Date: 14th November, 2022

CS 520 : Introduction to Artificial Intelligence

Project 2: The Circle Of Life

TEAM MEMBERS:

Naga Vamsi Krishna Bulusu (nb847)

Sakshi Ghadigaonkar (ssg156)

Project Summary:

One of the most impactful applications of algorithms in Artificial Intelligence consists of taking the right action, directed at the right target at the right time.This leads us to algorithms that make use of probability and decision theory to yield us significant results. Decision theory is a branch of applied probability theory concerned with the theory of making decisions based on assigning probabilities to various factors and assigning numerical consequences to the outcome.

This project focuses on the implementation and importance of probability and inference. Given a circular undirected graph of 50 nodes with random edges with predator, prey and agent, we find the probability of our agent catching the prey or the predator catching the agent under different circumstances and collect this data to conclude our results.

Programming language: Python

Environment: The environment is a circular undirected graph of 50 nodes. We add random edges from and to the edges that have degree less than 3 within 5 steps forward or backward along the primary loop. While doing this we make sure that there are no self loops or duplicate edges in the environment graph. Whenever we call the environment function within our main function it generates this graph following the given conditions. We also spawn the agent, prey and the predator at random nodes such that the agent and predator or the agent and the prey do not spawn at the same nodes.

We are also implementing Floyd Warshall Algorithm which uses a distance matrix to compute the shortest distance between all pairs of nodes.

Note: For a 50 node graph, we can add a maximum of 25 random edges. To find the smallest number of edges that we are always able to add, we performed random sampling on 20,000 graphs. We found that at least 20 random edges were formed.

Time Complexity:
The distance matrix that we generated for Floyd Warshall is used everytime we want to find the shortest distance between any two nodes. This takes $O(n^3)$ time where n is the number of nodes.

Prey: The prey is spawned at a random node which is not the same as the agent. Everytime the prey wants to move or, it will select one of its neighbours with uniform probability along with its current position. For this we have created a move_prey function which selects one node randomly from a list consisting of its current position and its neighbours and assigns it to the next prey position.

Note: After every prey movement, we check if the prey moves into our agent's node and return AGENT WON if it is true.

Predator: Like the prey and the agent , the predator is also spawned at a random node. Everytime the predator wants to move, it computes the shortest distance from all of its neighbours to the agent position and selects the neighbour with the shortest distance that takes it closer to the agent. Our move_predator function creates a list of tuples with the predator neighbours along with their distances to the agent position. The list is then sorted in ascending order of the distances and we select the neighbour with the minimum distance to the agent. If there are multiple neighbours with the minimum distance then we pick randomly from one of them.

Note: After the predator moves, we check if the Agent is alive or not and if the agent is dead then we terminate that simulation.

The Complete Information Setting (Agent 1 and Agent 2)

Agent 1:

In this scenario, at every timestep the agent is aware about the prey and predator's location. The movement of Agent 1 is guided by the following rules. The next potential neighbour(agent's next position) is generated by applying rule based filtering on the neighbours of the agent. Rules are prioritised from top to bottom.

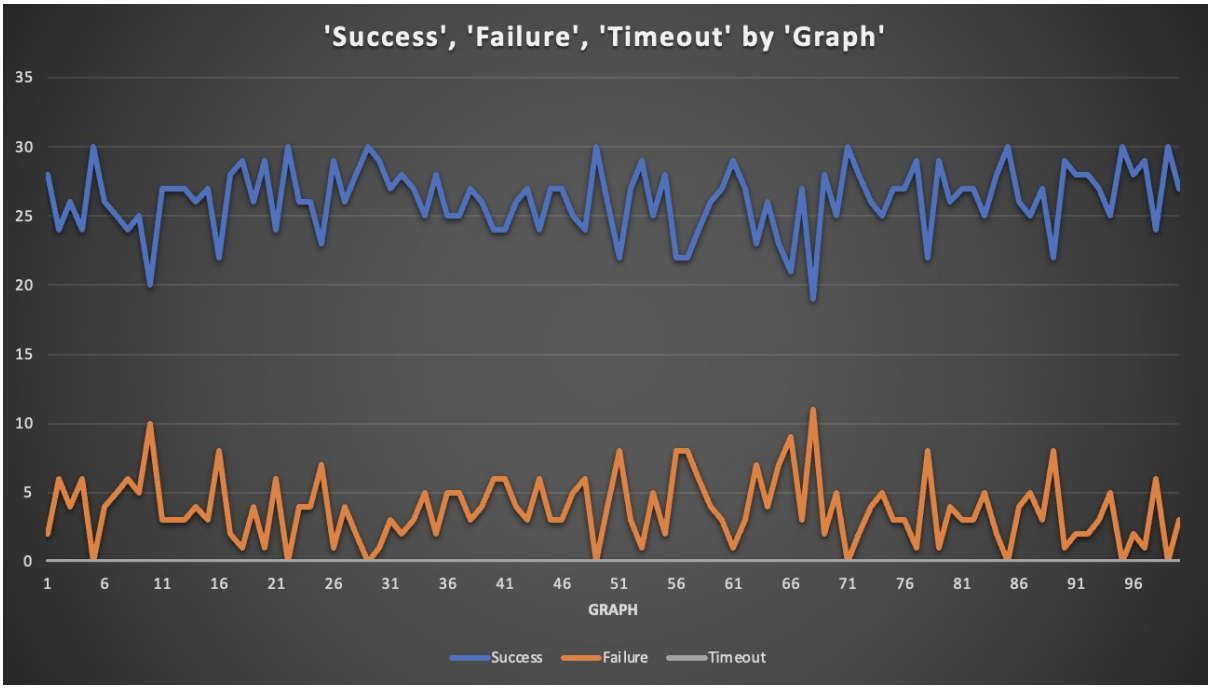
Let's denote:

- 'agent_prey' as the current distance between Agent & Prey
- 'agent_predator' the current distance between Agent & Predator.
- 'neighbour_prey' as the current distance between neighbour of Agent & Prey
- 'neighbour_pred' as the current distance between neighbour of agent & Predator

Rules	Comparison
Neighbours that are closer to the Prey and farther from the Predator	neighbour_prey < agent_prey And neighbour_predator > agent_predator
Neighbours that are closer to the Prey and not closer to the Predator	neighbour_prey < agent_prey And neighbour_predator == agent_predator
Neighbours that are not farther from the Prey and farther from the Predator	neighbour_prey == agent_prey And neighbour_predator > agent_predator
Neighbours that are not farther from the Prey and not closer to the Predator	neighbour_prey == agent_prey And neighbour_predator == agent_predator
Neighbours that are farther from the Predator	neighbour_predator > agent_predator
Neighbours that are not closer to the Predator	neighbour_predator == agent_predator

Data Analysis of AGENT 1:

For populating the data, we are running 30 simulations on 100 graphs each keeping 5000 timesteps as timeout.



1. Avg(Agent Success Rate): **87.46666666666667 %**
2. Avg(Agent Failure rate): **12.533333333333333 %**
3. Avg(Agent Timeout rate): **0.0 %**

Agent 2:

- We noticed that when our Agent 1 was following the above order of rules, it was choosing the neighbour with the same distance from the predator over the neighbour that takes it away from the predator.
- The predator never stays at the same location and moves closer to the agent. If the agent remains at the same distance from the predator subsequently, the predator gets closer to the agent ultimately catching our agent.
- So we changed the order of rules to follow in such a way that the agent will prefer moving away from the predator even if the distance from the prey is the same over remaining at the same distance from the predator.

