Probabilistic Pursuit: Adaptive Decision Making in Dynamic Environments

Himanshu Pahwa

1 The Environment

Q)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 have a total of 50 nodes and we need two nodes to form any edge. After forming 25 edges, the degree of all the nodes in the graph would be 3 which suggests that no new edges can be added. By experimentation, we always had at least 23 edges added in the graph. One possible explanation for this may be that at some point all the edges on the right or left side of a node may be occupied, leaving no room for another edge. This is usually very random though.

For the environment, we created a graph using a dictionary with the values of each key representing the neighbors. We initialize with every node having a degree 2. Then we selected a node from +- 5 nodes of that node, made sure it was of degree two and added an edge from that node to the current node. We repeated this process until we couldn't find a node with degree two.

3 Agents and Environments

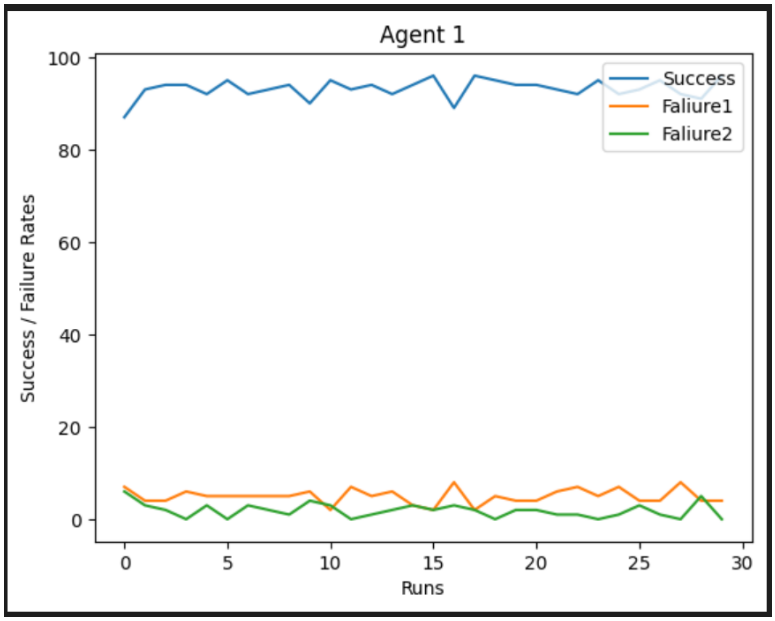
The results given below are denoted by the following

- How often the Predator catches the Agent - **Failure 1**
- How often the Agent catches the Prey - **Success**
- How often the simulation hangs - **Failure 2**
- How often the Agent knows exactly where the Prey is during a simulation where that information is partial. **Prey Confidence rate**
- How often the Agent knows exactly where the Predator is during a simulation where that information is partial **Pred Confidence Rate**

All our probability functions were verified as their sums for all the agents were amounting to 1.

Agent 1

Our agent moves according to the conditions mentioned in the assignment. After testing all the conditions, if there is nowhere to go, it sits still and prays. The observations for agent 1 are given below:



Average Observations(150 steps)

Success: 93.16%

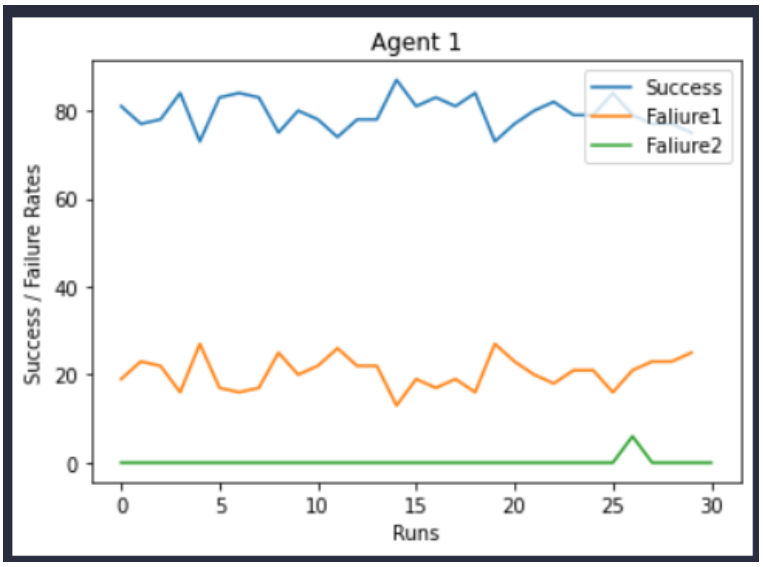
Failure 1: 4.96%

Failure 2: 1.86%

Std. Deviation Success: 2.07

Std. Deviation Failure 1: 1.63

Std. Deviation Failure 2: 1.55



Average Observations(5000 steps)

Success: 79.46%

Failure 1: 20.53%

Failure 2: 0.0%

Std. Deviation Success: 3.6

Std. Deviation Failure 1: 3.6

Std. Deviation Failure 2: 1.07

Agent 2

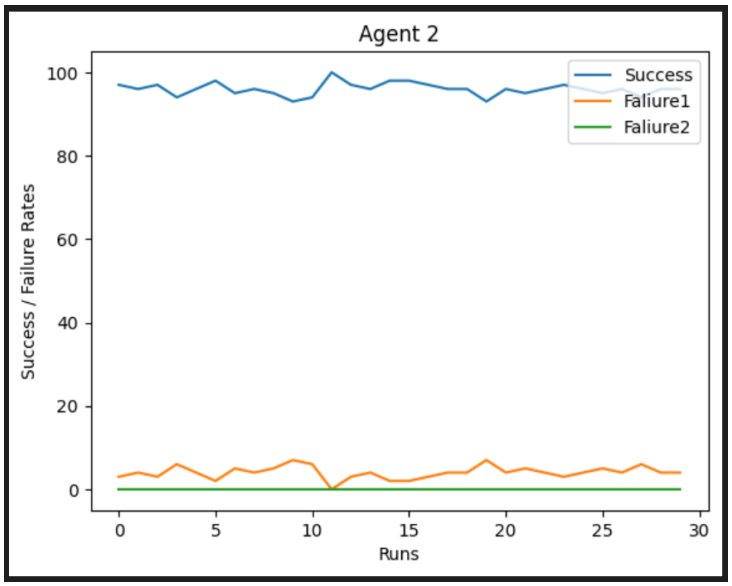
We had to make agent 2 better than agent 1. After thinking a lot we realized that agent one focuses equally on prey as well as predator. Logically, to drive the success rate up, we had to increase the number of times the agent caught the prey, at the same time making sure that there is no danger from the predator.

We took the radius approach for the predator which means we did not care about our distance from the predator unless it's within a distance of the agent. The best radius for this was r = 5.

