability: Bonus Agent survivability is 58.5 percent.
How often predator catches the agent? Death rate: Bonus Agent is dying in 41 percent scenarios.
How often simulation hangs? With the timeout being 100, Bonus Agent's hung cases are 0.15 percent
Average step count: Agent's average step count is 34.
If instead of simulating agent1(running on rules of agent 1), if agent2 is simulated, the survivability of the bonus agent increased from 58 to 60 percent.

6 Implementation

6.1 Code Organization

This entire project has been coded in python. The list of python files includes the following:

6.1.1 The Environment

environment.py consists of The_Environment class which includes the following variables:

- vertex - there are total of 50 nodes in the graph.
- edges - this is a dictionary where key is the node and value is a list of nodes that this node is connected to. (it is basically the adjacency list for the graph.)
- prey_location - this is the node where the prey is at initially.
- predator_location - this is the node where the predator is at initially.
- agent_location - this is the node where the agent is at initially.

python environment.py would create the environment including the above entities which are unique to that particular environment instance.
init_graph(): This function generates the initial graph. The initial graph is generated connecting nodes 1 to 50, connected in a large circle.
add_random_edges() - To increase connectivity across the graph, this function add edges at random to while following certain mentioned rules.
generate_ppa(): This function sets the initial location for the prey, the predator and the agent. If the agent location is same as the predator or the prey location, the agent location will be regenerated, as the agent can't occupy an already occupied node.

6.1.2 The Agents

agent.py consists of the Agent class which is inherited from The_Environment class. This class includes functions specific for all the agents.

- agent_1: This function implements Agent 1. This function follows a simple approach of assigning priorities to each of the agent choices. The agent moves to the choice which has the highest priority. This function returns whether or not agent was able to catch the prey and the total steps taken by the agent. **python agent.py –agent_1**
- agent_2: This function implements agent 2. Agent 2 knows the location of the prey and the predator. Calculates the average distance from each agent choice that is not a neighbor of the node with predator, to the future location of the prey. Based on this a priority is assigned to each agent choice and the choice with the highest priority is considered. This function returns the weather or not agent was able to catch the prey and the total steps taken by the agent. **python agent.py –agent_2**
- agent_3: This function implements agent 3. The partial prey information setting is implemented in this agent. Initially the predator and agent location is known. At every timestamp a node is surveyed at random and the probability of the prey is tracked. Based on these probabilities agent calculates the distance from the highest probability node to the agent and makes a choice to move. This function returns weather or not the agent was able to catch the prey and total step taken by the agent. **python agent.py –agent_3**
- agent_4: This function implements agent 4. Agent 4 knows the location of the predator and surveys nodes for prey. At every timestamp a node is surveyed at random and the probability of the prey is tracked. Based on these probabilities agent calculates the weighted average distance from each agent choice that is not a neighbor of the node with predator, to the future location of the prey. Based on this a priority is assigned to each agent choice and the choice with the highest priority is considered. This function returns the weather or not agent was able to catch the prey and the total steps taken by the agent. **python agent.py –agent_4**
- agent_5: This function implements the agent 5. The partial predator information setting is implemented in this agent. Initially the prey and the agent location is known by this agent. At every timestamp a node is surveyed at random and the probability of the predator is tracked. Based on these probabilities agent 5 assigns priorities to each of the choices and the node with

the highest priority is chosen. This function returns the weather or not the agent was able to catch the prey, weather or not the predator caught the agent and total step taken by the agent. If the agent exceeds the maximum step count it would return 0. **python agent.py –agent_5**

- agent_6: This function implements the agent 6. The partial predator information setting is used for this agent. Initially the prey location is known. At every timestamp a node is surveyed at random and the probability of the predator is tracked. Based on these probabilities agent 6 calculates the average distance from each agent choice to the future locations of the prey and assigns priorities to each of the choices and the node with the highest priority is chosen. This function returns weather or not the agent was able to catch the prey, weather or not the predator caught the agent and total step taken by the agent. If the agent exceeds the maximum step count it would return 0. **python agent.py –agent_6**