Let's denote:

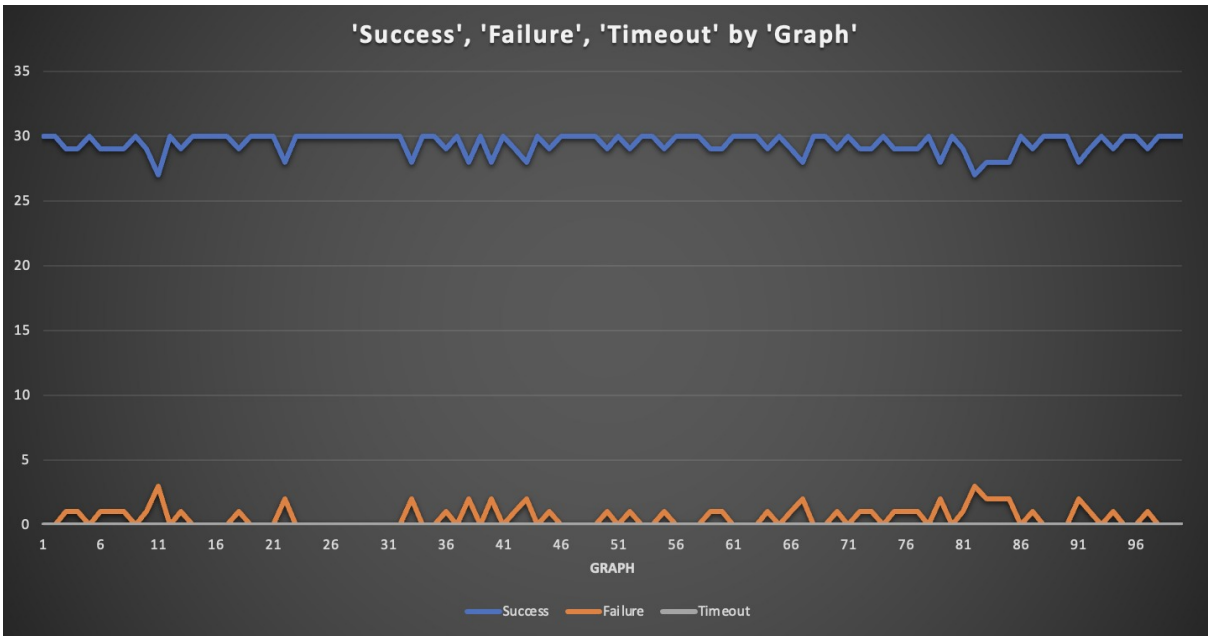
- 'agent_prey' as the current distance between Agent & Prey
- 'agent_predator' the current distance between Agent & Predator.
- 'neighbour_prey' as the current distance between neighbour of Agent & Prey
- 'neighbour_pred' as the current distance between neighbour of agent & Predator

Rules	Comparison
Neighbours that are closer to the Prey and farther from the Predator	neighbour_prey < agent_prey And neighbour_predator > agent_predator

Neighbours that are closer to the Prey and not closer to the Predator	neighbour_prey < agent_prey And neighbour_predator == agent_predator
Neighbours that are not farther from the Prey and farther from the Predator	neighbour_prey == agent_prey And neighbour_predator > agent_predator
Neighbours that are farther from the Predator	**neighbour_predator > agent_predator**
Neighbours that are not farther from the Prey and not closer to the Predator	neighbour_prey == agent_prey And neighbour_predator == agent_predator
Neighbours that are not closer to the Predator	neighbour_predator == agent_predator

Data Analysis of AGENT 2:

For populating the data, we are running 30 simulations on 100 graphs each keeping 5000 timesteps as timeout.



1. Avg(Agent Success Rate): **98.1 %**
2. Avg(Agent Failure rate): **1.9 %**
3. Avg(Agent Timeout rate): **0.0 %**

The Partial Prey Information Setting (Agent 3 and Agent 4)

Agent 3:

- In this scenario, the agent is aware of the predator's location at every timestep but may or may not know the prey's location.
- We maintain a belief array which contains the probability of prey being at each node in the graph. The sum of all the probabilities should sum up to 1. We initialise this belief array with equal probability of 1/49 for all the nodes except the agent position whose probability would be 0.
- So whenever the agent wants to move, it first decides to survey any node in the entire graph to check if the prey is there in that node or not. We choose this survey node as the node with the highest probability value in the belief array.If there are multiple nodes with the highest probability then we pick at random.
- If we find the prey at the survey node, we update the belief array with the probability of the survey node to 1 and all the other nodes to 0. If the survey function returns False, we update the probability of the survey node to 0 and the other nodes on the basis of the following equation:

  ➔ Prob(prey at node(i)) = Prob(prey at node(i) | prey not at Surveyed Node)

  ↓

  Prob(prey at node(i) | prey not at Surveyed Node) = Prob(prey at node(i) and prey not at Surveyed Node) / Prob(prey not at surveyed node)

  ↓

  Prob(prey at node(i)) = Prob(prey at node(i)) * Prob(prey not at Surveyed Node | prey at node(i)) / 1 - Prob(prey at Surveyed Node)

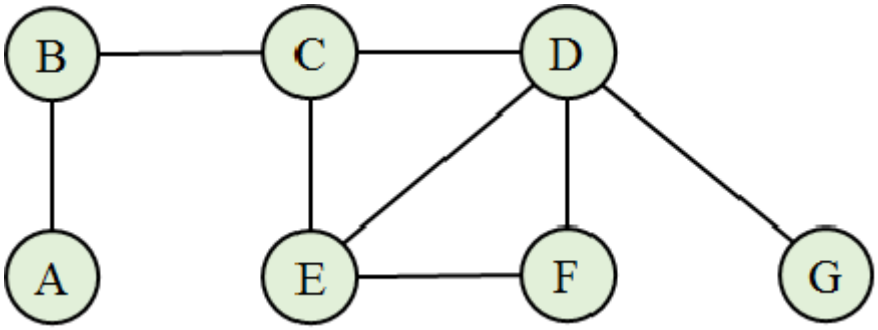
- After updating the belief array, we again look for the highest probability node and the agent moves in the direction of this node.
- After the agent moves we check if our agent caught the prey or not. If yes then our Agent won and we terminate the simulation or else we again update the belief array after the agent's movement in accordance with the same equation as we derived above:

  ➔ **Prob(prey at node(i)) = Prob(prey at node(i)) * Prob(prey not at Surveyed Node | prey at node(i)) / 1 - Prob(prey at Surveyed Node)**

  Note: After picking the highest probability node, the agent will move in accordance with the same order of rules as Agent 1.