If (r>5) -> Just follow the prey, and take a step from where the prey is closer

Else if (r<=5) -> Follow the rules of agent one

This increased our survivability dramatically as well as reduced the rate of failure 2, because the agent is rushing to catch the prey. Given below are some results:



Average Observations(150 steps)

Success: 95.96%

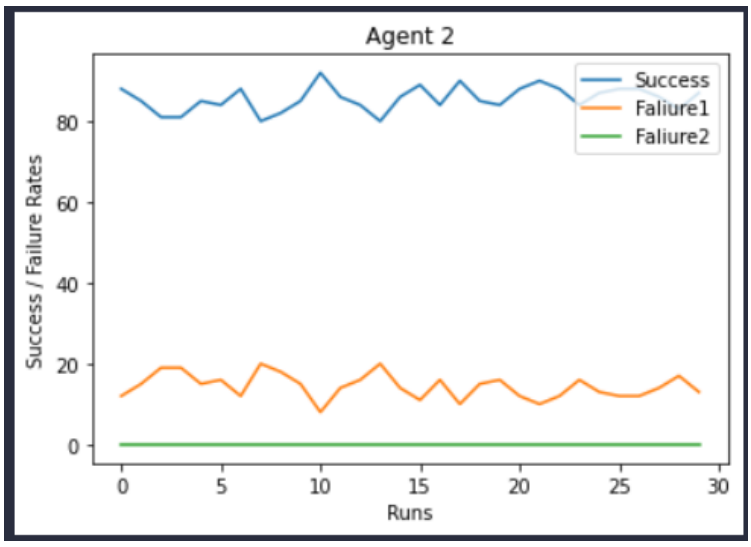
Failure 1: 4.03%

Failure 2: 0.0%

Std. Deviation Success: 1.52

Std. Deviation Failure 1: 1.52

Std. Deviation Failure 2: 0



Average Observations(5000 steps)

Success: 85.6%

Failure 1: 14.40%

Failure 2: 0.0%

Std. Deviation Success: 3.04

Std. Deviation Failure 1: 3.04

Std. Deviation Failure 2: 0

Q) In these cases, how should you be updating the probabilities of where the Prey is?

```
--> At the start

    P(0,N) = 1/49, for all N not containig the Agent
           = 0, if N contained the Agent

--> When Agent surveys at time t

    Surveyed a node X

    Case 1: Prey is not found at node X

        P(prey is at node N | surevey did not find prey at X) = P(Prey is at N)/P(Prey is not at X), for all N != X
                                                              = 0, if N == X

    Case 2: Prey is found at node X

            P(Prey is at node N | survey finds prey at X) = 0, for all N != X
                                                          = 1, if N == X

--> When Agent moves at time t

    Case 1: Prey is not found at node X

            P(Prey is at node N | prey not found at X) = P(Prey is at Node N)/P(Prey is not at X), for all N != X
                                                       = 0, if N == X

    Case 2: Prey is found at X

            P(Prey is at Node N | Prey found at X) = 0, for all N != X
                                                   = 1, if N == X

--> When Prey Moves at time (t+1)

    P(Prey is at N Now) =
    Summation(i = 1 to 50) P(Prey is at N Now and prey was at i then)
    = P(Prey was at N then and Prey is at N Now) + summation(i=1 to 50 and i != N) P(Prey was at I then) * P(Prey is at N now | Prey was at i then)
```

Agent 3

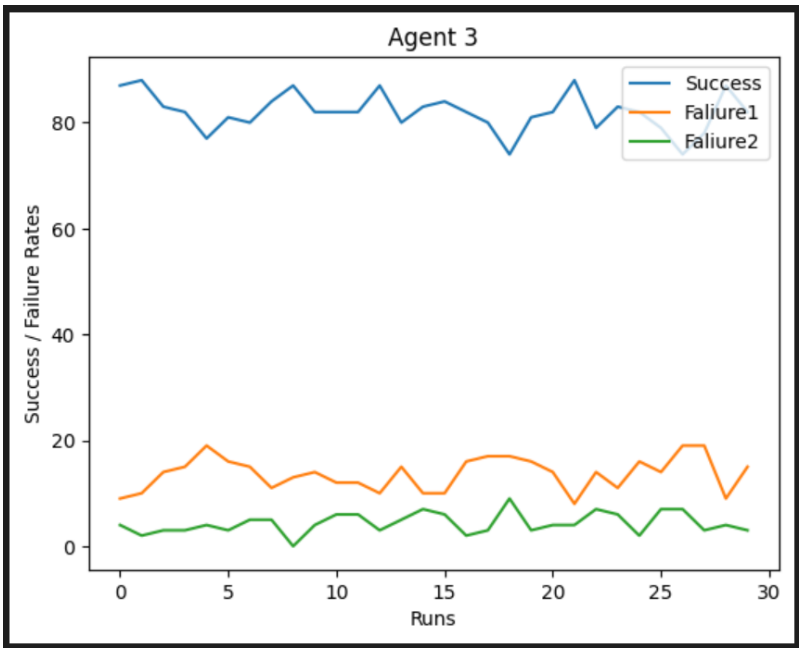
For agent 3 we didn't know the position of the prey. We had to collect information and assign probabilities based on that information to all the nodes in the graph. The information was collected in two ways - Surveying the node with the highest probability of the prey, and when the agent moves into a new node and checking whether the prey is there or not. After each of these moves, we had to update the probability with the function given below.

```
def prey_prob_updates_Agent(prob_dic,prey_found,surveyed_node):
    prob_dic_update = {}
    for nodes in prob_dic:
        if prey_found == False: #* Case 1
            if nodes != surveyed_node:
                prob_dic_update[nodes] = prob_dic[nodes] / (1 - prob_dic[surveyed_node])
            else:
                prob_dic_update[nodes] = 0
        elif prey_found == True: #* Case 2
            if nodes != surveyed_node:
                prob_dic_update[nodes] = 0
            else:
                prob_dic_update[nodes] = 1

    #prob_dic = prob_dic_update
    return prob_dic_update
```

Then, after the prey moves, we are required to update the probability at time t+1 knowing what we know at the present moment. The function for this is as below:

```
def prey_prob_updates_prey(prob_dic,graph): #* When prey moves, ie. t+1
    prob_dic_update = {}
    for nodes in prob_dic:
        prob_update = 0
        for sub_nodes in prob_dic:
            if sub_nodes == nodes:
                prob_update +=  prob_dic[sub_nodes] * (1/(len(graph[sub_nodes])+1)) # was 0
            else:
                if nodes in graph[sub_nodes]:
                    prob_update += prob_dic[sub_nodes] * (1/(len(graph[sub_nodes])+1))
                else:
                    prob_update += 0
        prob_dic_update[nodes] = prob_update

    return prob_dic_update
```

Results for agent 3:-



Average Observations(150 steps)