- agent_7: This function implements the agent 7. Combined partial information setting is used for this agent. Neither predator not the prey location is known. At every timestamp a node is surveyed at random and the probability of the predator is tracked. If the predator's location is certain, a node survey for the prey's location. Based on these probabilities, priorities are assigned to each of the agent choice. The node with the highest priority is chosen by the agent. This function returns weather or not the agent was able to catch the prey, weather or not the predator caught the agent and total step taken by the agent. **python agent.py –agent_7**

- agent_8: This function implements the agent 8. Combined partial information setting is used for this agent. Neither predator not the prey location is known. At every timestamp a node is surveyed at random and the probability of the predator is tracked. If the predator's location is certain, a node survey for the prey's location. Based on these probabilities, weighted average distance from the highest probable prey location to the agent choices is calculated. Priorities are assigned to each of the agent choice and node with the highest priority is chosen. This function returns weather or not the agent was able to catch the prey, weather or not the predator caught the agent and total step taken by the agent. **python agent.py –agent_8**

6.1.3 Utils

utils.py includes the other utilities for the project.

- get_distance() : Calculates the distance between two nodes of the graph.
- move_predator() : This function takes in the current predator and agent location and outputs the next best predator location.
- move_prey() : This function takes the current prey location as the input, uniformly choices the next prey location from all the available choices and outputs this location.
- get_full_information_choice(): This function assigns priorities to all the possible agent choices and outputs the choice with highest priority. If there are multiple choices with same probability, the node is chosen uniformly at random.

  There are more functions defined for various other purposes like choosing which node to survey, to update probabilities of the prey and predator, to derive utility and cost etc. etc.

7 Grey Box questions :

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?
   We can at most add only 25 edges. This is because of 2 reasons, first we can only add an edge to a vertex with degree less than 3. from the initial setup of the graph each vertex already has 2 edges i.e. the degree of each vertex is 2, meaning we can only add one more edge to each vertex as the maximum degree can be 3. Secondly, on adding one edge the degree for two vertices are changed from 2 to 3. Since we have 50 vertices, we can at most add only $50/2 = 25$ edges.

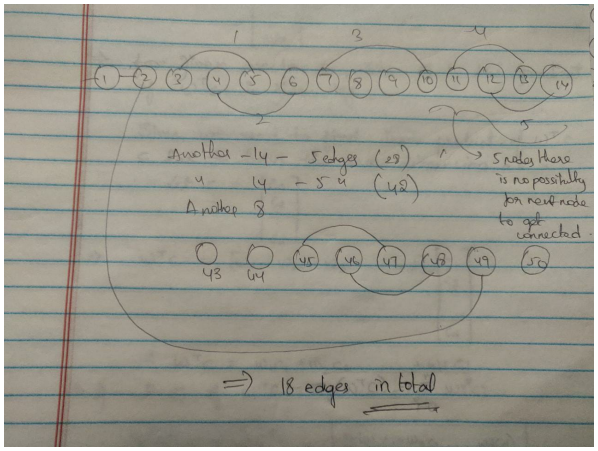


Figure 4: manual working to find minimum edges

Are you always able to add this many?
No we are not always able to add 25 edges. Since we can only connect to +5 and -5 nodes, there may be scenarios, where a node degree would get compromised to 2 only.
If not, what's the smallest number of edges you're always able to add?
After simulating the environment for 10 million times, minimum number of edges seen is 23. Tried simulating the environment 100 million times, aborted after 5 hours of run time, still the minimum edges seen is 23. So with this graph being connected this way, minimum number of edges is 23 and the probability of a graph having less than 23 edges is almost 0 when generated. But when worked up manually we are able to figure out a graph where we are able to add 18 edges only. So minimum edges would be 18. Refer to figures 4 and 5.

2. The Partial Prey Information Setting: How should you be updating the probabilities of where the Prey is?
   It is described in the Agent 3 section.