- After this, we first move the prey and check if the prey walked into the agent's location or not. We terminate the simulation and return AGENT WON if it is true. If not then we move our predator and check if our agent is alive or not. If our agent is alive we once again update the belief array on the basis of another equation as follows:

  Let us consider the following example:



  Suppose we want to compute the probability of the prey being at node B in the next timestep whose neighbours are A and C. We can compute that by adding the probability of the prey being at node (A, C, B)in the current timestep which is given by:

➔ **Prob (prey at node B_{t+1}) = Prob(prey at B_{t+1}, prey at node A_t) + Prob(prey at B_{t+1}, prey at node C_t) + Prob(prey at B_{t+1}, prey at node B_t)**

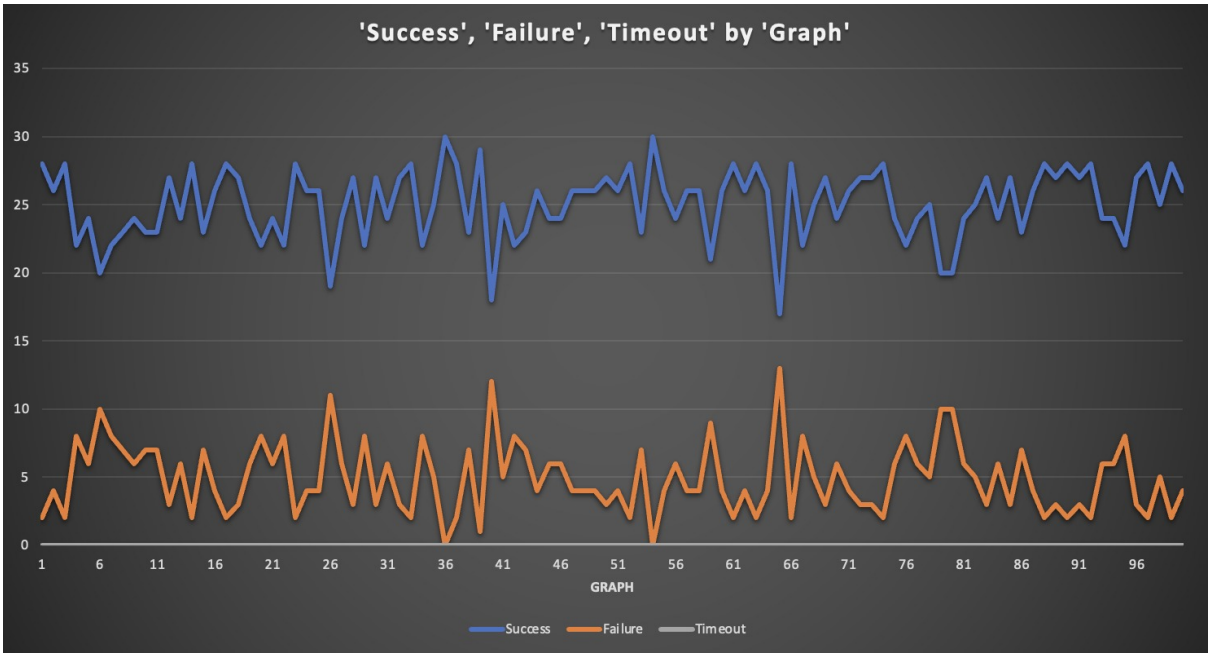
  where,
  Prob(prey at B_{t+1}, prey at node A_t) = 1/ (number of neighbours of A + 1)
  Prob(prey at B_{t+1}, prey at node C_t) = 1/ (number of neighbours of C + 1)
  Prob(prey at B_{t+1}, prey at node B_t) = 1/ (number of neighbours of B + 1)

Note: Throughout the simulation, we update our belief array 3 times in total:

1. After surveying a node
2. After the agent's movement
3. After the prey's movement

Data Analysis of AGENT 3:

For populating the data, we are running 30 simulations on 100 graphs each keeping 5000 timesteps as timeout.



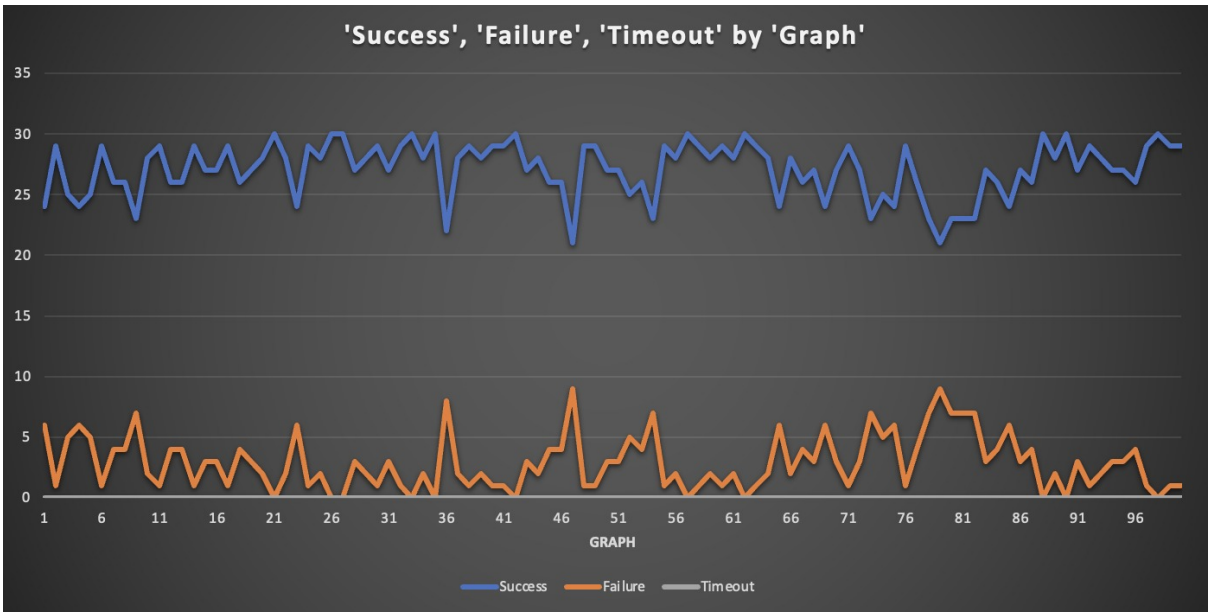
1. Avg(Agent Success Rate): **83.6 %**
2. Avg(Agent Failure rate): **16.4 %**
3. Avg(Agent Timeout rate): **0.0 %**
4. Avg(Agents Correct Survey): **6.111571345952452 %**

Agent 4:

We implemented Agent 4 in accordance with Agent 2 and Agent 3.

Data Analysis of AGENT 4:

For populating the data, we are running 30 simulations on 100 graphs each keeping 5000 timesteps as timeout.



1. Avg(Agent Success Rate): **93.6 %**
2. Avg(Agent Failure rate): **6.4 %**
3. Avg(Agent Timeout rate): **0.0 %**
4. Avg(Agents Correct Survey): **6.490030777465543 %**

The Partial Predator Information Setting (Agent 5 and Agent 6)

Agent 5:

- In this scenario, the agent is aware of the prey's location throughout the simulation but other than the start position, the agent may or may not know the predator's location.
- The predator in this setting is termed as an ' Easily Distracted Predator' meaning that it will choose the neighbour which takes it closer to the agent 60% of the time and for the remaining 40% it picks any neighbour at random.
- We maintain a belief array as we did in the partial prey information setting, which contains the probability of the predator being at each node in the graph.
- We initialise this belief array to probability 0 for all the nodes except the starting location of the predator whose probability would be 1.
- We pick the node with the highest probability and survey that node for the predator. If there are multiple nodes with the highest probability then we pick at random.
- If we find the predator at the survey node, we update the probability of the survey node to 1 and all the other nodes to 0. If the survey function returns False we update our belief array in accordance with the same equation we derived for the partial prey information setting:

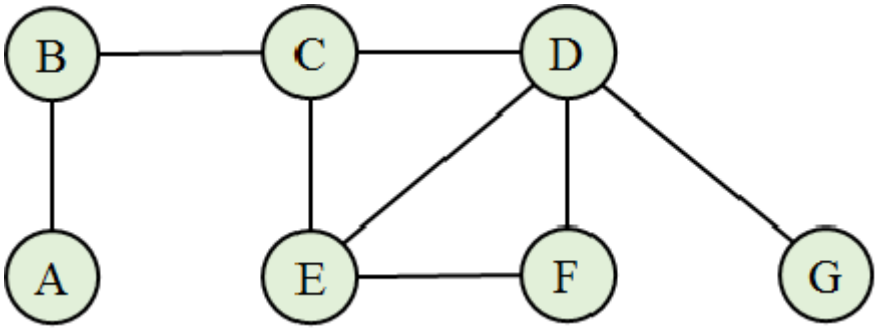
    ➔ **Prob(predator at node(i)) = Prob(predator at node(i)) * Prob(predator not at Surveyed Node | predator at node(i)) / 1 - Prob(predator at Surveyed Node)**

- After updating the belief array, we again look for the highest probability node and move our agent in the direction of this node. After the agent moves we check if our agent caught the prey or not. If yes then our AGENT WON and we terminate the simulation or else we again update the belief array after the agent's movement in accordance with the same equation given above:

    ➔ **Prob(predator at node(i)) = Prob(predator at node(i)) * Prob(predator not at Surveyed Node | predator at node(i)) / 1 - Prob(predator at Surveyed Node)**

- After this, we move the prey and check if the agent is alive or not. If our agent is alive then we update our belief array on the basis of the following equation:

Let us consider the following example:



Suppose we want to compute the probability of the predator being at node B in the next timestep whose neighbours are A and C. We can compute that in the following manner:

➔ **Prob (predator at node B_{t+1}) = Prob(predator at B_{t+1}, predator at node A_t) + Prob(predator at B_{t+1}, predator at node C_t) + Prob(predator at B_{t+1}, predator at node B_t)**

Now, Consider Agent is at Node A.

↓

Prob(predator at B_{t+1}, predator at node C_t) = Prob(predator at C_t) * Prob(predator at B_{t+1} | predator at C_t)

↓

We get the value of **Prob(predator at C_t) from our belief array**

↓

C has neighbours B, D and E.

↓

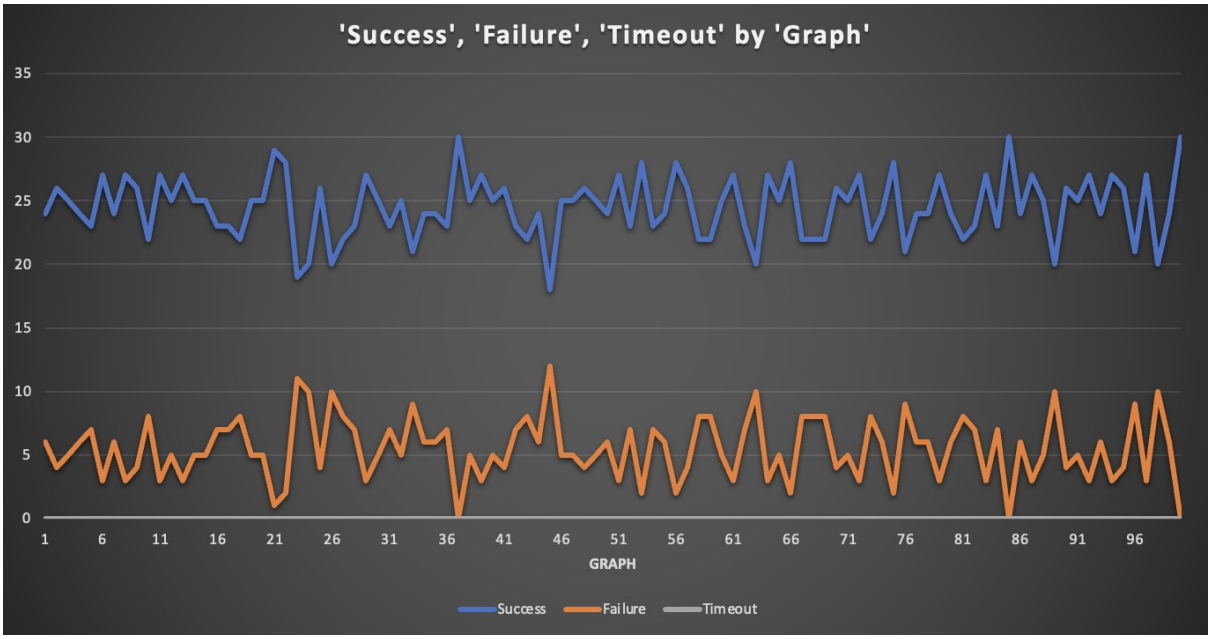
Since our predator in this setting is easily distracted so, **Prob(predator at B_{t+1} | predator at C_t) ={ (0.4 * 1/(number of neighbours of C))+ (0.6 * number of shortest paths to A via B from C/(total number of shortest paths to A from C))}**

Note: Throughout the simulation, we update our belief array 3 times in total:

1. After surveying a node
2. After the agent's movement
3. After the predator's movement

Data Analysis of AGENT 5:

For populating the data, we are running 30 simulations on 100 graphs each keeping 5000 timesteps as timeout.



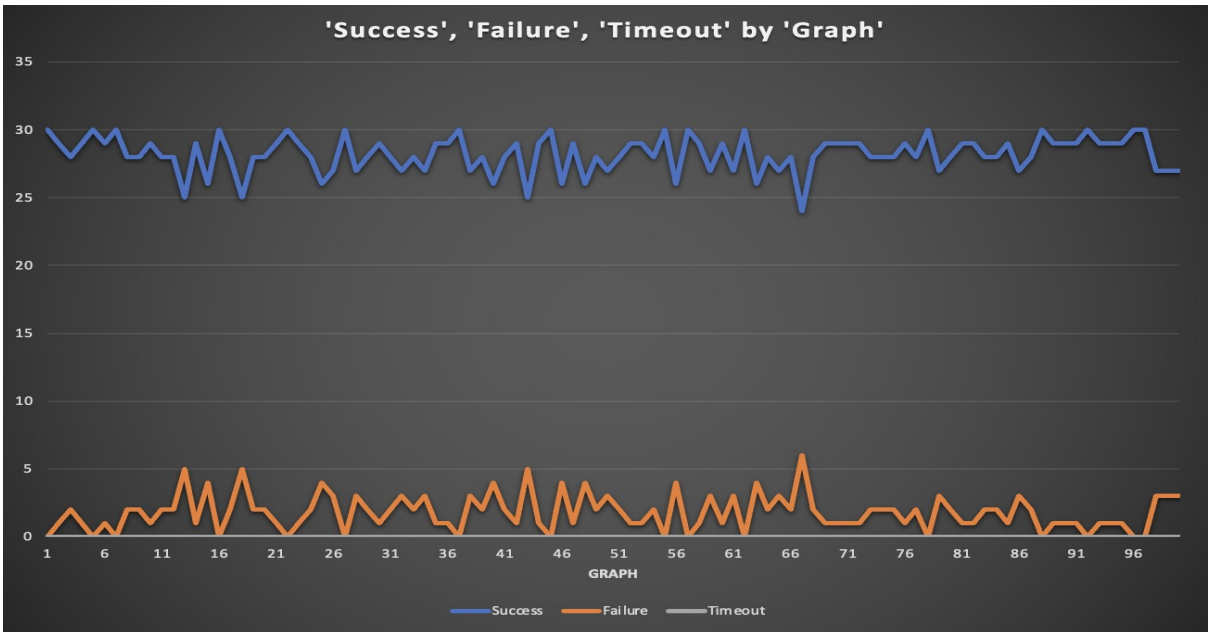
1. Avg(Agent Success Rate): **81.76666666666667 %**
2. Avg(Agent Failure rate): **18.233333333333334 %**
3. Avg(Agent Timeout rate): **0.0 %**
4. Avg(Agents Correct Survey): **58.467724777669034 %**

Agent 6:

We implemented Agent 6 in accordance with Agent 2 and Agent 5 .