Success: 81.7%

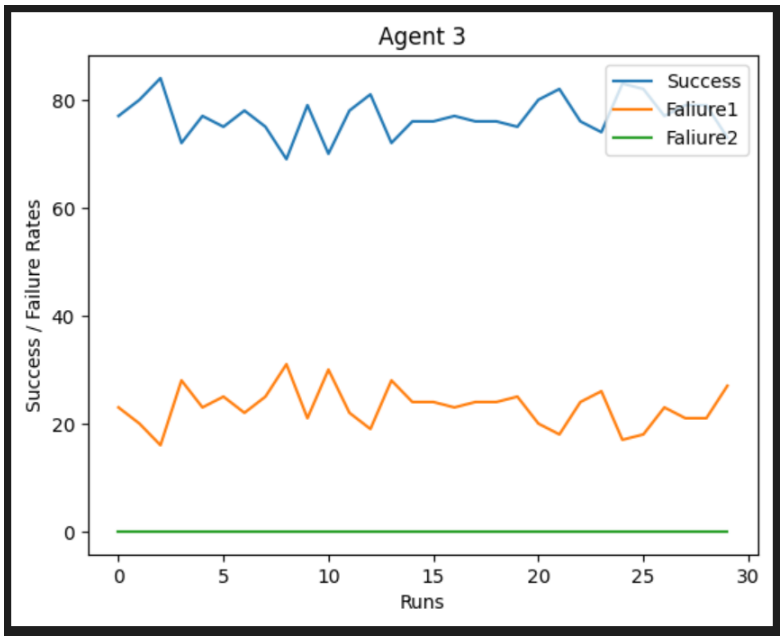
Failure 1: 13.9%

Failure 2: 4.3%

Std. Deviation Success: 3.64

Std. Deviation Failure 1: 3.14

Std. Deviation Failure 2: 1.97



Average Observations(5000 steps)

Success: 76.9%

Failure 1: 23.06%

Failure 2: 0.0%

Prey Confidence Rate: 9.9

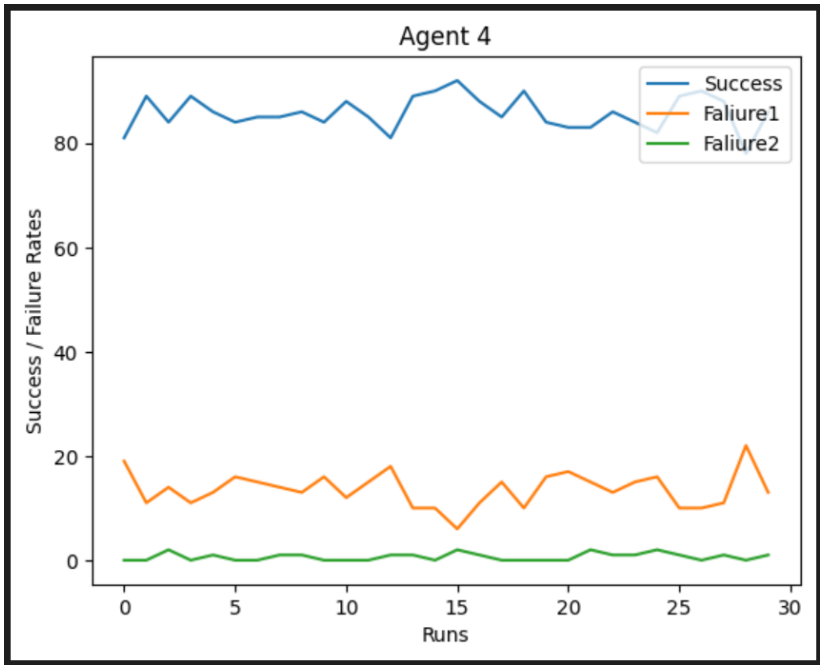
Std. Deviation Success: 3.65

Std. Deviation Failure 1: 3.65

Std. Deviation Failure 2: 0

Agent 4

Agent 4 is a better agent 3. Even though it uses the same probability functions. The agent moves according to the rules of agent 2, giving it a leverage over agent 3 which uses the rules of agent 1. The results corroborate with the expected observation.



Average Observations(150 steps)

Success: 85.8%

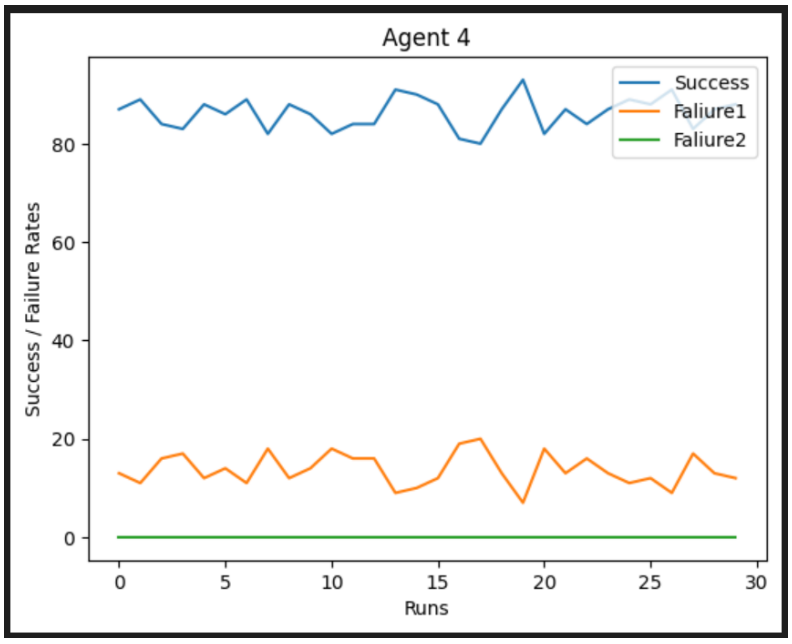
Failure 1: 13.56%

Failure 2: 0.63%

Std. Deviation Success: 3.23

Std. Deviation Failure 1: 3.31

Std. Deviation Failure 2: 0.72



Average Observations (5000 steps)