3. The Partial Predator Information Setting: How do the probability updates for the Predator differ from the probability updates for the Prey?
   Belief state updates happen in the below scenarios:

1. When surveyed: In this scenario, the belief state update of Predator is the same as prey

2. When Agent moves: In this scenario as well belief state update of the predator is the same as the prey

3. When the object moves: In this scenario, belief state updates of prey differ from the predator. In the case of prey, the probability of the node is distributed uniformly between it's neighbors and itself(described in Agent 3 section). In the case of the predator, the probability of the node is distributed in 0.6-0.4 weightage(described in Agent 5 section)

4. Some questions to consider when contemplating your own Agents: The below questions differ from agent to agent, based on the environment they run in, based on the information they have. It have been adressed in the respective agent sections.

   - What node should the Agent move towards, given its current information? - Generally, any agent tries to run away from the predator and trying to get towards the prey. The predator and prey locations is subjective to the environment they are present in.
   - What node should the Agent survey, given its current information? A shortcut answer would be
     - In case of prey: The node with the highest probability(breaking ties at random)
     - In case of predator: The node with the highest probability(breaking ties based on proximity first and random next)

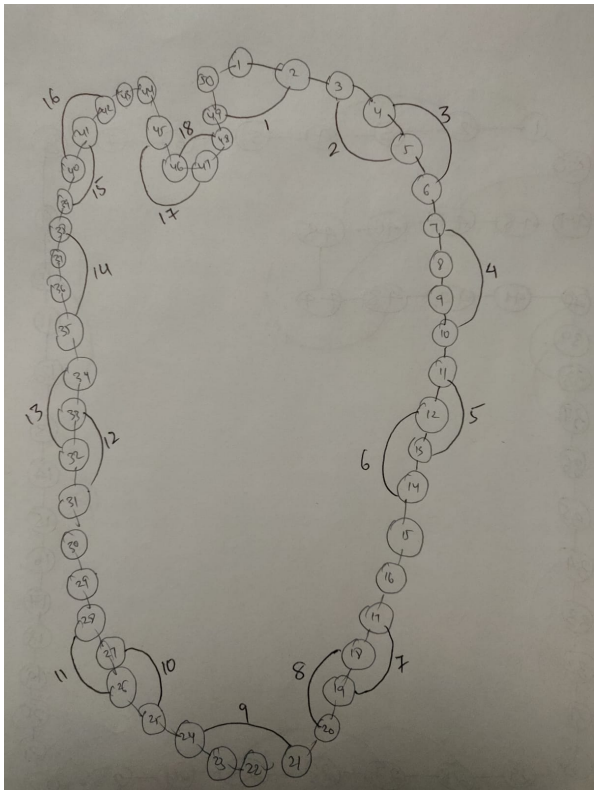


Figure 5: Graph with minimum edges.

- How best can the Agent use all the knowledge it has available to it to make a decision? - Differs from agent to agent