Data Analysis of AGENT 6:

For populating the data, we are running 30 simulations on 100 graphs each keeping 5000 timesteps as timeout.



1. Avg(Agent Success Rate): **94.06666666666666 %**
2. Avg(Agent Failure rate): **5.933333333333334 %**
3. Avg(Agent Timeout rate): **0.0 %**
4. Avg(Agents Correct Survey): **50.83181936153647 %**

The Combined Partial Information Setting (Agent 7 and Agent 8)

Agent 7:

- In this scenario, the agent may or may not know the prey or predator location except for the starting location of the predator.
- Like the previous setting, our predator is 'Easily distracted' in this setting as well. So this time, we maintain two belief arrays, one each for the prey and predator.
- In accordance with the previous two settings, we initialise the belief_prey array with equal probability of 1/49 for all the nodes except the agent position whose probability would be 0 and belief_predator array to probability 0 for all the nodes except the starting location of the predator whose probability would be 1.
- We first pick two nodes, one with the highest probability value from the belief_prey array(prey_node) and other with the highest probability value from the belief_predator array(pred_node) and compare them.
- If the value of pred_node > prey_node, we survey the pred_node for the predator. If we find the predator at the survey node, we update the probability of the survey node to 1 and all the other nodes to 0 in the belief_predator array. If the survey function returns False we update our belief array in accordance with the same equation we derived in the partial predator information setting:

    ➔ **Prob(predator at node(i)) = Prob(predator at node(i)) * Prob(predator not at Surveyed Node | predator at node(i)) / 1 - Prob(predator at Surveyed Node)**

    Note: When we survey for the predator, we are in turn getting information about the prey as well. So we update our belief_prey array in a similar fashion depending on whether we find the prey in that node or not.

- Else, we survey the node with the highest probability value in the belief_prey array for the prey.
- If we find the prey at the survey node, we update the probability of the survey node to 1 and all the other nodes to 0 in the belief_prey array. If the survey function returns False we update our belief array in accordance with the same equation we derived in the partial prey information setting:

➔ **Prob(prey at node(i)) = Prob(prey at node(i)) * Prob(prey not at Surveyed Node | prey at node(i)) / 1 - Prob(prey at Surveyed Node)**

Note: When we survey for the prey, we are in turn getting information about the predator as well. So we update our belief_predator array in a similar fashion depending on whether we find the predator in that node or not.

- After this, we move our agent considering both highest probability nodes of prey and predator and check if our agent caught the prey or not. If yes then our AGENT WON and we terminate the simulation.
- If not then we once again update both the belief arrays in accordance with the same equation given above:
    a) belief_predator array:
        ➔ **Prob(predator at node(i)) = Prob(predator at node(i)) * Prob(predator not at Surveyed Node | predator at node(i)) / 1 - Prob(predator at Surveyed Node)**
    b) belief_predator array:
        ➔ **Prob(prey at node(i)) = Prob(prey at node(i)) * Prob(prey not at Surveyed Node | prey at node(i)) / 1 - Prob(prey at Surveyed Node)**
- After that, we move the prey and check if the prey walked into the agent's location or not. If yes, then our AGENT WON and we terminate the simulation.
- Else, we move the predator and check if our agent is alive or not. If the agent is alive then, we update our belief_prey array and our belief_predator array in accordance with the following equations we derived for the partial predator information setting and partial prey information setting respectively:
    a) belief_prey array:
        ➔ **Prob (prey at node B_{t+1}) = Prob(prey at B_{t+1}, prey at node A_t) + Prob(prey at B_{t+1}, prey at node C_t) + Prob(prey at B_{t+1}, prey at node B_t)**

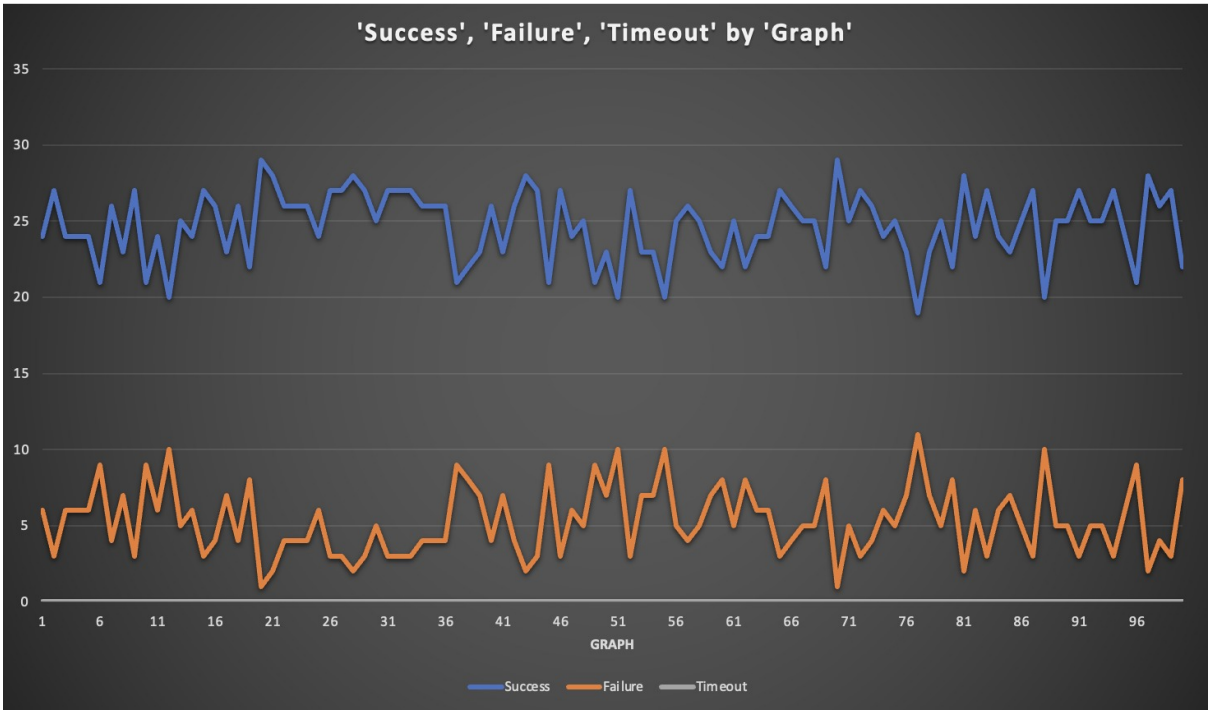
    b) belief_predator array:
        ➔ **Prob (predator at node B_{t+1}) = Prob(predator at B_{t+1}, predator at node A_t) + Prob(predator at B_{t+1}, predator at node C_t) + Prob(predator at B_{t+1}, predator at node B_t)**

Note: Throughout the simulation, we update our belief array 3 times in total:

1. After surveying a node(according to the condition either belief_prey or belief_predator)
2. After the agent's movement
3. After the prey and predator's movement

Data Analysis of AGENT 7:

For populating the data, we are running 30 simulations on 100 graphs each keeping 5000 timesteps as timeout.