Success: 86.26%

Failure 1: 13.7%

Failure 2: 0.0%

Prey Confidence Rate:8.28%

Std. Deviation Success: 3.25

Std. Deviation Failure 1: 3.25

Std. Deviation Failure 2: 0

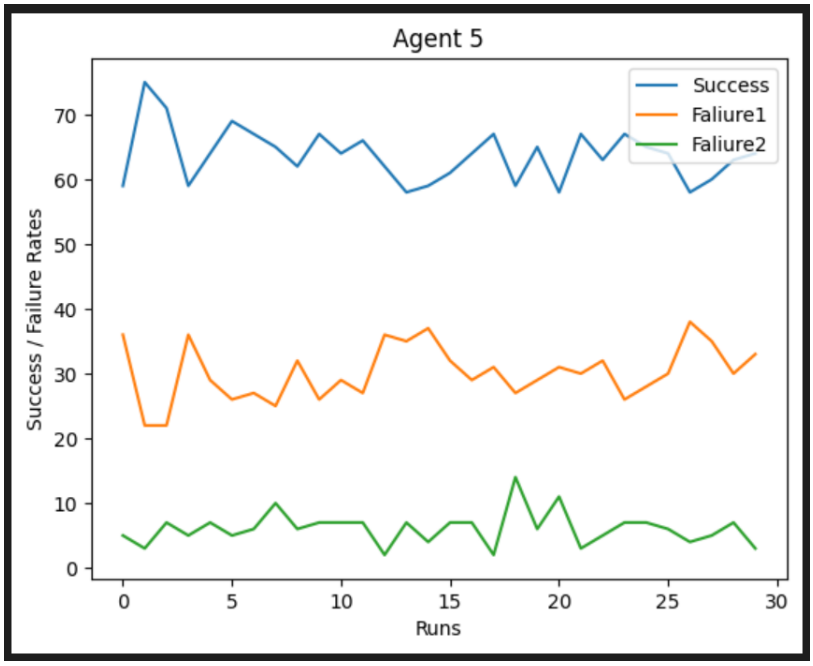
Q) How do the probability updates for the Predator differ from the probability updates for the Prey? Be explicit and clear.

```
--> At the start

P(0,N)  = 0, for all N != Predator position
        = 1, if N == Predator position

--> When Agent surveys at time t

    Surveyed a node X

    Case 1: Predator is not found at node X

        P(Predator is at node N | surevey did not find Predator at X) = P(Predator is at N)/P(Predator is not at X), for all N != X
                                                                      = 0, if N == X

    Case 2: Predator is found at node X

            P(Predator is at node N | survey finds Predator at X) = 0, for all N != X
                                                                  = 1, if N == X

--> When Agent moves at time t

    Case 1: Predator is not found at node X

            P(Predator is at node N | Predator not found at X) = P(Predator is at Node N)/P(Predator is not at X), for all N != X
                                                               = 0, if N == X

    Case 2: Predator is found at X

            P(Predator is at Node N | Predator found at X) = 0, for all N != X
                                                           = 1, if N == X
```

```
--> When Predator moves at time (t+1)

   P(Predator is at Node N now)
   = summation(i=1 to 50) P(Predator is at N now and Predator was at i then)
   = summation(i=1 to 50) P(Predator is at i then and Predator was at n now)
   = P(Predator was at N then and Predator is at N now) + summation(i=1 to 50, i != N) P(Predator was at i then) * P(Predator is at N now | Predator was at i then)
     ------------------------------------------------
                        ↓
                        0
```

Agent 5

For agent 5 we didn't know the position of the pred. We had to collect information and assign probabilities based on that information to all the nodes in the graph. The information was collected in two ways - Surveying the node with the highest probability of the predator, and when the agent moves into a new node and checking whether the predator is there or not. After each of these moves, we had to update the probability with the function given below.



Average Observations(150 steps)

Success: 63.73%

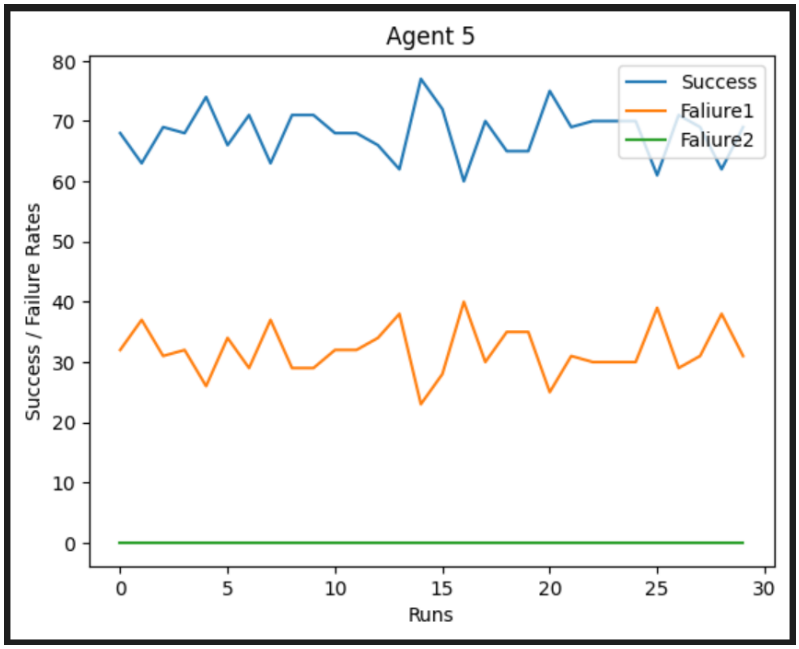
Failure 1: 30.2%

Failure 2: 6.06%

Std. Deviation Success: 4.1

Std. Deviation Failure 1: 4.27

Std. Deviation Failure 2: 2.55



Average Observations(5000 steps)

Success: 68.1%

Failure 1: 31.9%

Failure 2: 0.0%

Pred Confidence Rate:45.5%

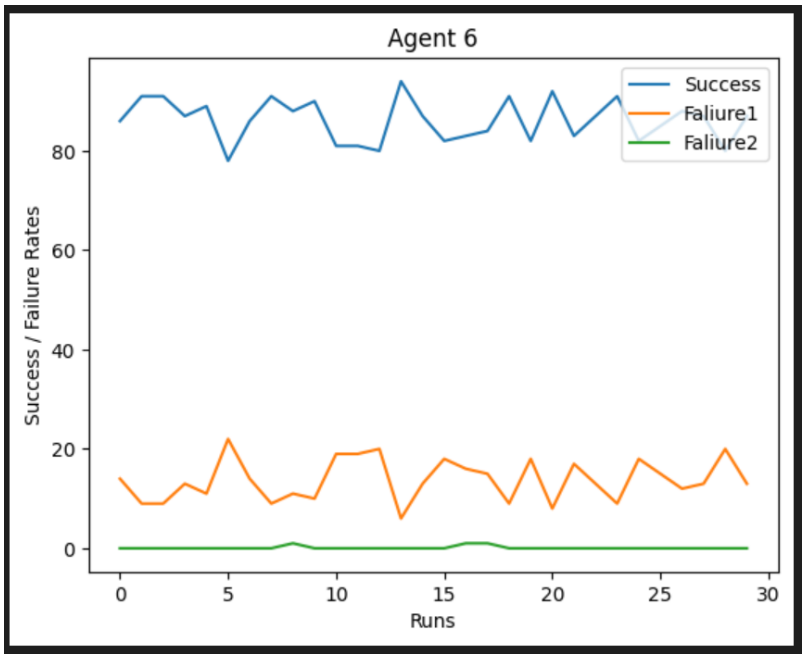
Std. Deviation Success: 4.14

Std. Deviation Failure 1: 4.14

Std. Deviation Failure 2: 0

Agent 6

Agent 6 is a better agent 5. Even though it uses the same probability functions. The agent moves according to the rules of agent 2, giving it a leverage over agent 5 which uses the rules of agent 1. The results corroborate with the expected observation.