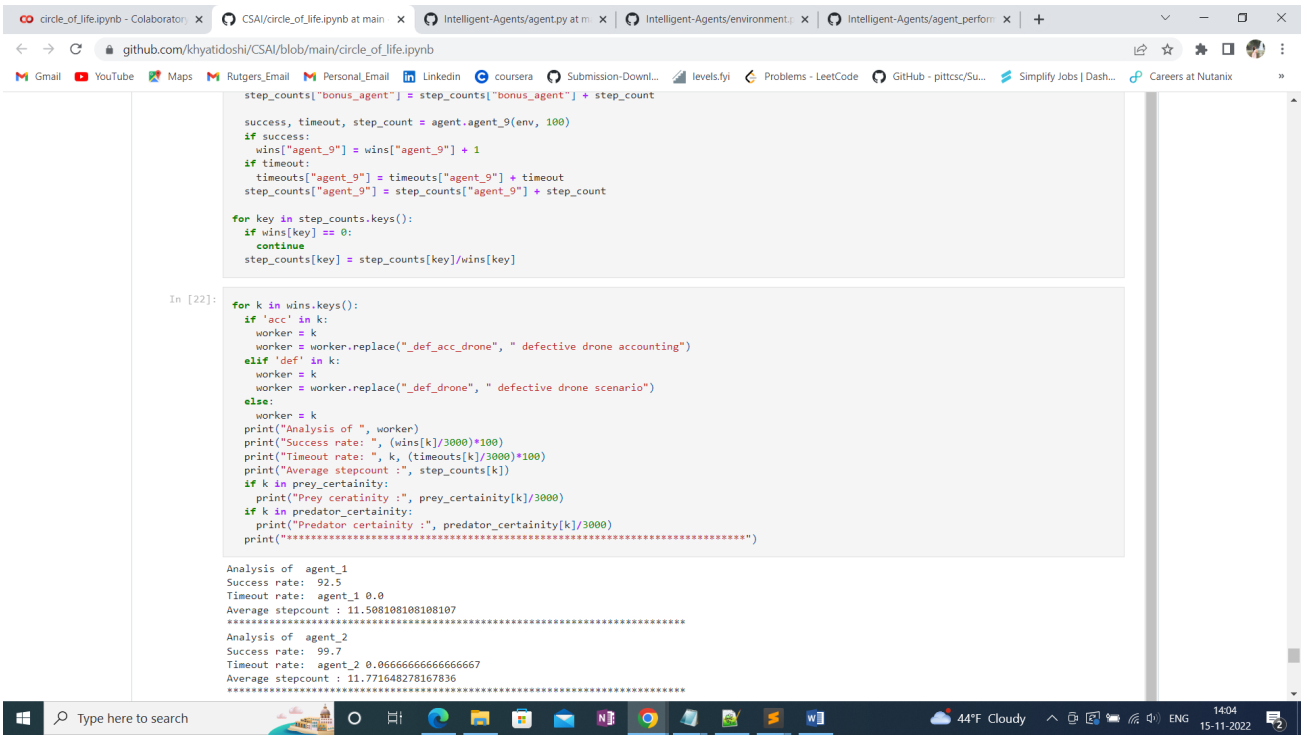
8 Summary

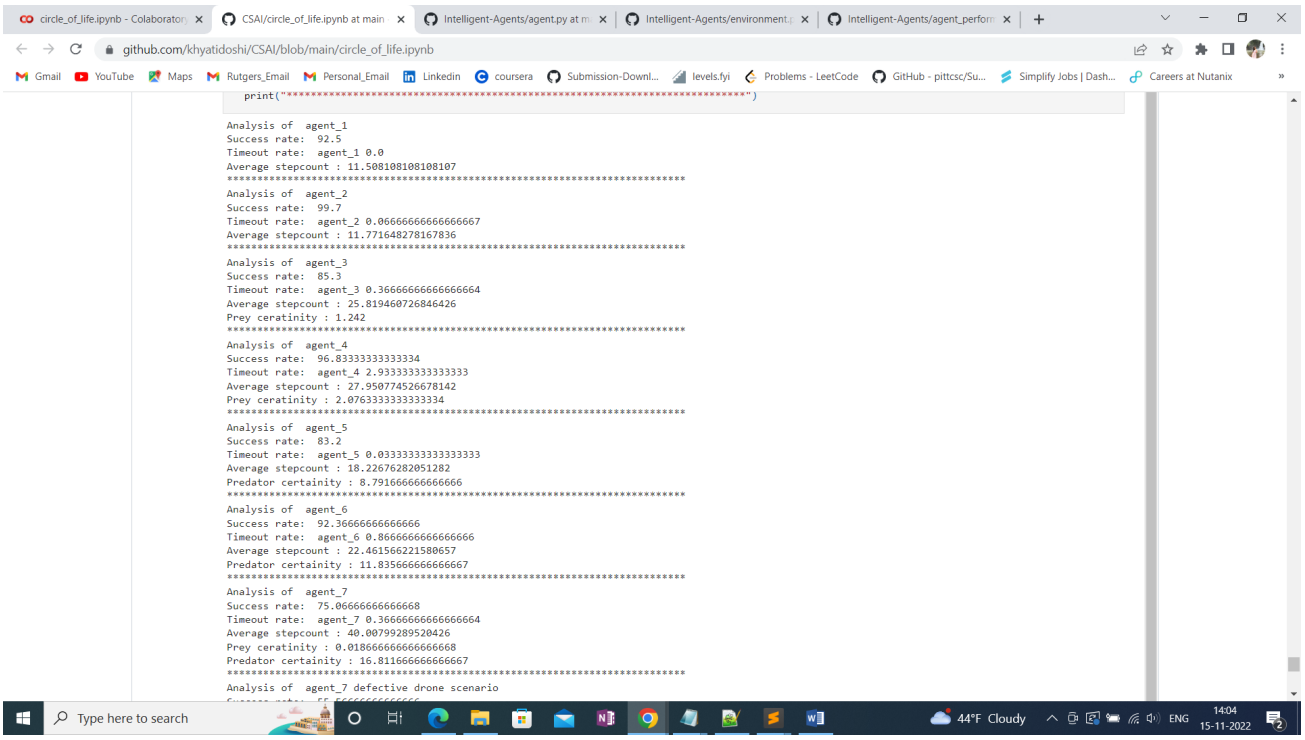
To conclude the survivability of agents looks like the following:

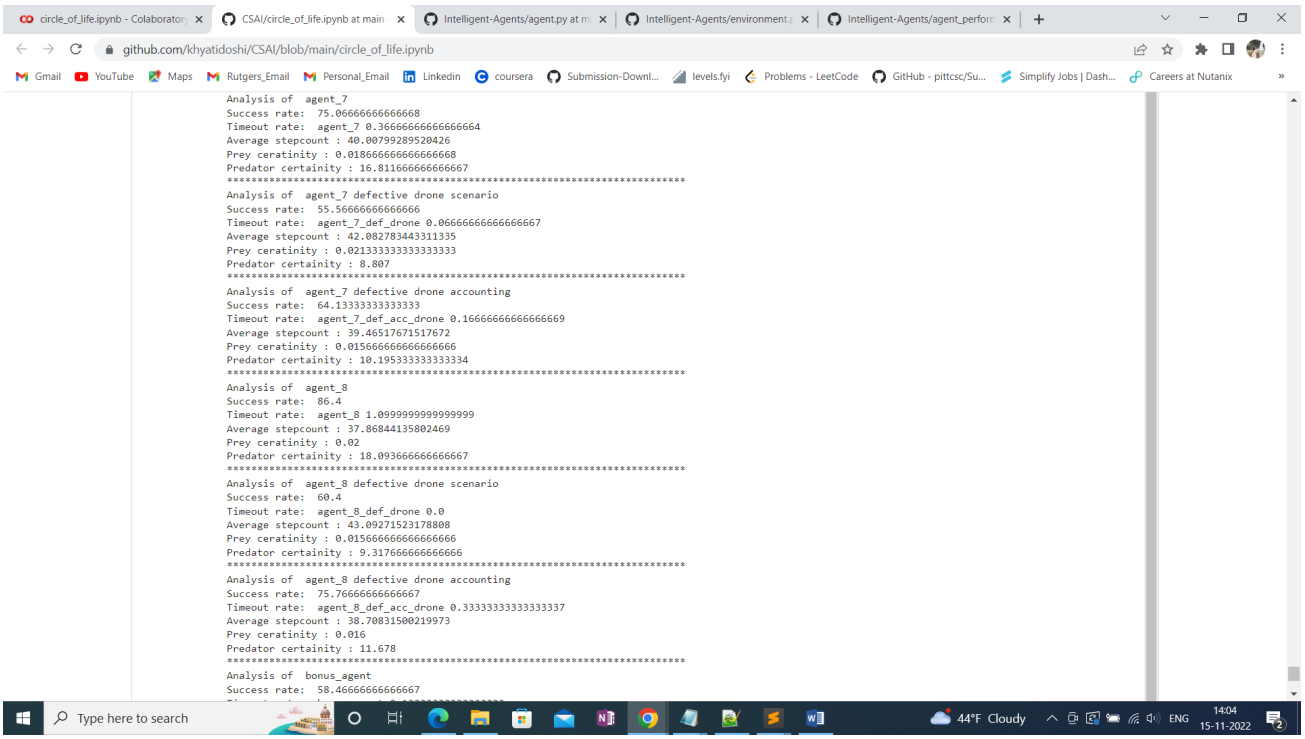
```
Agent 2 > Agent 4 > Agent 1 = Agent 6 > Agent 8 > Agent 3 > Agent 5 > Agent 7 > bonus Agent
In case of defective Drone scenarios and accounting:
    Agent 9 > Agent 8 > Agent 7
```