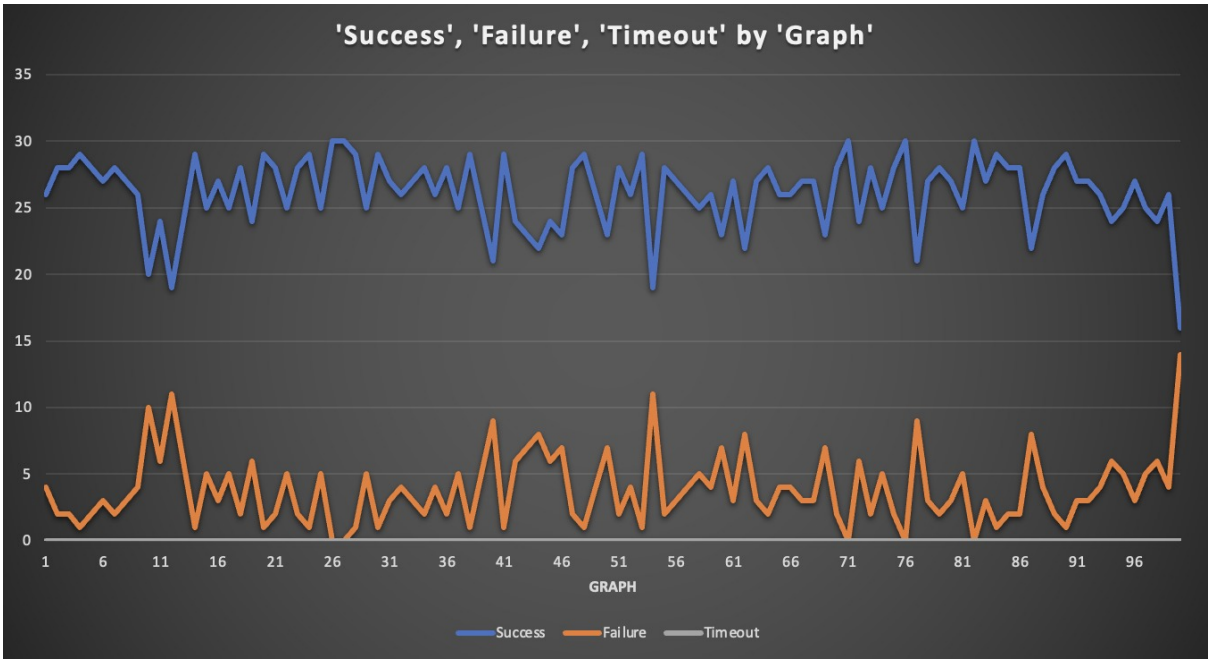
1. Avg(Agent Success Rate): **82.3 %**
2. Avg(Agent Failure rate): **17.7 %**
3. Avg(Agent Timeout rate): **0.0 %**
4. Avg(Agents Correct Prey Survey): **0.4411865295606375 %**
5. Avg(Agents Correct Predator Survey): **51.5595430812376 %**

Agent 8:

We implemented Agent 8 in accordance with Agent 2 and Agent 7 .

Data Analysis of AGENT 8:

For populating the data, we are running 30 simulations on 100 graphs each keeping 5000 timesteps as timeout.



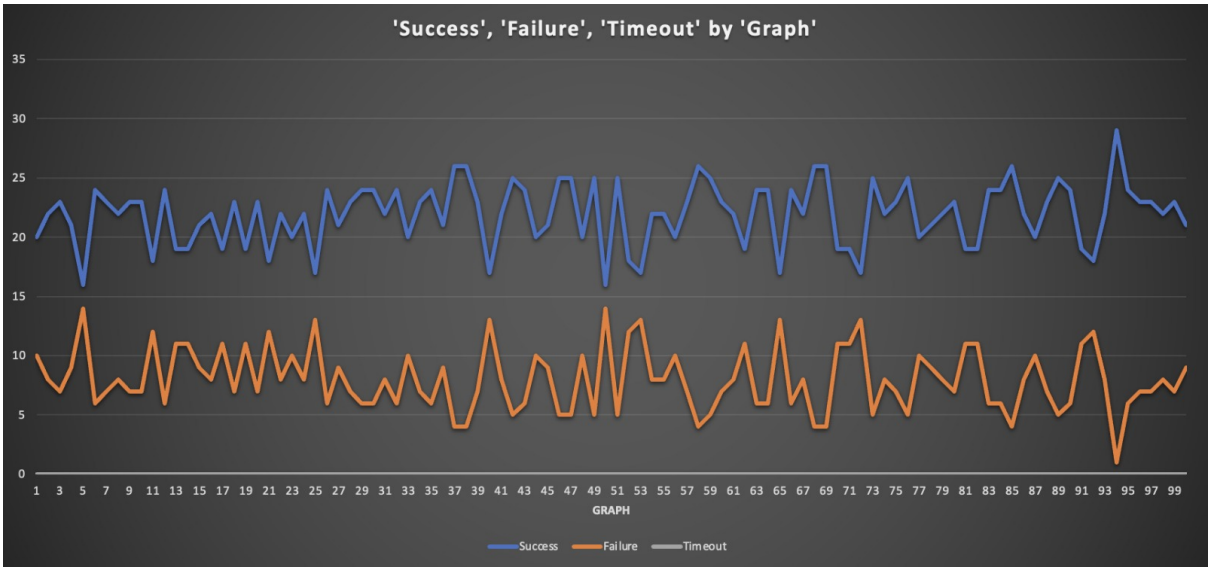
1. Avg(Agent Success Rate): **87.23333333333333 %**
2. Avg(Agent Failure rate): **12.766666666666667 %**
3. Avg(Agent Timeout rate): **0.0 %**
4. Avg(Agents Correct Prey Survey): **0.0 %**
5. Avg(Agents Correct Predator Survey): **56.078130847783854 %**

The Combined Partial Information Setting in noisy-survey environment:

For this scenario, we take into consideration that in our Combined Partial Information Setting, our survey function(the drone) is faulty. When the result is False, it will return False 90% of the times but will return True the remaining 10% of the time. Let us see what effect this faulty survey function has on our Agent 7 and Agent 8.

Data Analysis of AGENT 7 with the faulty drone:

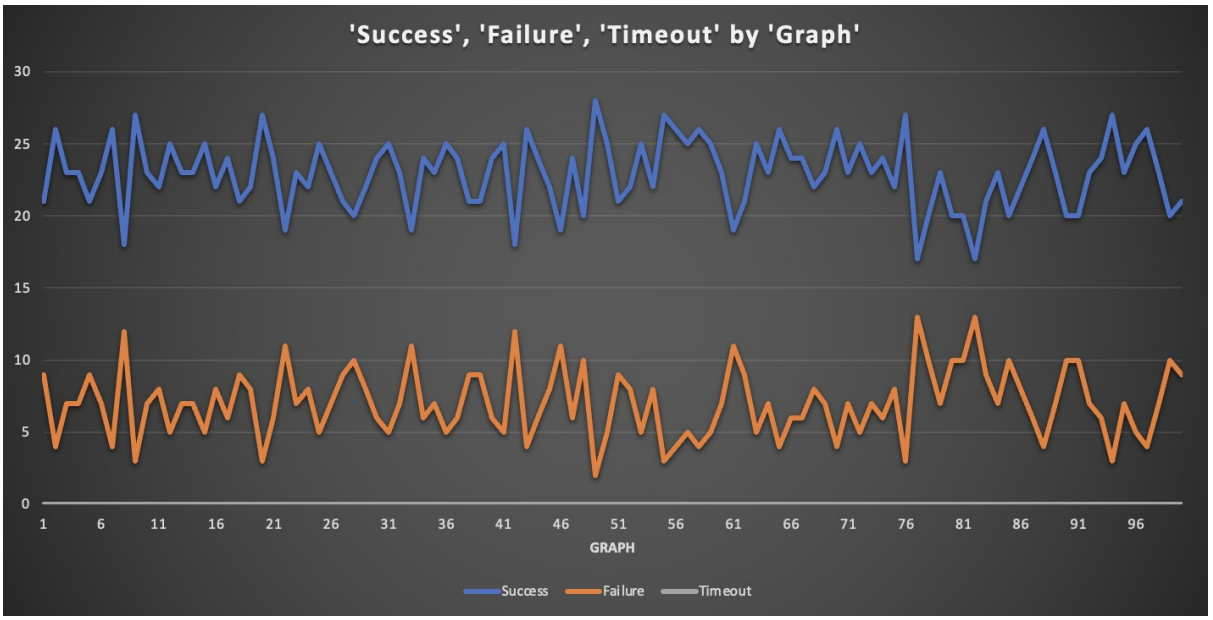
For populating the data, we are running 30 simulations on 100 graphs each keeping 5000 timesteps as timeout.