Average Observations(150 steps)

Success: 86.14%

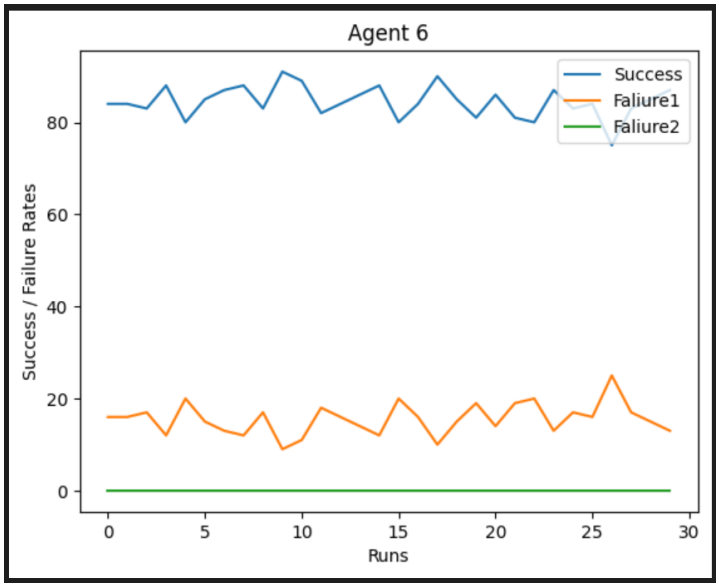
Failure 1: 13.76%

Failure 2: 0.1%%

Std. Deviation Success: 4.23

Std. Deviation Failure 1: 4.21

Std. Deviation Failure 2: 0.3



Average Observations(5000 steps)

Success: 84.43%

Failure 1: 15.56%

Failure 2: 0.1%%

Pred Confidence Rate: 49.36

Std. Deviation Success: 3.45

Std. Deviation Failure 1: 3.45

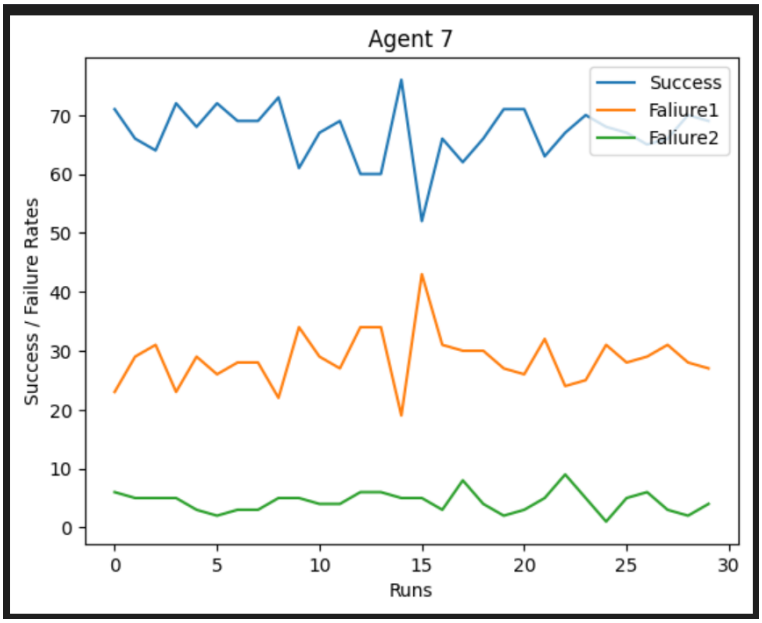
Std. Deviation Failure 2: 0

The Combined Partial Information Setting:

```
Complete Partial

--> Probability update of prey from partial prey scenario
--> Probability update of predator from partial predator scenario
```

Agent 7

In agent 7, we don't know the position of the prey or the predator. We have to collect information for both predator and prey by surveying and receiving info after the agent takes a step. There is one condition though. Here we are prioritizing predator position. If we are atlease 50% sure about the position of the predator, we survey for the prey, otherwise we keep surveying for the predator. We also update the probability for both once the agent moves. After the prey and pred both move, we update the probability at time t+1 given the current information. The observations for this agent are given below.



Average Observations(150 steps)

Success: 67%

Failure 1: 28.6%

Failure 2: 4.4%

Std. Deviation Success: 4.8

Std. Deviation Failure 1: 4.51

Std. Deviation Failure 2: 1.75

Average Observations(5000 steps)

Success: 61.96%

Failure 1: 38.03%

Failure 2: 0.0%

Confidence Rate Prey,Pred:3.49%, 22.09%

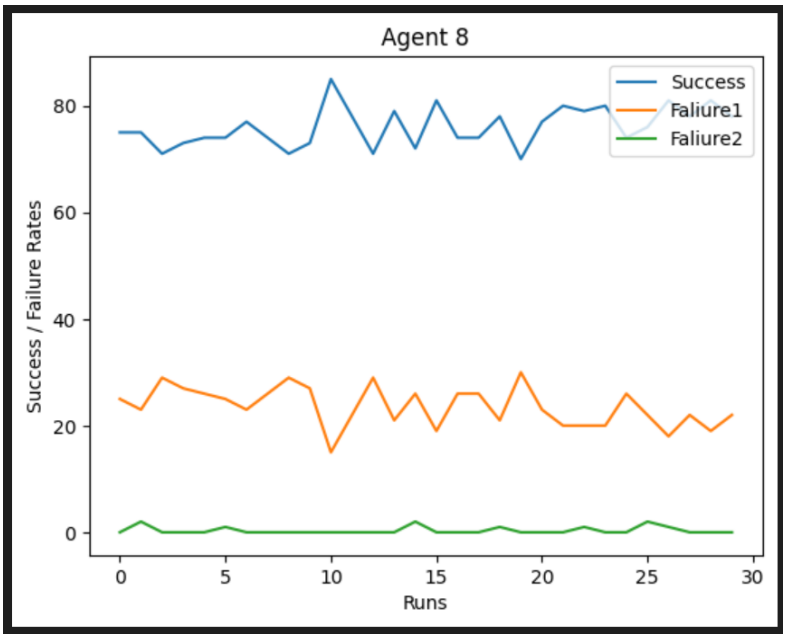
Std. Deviation Success: 4.73

Std. Deviation Failure 1: 4.73

Std. Deviation Failure 2: 0

Agent 8

Agent 8 is a better agent 7. Even though it uses the same probability functions. The agent moves according to the rules of agent 2, giving it a leverage over agent 7 which uses the rules of agent 1. The results corroborate with the expected observation.



Average Observations

Success: 79.4%

Failure 1: 21.9%

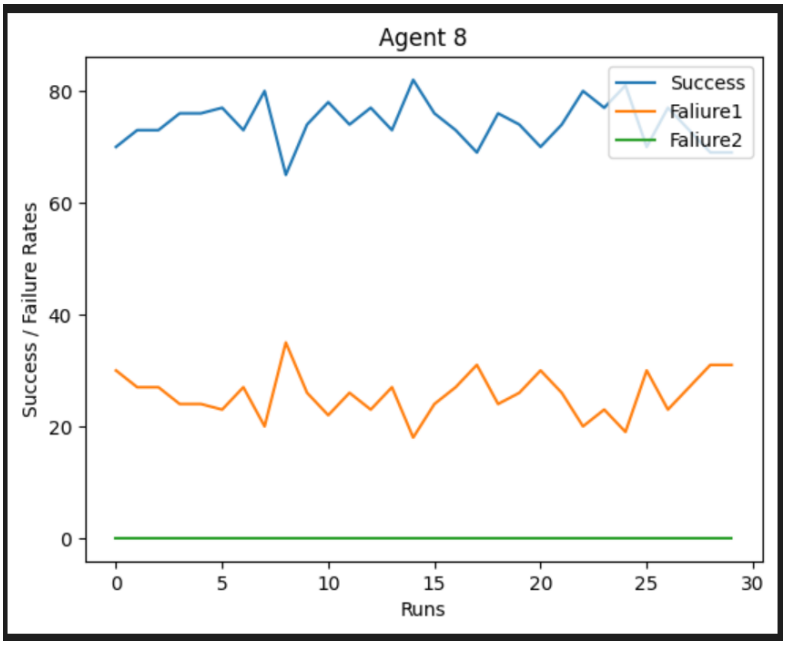
Failure 2: 0.34%

Confidence Rate:

Std. Deviation Success: 3.69

Std. Deviation Failure 1: 3.72

Std. Deviation Failure 2: 0.66



Average Observations

Success: 74.3%

Failure 1: 25.7%

Failure 2: 0.0%

Confidence Rate
Prey,Pred:4.3%, 21.36%

Std. Deviation Success: 3.98

Std. Deviation Failure 1: 3.98

Std. Deviation Failure 2: 0

Q) Some questions to consider, when contemplating your own Agents:

What node should the Agent move towards, given its current information?