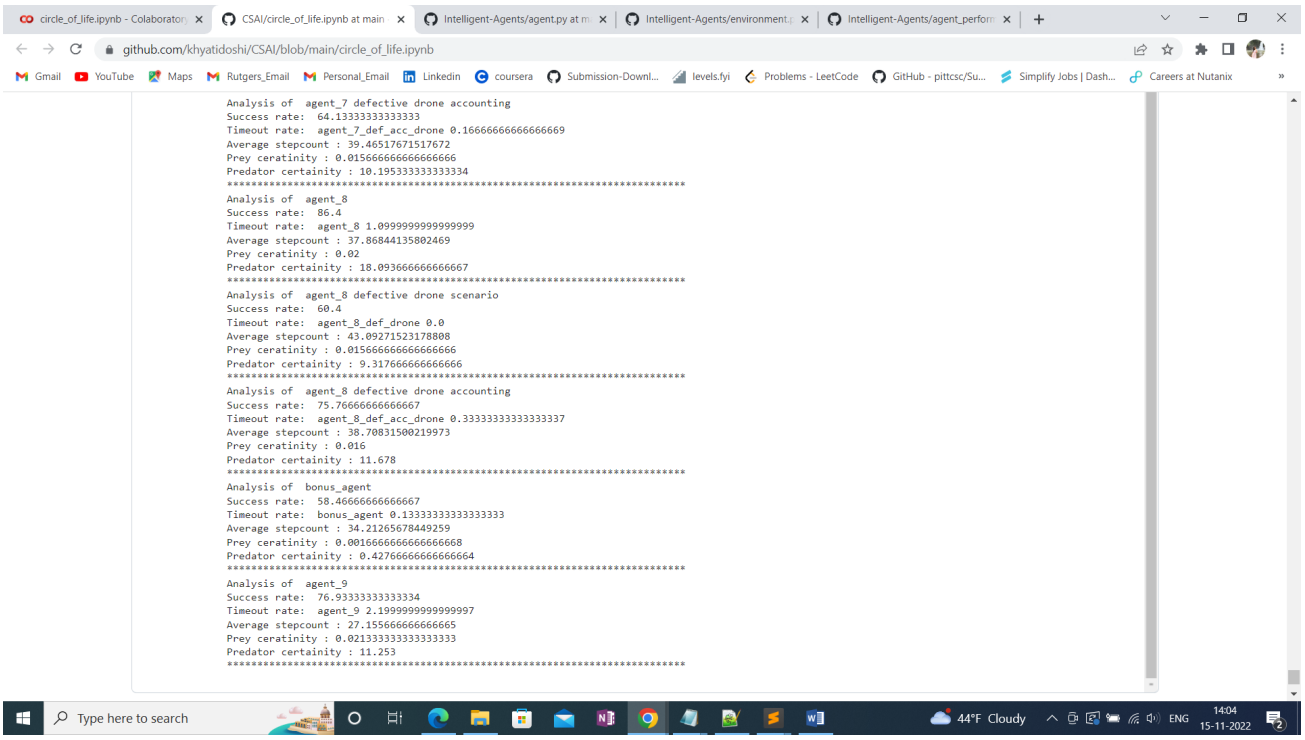
To conclude about Agent's play, Agents in the way of escaping from the predator is catching the prey, since if we can notice Agents(especially in case of partial information) knows very little information about prey.