1. Avg(Agent Success Rate): **73.3 %**
2. Avg(Agent Failure rate): **26.7 %**
3. Avg(Agent Timeout rate): **0.0 %**
4. Avg(Agents Correct Prey Survey): **0.8113742182721474 %**
5. Avg(Agents Correct Predator Survey): **32.355107288637015 %**

Data Analysis of AGENT 8 with faulty drone:

For populating the data, we are running 30 simulations on 100 graphs each keeping 5000 timesteps as timeout.



1. Avg(Agent Success Rate): **76.5 %**
2. Avg(Agent Failure rate): **23.5 %**
3. Avg(Agent Timeout rate): **0.0 %**
4. Avg(Agents Correct Prey Survey): **0.14683489232107896 %**
5. Avg(Agents Correct Predator Survey): **30.506852294974983 %**

The Combined Partial Information Setting after updating the belief update rules to account for the noisy survey (Agent 7.1 and Agent 8.1):

To account for the faulty survey function, we need to change our rules for updating the belief array. We need to find the probability of prey/predator being at the survey node when the survey function returns FALSE which is given by the following equation:

- ➔ Let X denote prey/predator. We have to find Prob(X is at surveyed node | survey function returns False)
- ➔ Using conditional probability,
  Prob(X is at surveyed node | survey function returns False) = Prob(X is at surveyed node and survey function returns False) / Prob(survey function returns False)
- ➔ We get the value of Prob(X is at surveyed node) from the belief array.
- ➔ Prob(X is at surveyed node and survey function returns False) = Prob(X is at surveyed node) * Prob(survey function returns False | X is at surveyed node) where Prob(survey function returns False | X is at surveyed node) = 0.1
- ➔ Prob(survey function returns False) = Prob(survey function returns False and X is at surveyed node) + Prob(survey function returns False and X is not at surveyed node)
- ➔ Using conditional factoring,
  a) Prob(survey function returns False and X is at surveyed node) = Prob(X is at surveyed node) * Prob(survey function returns False | X is at surveyed node)

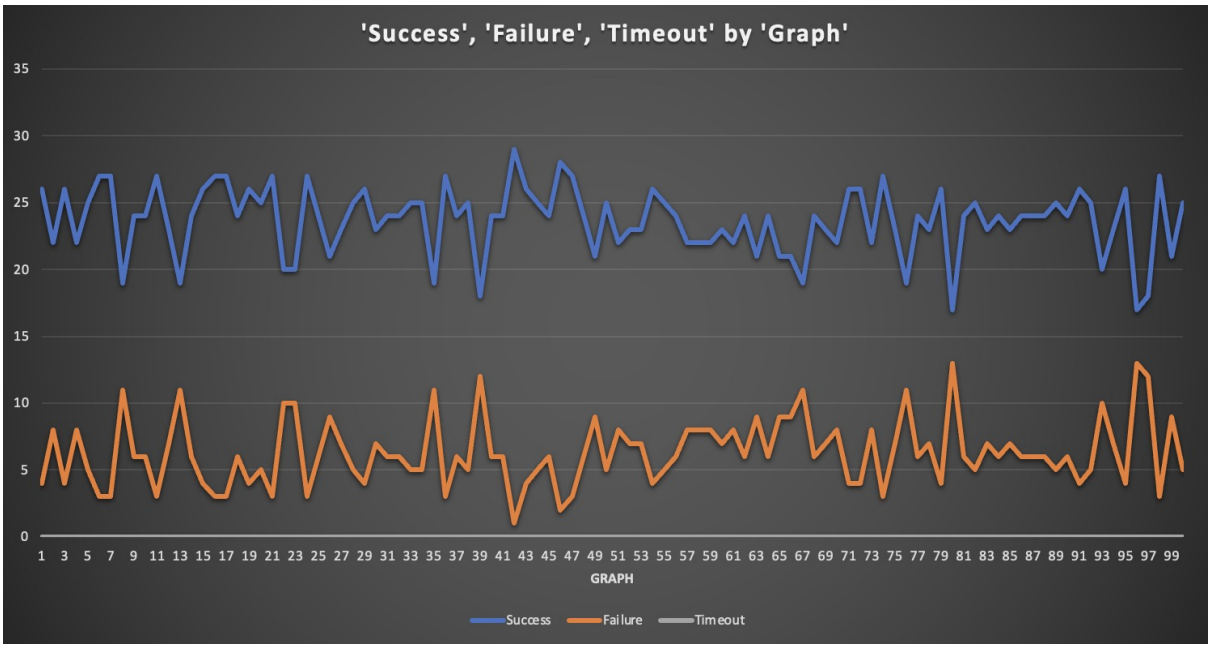
     Where we get the value of Prob(X is at surveyed node) from the belief array and Prob(survey function returns False | X is at surveyed node) = 0.1

  b) Prob(survey function returns False and X is not at surveyed node) = Prob(X is at surveyed node) * Prob(survey function returns False | X is not at surveyed node)

     Where we get the value of Prob(X is not at surveyed node) = (1 - the value in the belief array) and Prob(survey function returns False | X is not at surveyed node) = 1

Data Analysis of AGENT 7.1:

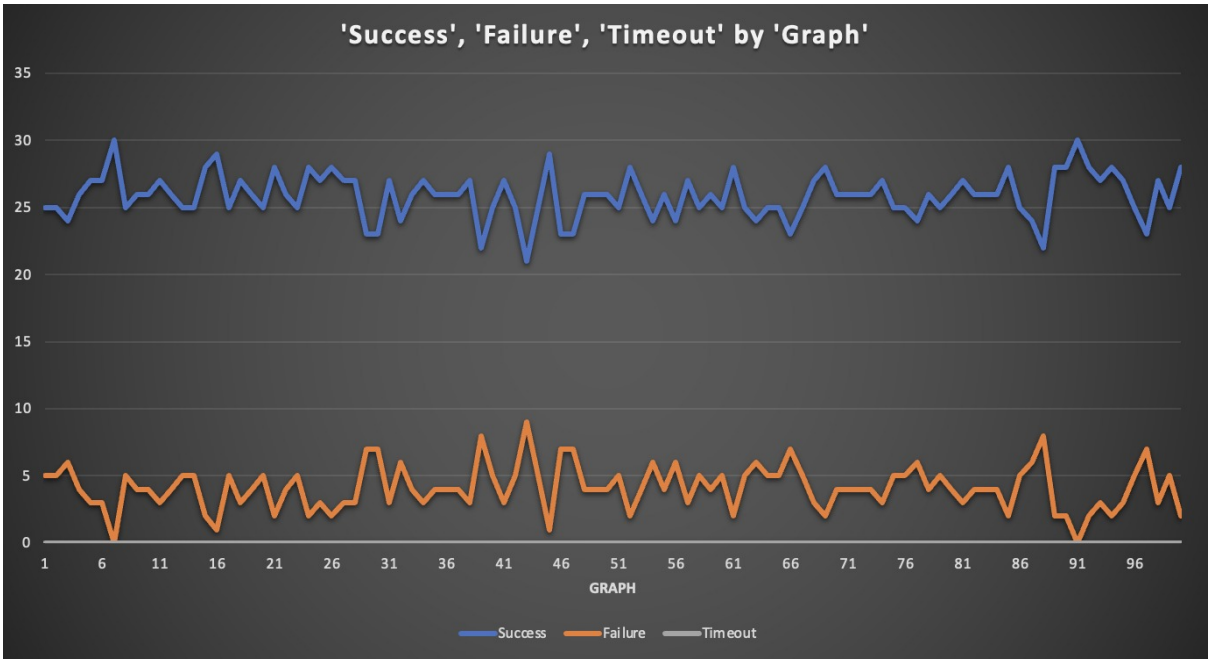
For populating the data, we are running 30 simulations on 100 graphs each keeping 5000 timesteps as timeout.