Agent should move towards the node where the probability of the prey is the highest as long as the predator is outside the defined radius of the agent.

What node should the Agent survey, given its current information?

The agent should survey the node with the highest probability of the prey/pred being there.

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

Agent should utilize the probability information available based on the survey/move to make a decision

Q) 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?

Here, the success is implied by the Agent catching the prey, not just by the survivability of the agent i.e. the predator never catches the prey. To better the success rate of agent 2 and hence agents 4,6 and 8, we incline towards catching the prey as long as we are not 'reasonably threatened' by the predator. We use an approach where we set a radius around the agent. We assume the predator to be threatening if it's within the 5 nodes of the agent. In this case, agent 2 moves carefully as per the rules of agent 1. However when the predator is not seen within the radius of 5 nodes, the agent moves in the direction where distance to the prey is minimized.

We use the given information in the most efficient way. We know that the predator moves in a particular manner, one step at a time. As long as it's not within the radius of the agent, it discards that information and focuses on catching the prey. Not only does it increase the success rate but also reduces the failure 2 rate, greatly when the steps or time constraints are low.

For the Combined Partial Information Setting: imagine that your survey drone is defective, and that if something is actually occupying a node being surveyed, there is a 0.1 probability that it gets reported as unoccupied (a false negative).

- **How do Agents 7 and 8 compare in this setting, if you do not update your belief update rules to account for this?**

  Here, the probability won't add up to 1 because we're not accounting for the false negatives by updating the probabilities accordingly.

- **How should you update your belief update rules to account for this?**

```
Complete Partial defective drone

The only change happens when Agent surveys or moves and the dron gives false negative

--> When Agent surveys node X at time t

    Case 1: Prey not found

        P(Prey is at node N | surevey did not find Prey at X)    = (P(Prey is at N)*(0.1))/(1-0.9*P(Prey is at X)), for all N != X
                                                                 = (P(Prey is at N)*(0.1))/(1-0.9*P(Prey is at N)), if N == X

    >> Rest of the cases remain the same
```

- **How do Agents 7 and 8 compare in this setting, once belief update rules have been updated to account for this?**

  Belief updates will be in the following ways:

For P(t)

```python
#0.9 probability that the result of surveyed node is correct
def prey_prob_updates_Agent_modified(prob_dic,prey_found,surveyed_node):
    prob_dic_update = {}
    for nodes in prob_dic:
        if prey_found == False: #* Case 1
            if nodes != surveyed_node:
                prob_dic_update[nodes] = prob_dic[nodes] / (1 - 0.9*prob_dic[surveyed_node])# prob_dic[nodes] / (1 - prob_dic[surveyed_node])
            else:
                prob_dic_update[nodes] = 0.1*prob_dic[nodes]/(1-0.9*prob_dic[nodes])# 0
        elif prey_found == True: #* Case 2
            if nodes != surveyed_node:
                prob_dic_update[nodes] = 0
            else:
                prob_dic_update[nodes] = 1

    #prob_dic = prob_dic_update
    return prob_dic_update
```

P(t+1)

```python
#0.9 probability that the result of surveyed node is correct
def prey_prob_updates_Agent_modified(prob_dic,prey_found,surveyed_node):
    prob_dic_update = {}
    for nodes in prob_dic:
        if prey_found == False: #* Case 1
            if nodes != surveyed_node:
                prob_dic_update[nodes] = prob_dic[nodes] / (1 - 0.9*prob_dic[surveyed_node])# prob_dic[nodes] / (1 - prob_dic[surveyed_node])
            else:
                prob_dic_update[nodes] = 0.1*prob_dic[nodes]/(1-0.9*prob_dic[nodes])# 0
        elif prey_found == True: #* Case 2
            if nodes != surveyed_node:
                prob_dic_update[nodes] = 0
            else:
                prob_dic_update[nodes] = 1

    #prob_dic = prob_dic_update
    return prob_dic_update
```

Average Observations(150 observations)

Success: 64.67%

Failure 1: 29.13%

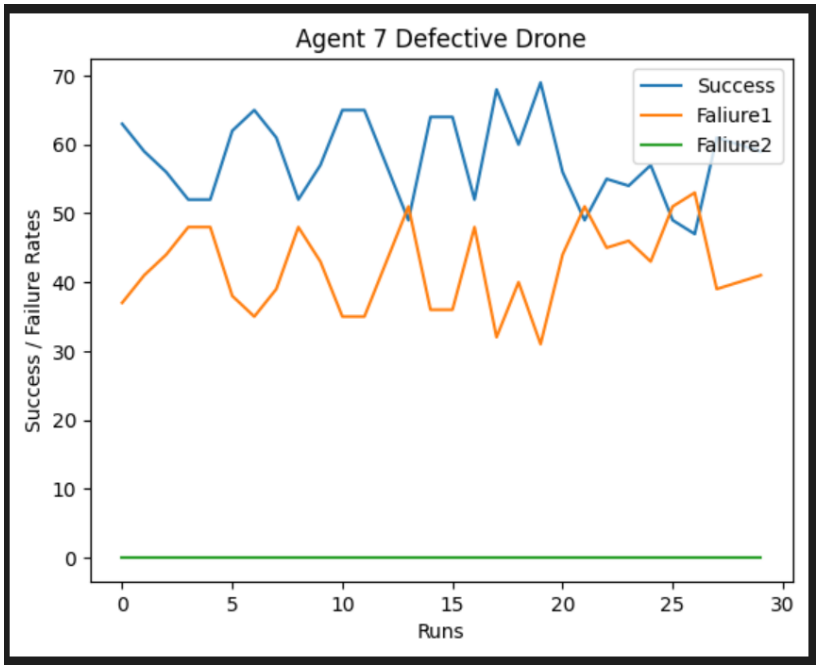
Failure 2: 6.2%

Confidence Rate:

Std. Deviation Success: 5.33

Std. Deviation Failure 1: 4.85

Std. Deviation Failure 2: 2.38



Average Observations(5000 steps)

Success: 57.9%

Failure 1: 42.1%

Failure 2: 0.0%

Confidence Rate Prey,Pred:2.9%, 17.8%

Std. Deviation Success: 6.02

Std. Deviation Failure 1: 6.02

Std. Deviation Failure 2:0

Average Observations(150 steps)