9 Results


circle_of_life.ipynb - Colaborator
CSAI/circle_of_life.ipynb at main
Intelligent-Agents/agent.py at m
Intelligent-Agents/environment.
Intelligent-Agents/agent_perform
github.com/khyatidoshi/CSAI/blob/main/circle_of_life.ipynb
Gmail YouTube Maps Rutgers_Email Personal_Email Linkedin coursera Submission-Downl... levels.fyi Problems - LeetCode GitHub - pittcsc/Su... Simplify Jobs | Dash... Careers at Nutanix
step_counts["bonus_agent"] = step_counts["bonus_agent"] + step_count
success, timeout, step_count = agent.agent_9(env, 100)
if success:
wins["agent_9"] = wins["agent_9"] + 1
if timeout:
timeouts["agent_9"] = timeouts["agent_9"] + timeout
step_counts["agent_9"] = step_counts["agent_9"] + step_count
for key in step_counts.keys():
if wins[key] == 0:
continue
step_counts[key] = step_counts[key]/wins[key]
In [22]:
for k in wins.keys():
if 'acc' in k:
worker = k
worker = worker.replace("_def_acc_drone", " defective drone accounting")
elif 'def' in k:
worker = k
worker = worker.replace("_def_drone", " defective drone scenario")
else:
worker = k
print("Analysis of ", worker)
print("Success rate: ", (wins[k]/3000)*100)
print("Timeout rate: ", k, (timeouts[k]/3000)*100)
print("Average stepcount :", step_counts[k])
if k in prey_certainity:
print("Prey ceratinity :", prey_certainity[k]/3000)
if k in predator_certainity:
print("Predator certainity :", predator_certainity[k]/3000)
print("**********")
Analysis of agent_1
Success rate: 92.5
Timeout rate: agent_1 0.0
Average stepcount : 11.508108108108107
Analysis of agent_2
Success rate: 99.7
Timeout rate: agent_2 0.06666666666666667
Average stepcount : 11.771648278167836
Type here to search
44°F Cloudy
ENG
14:04
15-11-2022


circle_of_life.ipynb - Colaborator
CSAI/circle_of_life.ipynb at main
Intelligent-Agents/agent.py at m
Intelligent-Agents/environment.
Intelligent-Agents/agent_perform
github.com/khyatidoshi/CSAI/blob/main/circle_of_life.ipynb
Gmail YouTube Maps Rutgers_Email Personal_Email Linkedin coursera Submission-Downl... levels.fyi Problems - LeetCode GitHub - pittcsc/Su... Simplify Jobs | Dash... Careers at Nutanix
print("**********")
Analysis of agent_1
Success rate: 92.5
Timeout rate: agent_1 0.0
Average stepcount : 11.508108108108107
Analysis of agent_2
Success rate: 99.7
Timeout rate: agent_2 0.06666666666666667
Average stepcount : 11.771648278167836
Analysis of agent_3
Success rate: 85.3
Timeout rate: agent_3 0.36666666666666664
Average stepcount : 25.819460726846426
Prey ceratinity : 1.242
Analysis of agent_4
Success rate: 96.83333333333334
Timeout rate: agent_4 2.933333333333333
Average stepcount : 27.950774526678142
Prey ceratinity : 2.0763333333333334
Analysis of agent_5
Success rate: 83.2
Timeout rate: agent_5 0.03333333333333333
Average stepcount : 18.22676282051282
Predator certainity : 8.791666666666666
Analysis of agent_6
Success rate: 92.36666666666666
Timeout rate: agent_6 0.8666666666666666
Average stepcount : 22.461566221580657
Predator certainity : 11.835666666666667
Analysis of agent_7
Success rate: 75.06666666666668
Timeout rate: agent_7 0.36666666666666664
Average stepcount : 40.00799289520426
Prey ceratinity : 0.018666666666666668
Predator certainity : 16.811666666666667
Analysis of agent_7 defective drone scenario
Type here to search
44°F Cloudy
ENG
14:04
15-11-2022