1. Avg(Agent Success Rate): **78.93333333333334 %**
2. Avg(Agent Failure rate): **21.066666666666666 %**
3. Avg(Agent Timeout rate): **0.0 %**
4. Avg(Agents Correct Prey Survey): **0.6356223229560809 %**
5. Avg(Agents Correct Predator Survey): **36.38282196331123 %**

Data Analysis of AGENT 8.1:

For populating the data, we are running 30 simulations on 100 graphs each keeping 5000 timesteps as timeout.



1. Avg(Agent Success Rate): **81.2 %**
2. Avg(Agent Failure rate): **18.8 %**
3. Avg(Agent Timeout rate): **0.0 %**
4. Avg(Agents Correct Prey Survey): **0.992681363896977 %**
5. Avg(Agents Correct Predator Survey): **30.5212532742746 %**

Strategy for Bonus Agent for the combined partial information settings :

- For this setting, our agent has to decide whether to move in the direction of the highest probability node or to survey.
- Our strategy is that whenever the max probability of a node from the belief_predator array is more than the max probability of a node from the belief_prey array, instead of directly surveying that node, we will first check if the survey node is in the agent's vicinity or not. If yes then, we first survey that node or else our agent moves in the direction of the survey node.
- We define this vicinity as the threshold value for the distance between the agent and Predator.
- Since the random edges from any node are added only within 5 steps,the probability of the predator reaching the agent in the next time step if it is beyond 4 steps in the current timestep is 0 so we can set this threshold value to 4.

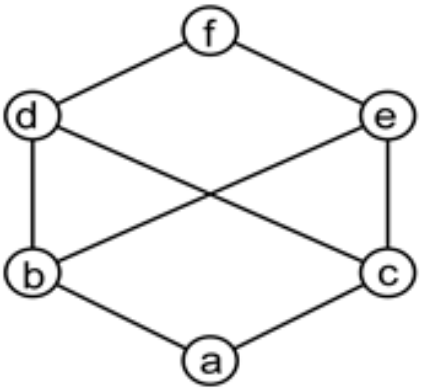
Strategy for Agent 9:

To improve our Agents in the combined partial setting with faulty survey function after changing the belief update rules, we thought of the following strategy:

- We follow the same steps as agent 7.1 and agent 8.1 before the agent's movement.

- Now, before moving our agent, we will consider the pred_node as the predator position and prey_node and the prey position.

  Let us consider the following example:



  ➔ Let the agent be at a, predator at f and prey at e.

Neighbours of a:	b,c
Neighbours of f :	d, e
Neighbours of e:	b,c,f

➔ Now, we will calculate the average of shortest distances between the agent's position and its neighbours to the predator's position and all of its neighbours. Similarly calculate the average of shortest distances between the agent's position and its neighbours to prey's position and all of its neighbours.

➔ Example:

Let the distance between:

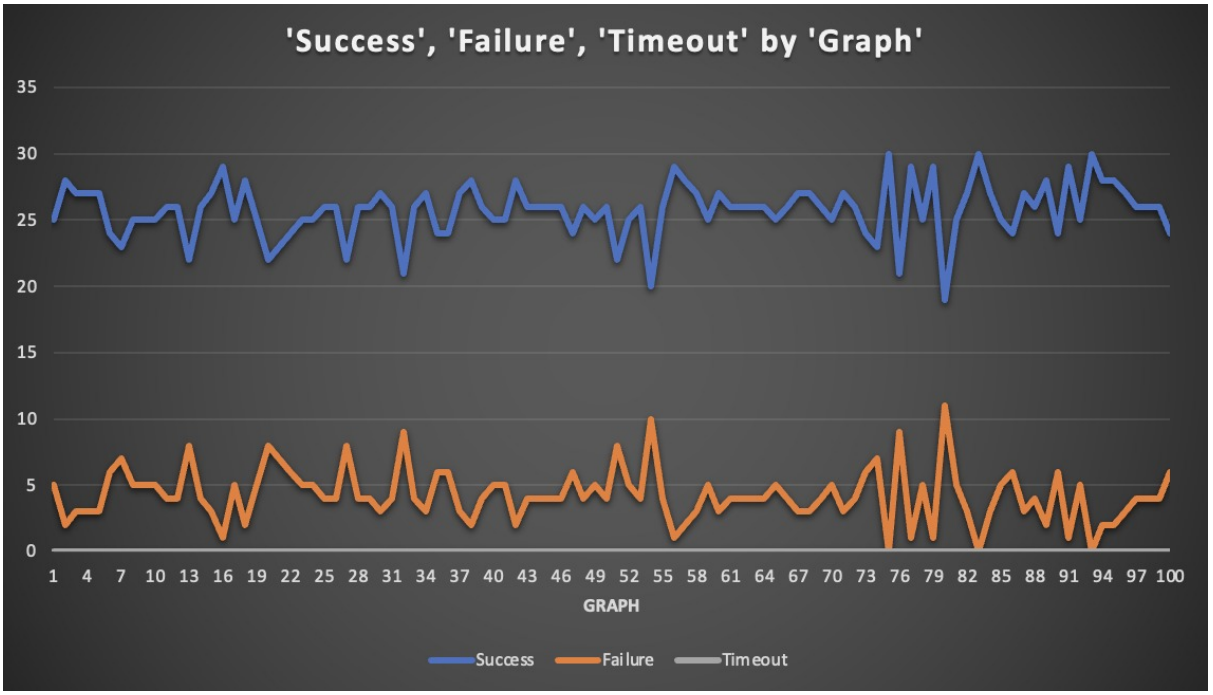
- a and f be x1
- a and d be x2
- a and e be x3

So the Average shortest distance from node a to the predator is (x1+ x2+ x3) /3.

➔ For this our agent 9, we move in accordance with the rules of agent 1 by the average distances that we calculated.

➔ We pick the neighbour with the max average distance to the predator and min average distance to the prey for our agent's movement.

Data Analysis of Agent 9:



1. Avg(Agent Success Rate): **85.8 %**
2. Avg(Agent Failure rate): **14.2 %**
3. Avg(Agent Timeout rate): **0.0 %**
4. Avg(Agents Correct Prey Survey): **1.0787921011952712 %**
5. Avg(Agents Correct Predator Survey): **33.52311075308615 %**