Success: 69.0%

Failure 1: 30.7%

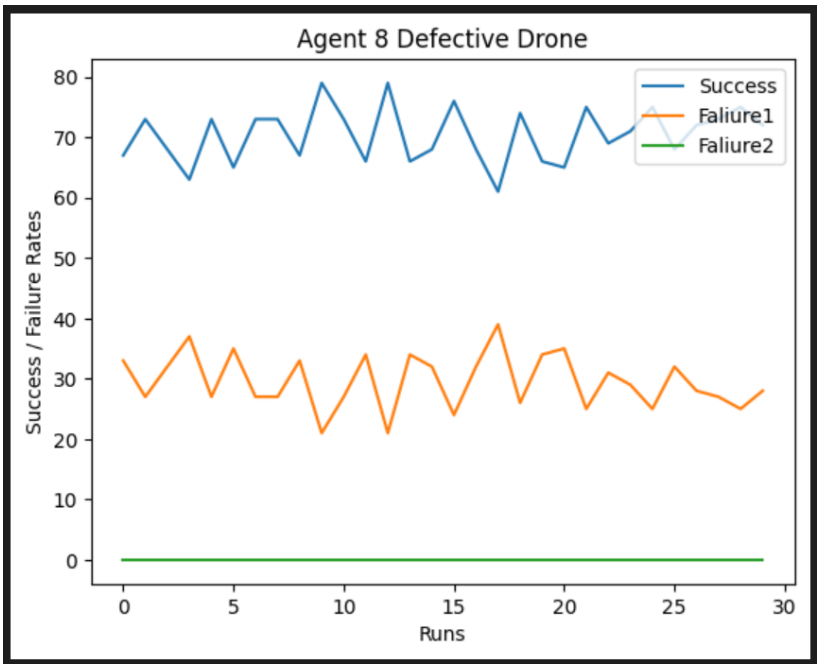
Failure 2: 0.26%

Confidence Rate:

Std. Deviation Success: 4.03

Std. Deviation Failure 1: 4.07

Std. Deviation Failure 2: 0.52



Average Observations(5000 steps)

Success: 70.4%

Failure 1: 29.5%

Failure 2: 0.0%

Confidence Rate Prey,Pred:3.3%, 18.5%

Std. Deviation Success: 4.58

Std. Deviation Failure 1: 4.58

Std. Deviation Failure 2:0

Q) Can you build an Agent 9 to do better?

After thinking about all possible cases where our strategies so far may be failing is the case where the predator is in the middle of the prey and the agent for a long time.

(Agent - Pred - Prey)

In this case the agent wants to move towards the prey but that is the same direction the predator is in. Agent is calculating the distance without considering the pred in the way. We believe that resolving this conflict may help in increasing the success rate of the agent.

It goes as follows:

When calculating the shortest path of agent's neighbors to the prey, take in account, the position of predator and

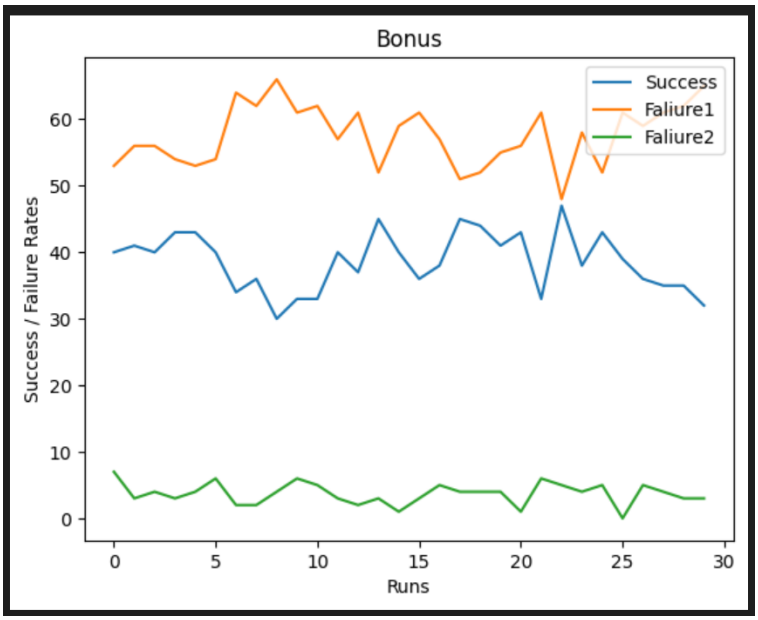
Q) Bonus:

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

Since we can only either survey or move at every timestep, we have to think of a condition based on which the bonus agent can take this decision. It seems logical to make a move only when we have enough information about either the prey or the predator. Until then the agent can survey the map, gather all the information and update probabilities accordingly. Once the agent is at least 50% sure about the position of either prey or predator, it moves.

P(prey/pred at node N | prey/pred not/at surveyed node) >= 0.5

We implemented the bonus agent and the survivability results are as follows:



Average Observations(150 steps)

Success: 38.6%

Failure 1: 57.6%

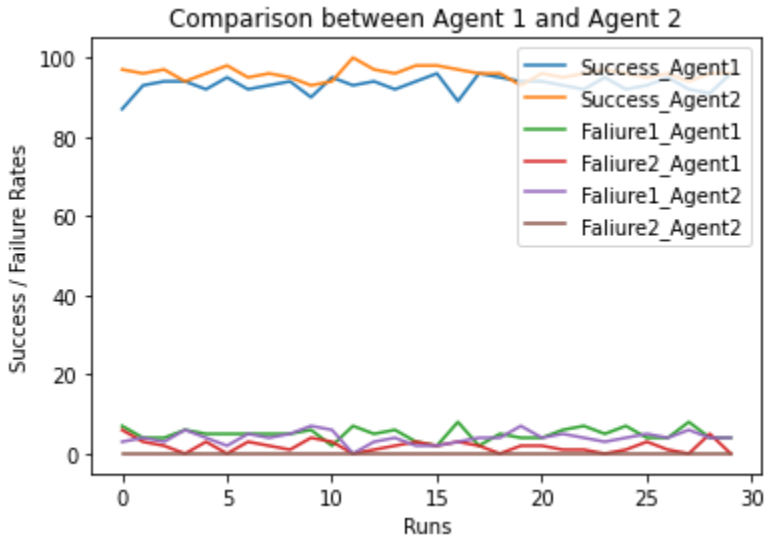
Failure 2: 3.69%

Std. Deviation Success: 4.4

Std. Deviation Failure 1: 4.59

Std. Deviation Failure 2: 1.62

Comparison graphs

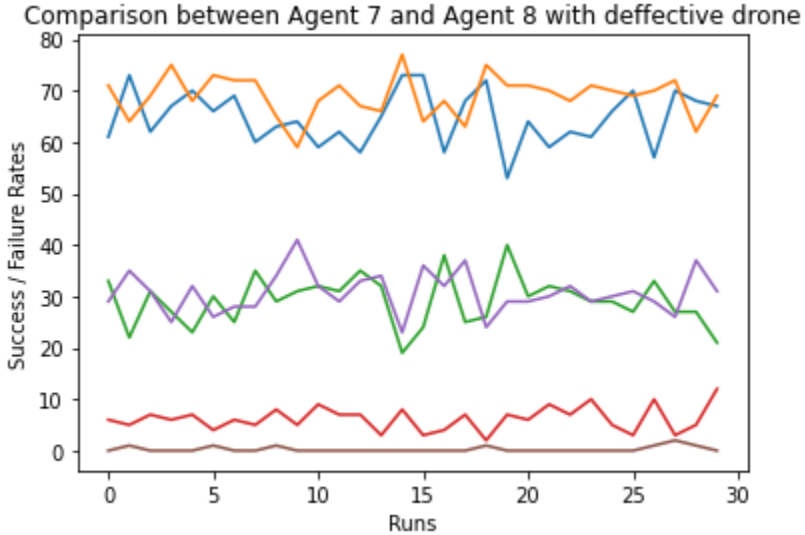


Here we can see that agent 2 outperforms agent 1 in almost all the trials, the orange line is visibly above the blue line in the comparison graph.