circle_of_life.ipynb - Colaborator
CSAI/circle_of_life.ipynb at main
Intelligent-Agents/agent.py at m
Intelligent-Agents/environment.
Intelligent-Agents/agent_perform
github.com/khyatidoshi/CSAI/blob/main/circle_of_life.ipynb
Gmail YouTube Maps Rutgers_Email Personal_Email Linkedin coursera Submission-Downl... levels.fyi Problems - LeetCode GitHub - pittcsc/Su... Simplify Jobs | Dash... Careers at Nutanix
Analysis of agent_7
Success rate: 75.06666666666668
Timeout rate: agent_7 0.36666666666666664
Average stepcount : 40.00799289520426
Prey ceratinity : 0.018666666666666668
Predator certainity : 16.811666666666667
****
Analysis of agent_7 defective drone scenario
Success rate: 55.56666666666666
Timeout rate: agent_7_def_drone 0.06666666666666667
Average stepcount : 42.082783443311335
Prey ceratinity : 0.021333333333333333
Predator certainity : 8.807
****
Analysis of agent_7 defective drone accounting
Success rate: 64.13333333333333
Timeout rate: agent_7_def_acc_drone 0.16666666666666669
Average stepcount : 39.46517671517672
Prey ceratinity : 0.015666666666666666
Predator certainity : 10.195333333333334
****
Analysis of agent_8
Success rate: 86.4
Timeout rate: agent_8 1.0999999999999999
Average stepcount : 37.86844135802469
Prey ceratinity : 0.02
Predator certainity : 18.093666666666667
****
Analysis of agent_8 defective drone scenario
Success rate: 60.4
Timeout rate: agent_8_def_drone 0.0
Average stepcount : 43.09271523178808
Prey ceratinity : 0.015666666666666666
Predator certainity : 9.317666666666666
****
Analysis of agent_8 defective drone accounting
Success rate: 75.76666666666667
Timeout rate: agent_8_def_acc_drone 0.33333333333333337
Average stepcount : 38.70831500219973
Prey ceratinity : 0.016
Predator certainity : 11.678
****
Analysis of bonus_agent
Success rate: 58.46666666666667
Type here to search
44°F Cloudy
ENG
14:04
15-11-2022


circle_of_life.ipynb - Colaborator
CSAI/circle_of_life.ipynb at main
Intelligent-Agents/agent.py at m
Intelligent-Agents/environment.
Intelligent-Agents/agent_perform
github.com/khyatidoshi/CSAI/blob/main/circle_of_life.ipynb
Gmail YouTube Maps Rutgers_Email Personal_Email Linkedin coursera Submission-Downl... levels.fyi Problems - LeetCode GitHub - pittcsc/Su... Simplify Jobs | Dash... Careers at Nutanix
Analysis of agent_7 defective drone accounting
Success rate: 64.13333333333333
Timeout rate: agent_7_def_acc_drone 0.16666666666666669
Average stepcount : 39.46517671517672
Prey ceratinity : 0.015666666666666666
Predator certainity : 10.195333333333334
****
Analysis of agent_8
Success rate: 86.4
Timeout rate: agent_8 1.0999999999999999
Average stepcount : 37.86844135802469
Prey ceratinity : 0.02
Predator certainity : 18.093666666666667
****
Analysis of agent_8 defective drone scenario
Success rate: 60.4
Timeout rate: agent_8_def_drone 0.0
Average stepcount : 43.09271523178808
Prey ceratinity : 0.015666666666666666
Predator certainity : 9.317666666666666
****
Analysis of agent_8 defective drone accounting
Success rate: 75.76666666666667
Timeout rate: agent_8_def_acc_drone 0.33333333333333337
Average stepcount : 38.70831500219973
Prey ceratinity : 0.016
Predator certainity : 11.678
****
Analysis of bonus_agent
Success rate: 58.46666666666667
Timeout rate: bonus_agent 0.13333333333333333
Average stepcount : 34.21265678449259
Prey ceratinity : 0.0016666666666666668
Predator certainity : 0.4276666666666664
****
Analysis of agent_9
Success rate: 76.93333333333334
Timeout rate: agent_9 2.1999999999999997
Average stepcount : 27.155666666666665
Prey ceratinity : 0.021333333333333333
Predator certainity : 11.253
****
Type here to search
44°F Cloudy
ENG
14:04
15-11-2022