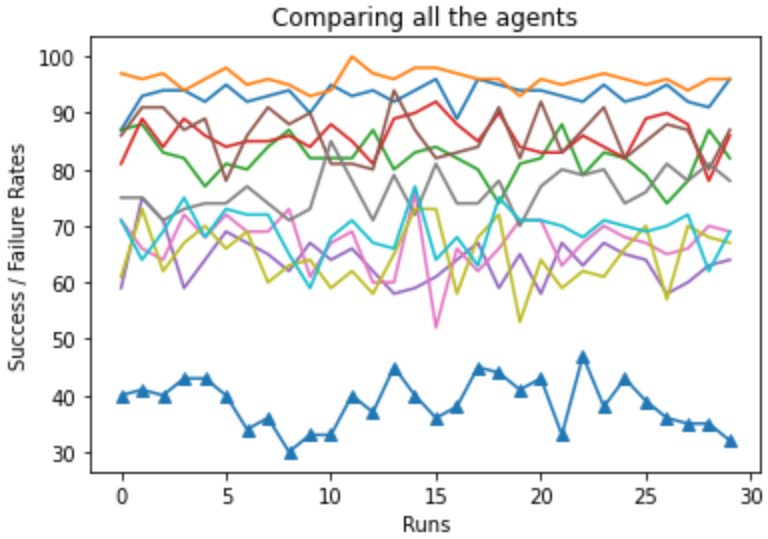
Here we can see that agent 4 outperforms agent 3 in almost all the trials, the orange line is visibly above the blue line in the comparison graph in most cases.

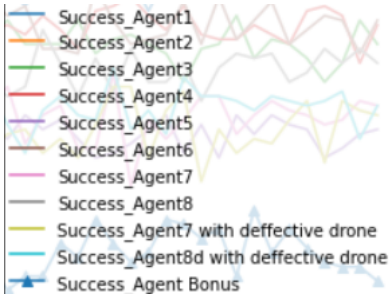
Here we can see that agent 6 outperforms agent 5 in almost all the trials, the orange line is visibly above the blue line in the comparison graph in most cases.

Here we can see that agent 8 outperforms agent 7 in almost all the trials, the orange line is visibly above the blue line in the comparison graph in most cases.



Here we can see that agent 8 outperforms agent 7 in almost all the trials, the orange line is visibly above the blue line in the comparison graph in most cases.



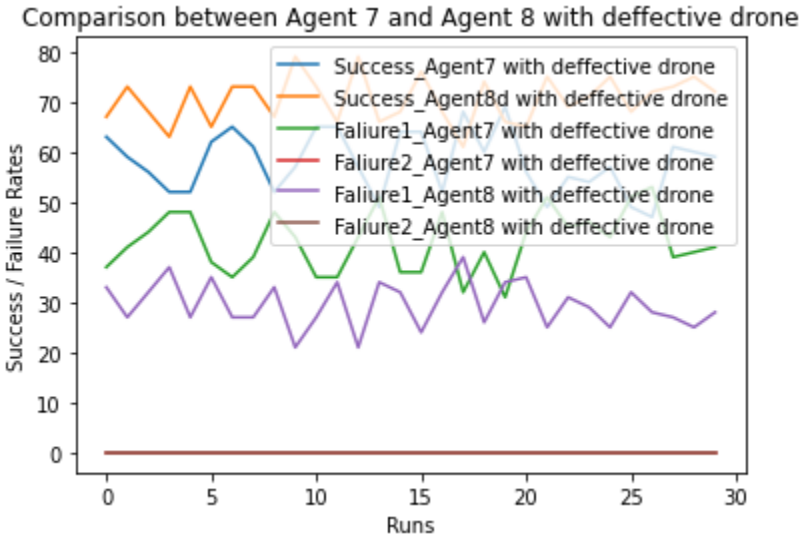


We get a rough idea about the success rates of all the agents over a span of 30 trials.

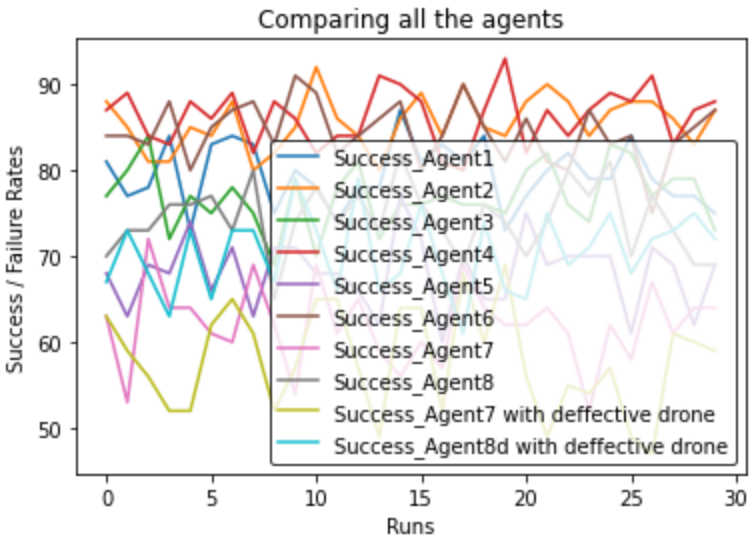
Comparison graphs For 5000 Steps


Comparison between Agent 3 and Agent 4
Success_Agent3
Success_Agent4
Faliure1_Agent3
Faliure2_Agent3
Faliure1_Agent4
Faliure2_Agent4
Success / Failure Rates
Runs
0
20
40
60
80
0
5
10
15
20
25
30


Comparison between Agent 5 and Agent 6
Success_Agent5
Success_Agent6
Faliure1_Agent5
Faliure2_Agent5
Faliure1_Agent6
Faliure2_Agent6
Success / Failure Rates
Runs
0
20
40
60
80
0
5
10
15
20
25
30



***We have attached a separate file containing all the results, calculations and graph for different step numbers.**


Comparing all the agents
Success / Failure Rates
Runs
90
80
70
60
50
0
5
10
15
20
25
30
Success_Agent1
Success_Agent2
Success_Agent3
Success_Agent4
Success_Agent5
Success_Agent6
Success_Agent7
Success_Agent8
Success_Agent7 with deffective drone
Success_Agent8d with deffective drone