CITS3001 Project Report

Name- Harshawardhan Mane

Student no. – 23273893

A.I. modelling of a game scenario

2022

The Project required to make an A.I. based game model, that developed a scenario between three hypothetical countries. Country A and Country B were trying to influence the elections of the third Country C. The game had to be incorporated with a few assumptions and required to give a few desired outputs.

->Language Used – Python

->External libraries used – networkx and matplotlib to plot a randomly generated graph of n nodes, given the probability of connections being p. (G(n, p))

The basic structure of my game model asks for whether to play as the red team or the blue team. I will cover all of the sections of my implementation in detail.

Function -> play():

This function takes input from the user to play as either red, or blue team.

Function->simulate():

This function is the main body of my game model that allocates certain variables, fetches required data and parameters from the user and passes them to the required agents and functions.

This function incorporates the idea of 'energy level' of the blue team as well as the no-of-followers of the red team. Essentially my assumption is that the blue team will have a full energy bar=100 points. As the game requires the blue team to make more moves, depending upon what move the blue team makes and what potency of its move is, the blue team will lose a certain number of energy points. The same goes for the red team. Except my assumption is that the initial number of non voters->(100-voting_percentage), will be set as the followers of red team. Over the course of instantiations and teams taking turns, the red will lose some of its followers as well, although the rate has been set to very low because -> if the player choses to play as blue agent (most likely will), the player will set the initial voter percentage very high (in order to increase his chances of winning), for example 75% or so, which will be hazardous for the red team. One thing to note is that the blue team can regain some energy points based on what move the red team makes, i.e. a certain fraction of followers that the red will lose will be added to the energy counter of the blue team. The green population has been populated using the implementation of a Erdos-Renyi Graph, to randomly generate a given number of nodes n, with the probability of connections in the network being p. G(n, p). Every node in the green population will have an opinion (whether to vote or not to vote) and an uncertainty value associated with them. An initial iteration of my project implementation regarded uncertainty value as 'How uncertain the person is with their decision'.

For example->A person has a high uncertainty value of 0.7 and their opinion is to vote, by this we can infer that they are highly uncertain about their decision and will likely change their opinion to not vote once a person with low uncertainty value with opinion 'to vote' will connect with them. This implementation, however, was far too complicated and had to take many things into its consideration. Therefore, this idea of implementation was discarded, and I decided to move with a simple, clear and easy to explain implementation, i.e. with the assumption that opinion and uncertainty are not two completely different things but will be associated to each other. In explanatory terms, I assumed that a node with high uncertainty means he will not vote, and a node with low uncertainty means he will vote. Essentially, I initially assigned all the nodes in the network a voting opinion and an uncertainty value, which, after every iteration, would determine each other.

Let’s say for example-> initially all the green nodes had an uncertainty of 0.62 and all of them were assigned some opinion (to vote or not to vote).

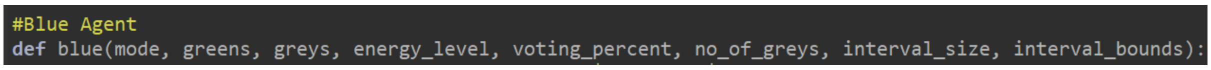
After the first turn taken by all the agents, the resulting green network has half the number of people with uncertainty 0.3 and half have uncertainty 0.7, so given my assumption, half of the nodes have their opinion to vote set to ‘yes’ and half of them do not opine to vote. These values thus, change constantly with each turn as the agents interact with the green nodes. To add more randomness and complexity, the opinions of green nodes were also influenced by the other connected green nodes every turn of the game. All the data about the green population was stored in a dictionary for later efficient usage. The no of grey agents and the proportion of them who are supposed to be spies, are to be provided by the user every instance of the game. This data is used to divide the greys in appropriately sized factions (grey-good and grey-bad). All the data about grey agents is stored in a dictionary as well for later use.

Agent Design - >

All the agents have been somewhat inspired from decision trees, where the agents calculate every move at their turn and given all the conditions (voting_percent in this case), will play the best move possible for them, or the move that will yield them the best result in the immediate forthcoming turn. All the data such as the uncertainty values of the green members, their opinion etc, is obscured from the agents and the move they make is dependant on what their previous move have had on the ’voting_percent’, the sole determining parameter in this case.

->Designing the blue agent.

The blue agent takes in a few parameters –



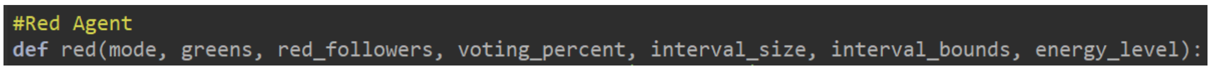
```
#Blue Agent
def blue(mode, greens, greys, energy_level, voting_percent, no_of_greys, interval_size, interval_bounds):
```

The mode parameter specifies whether the agent is being controlled by a human player, or is being run in automation (In the case when the player decides to play as a red agent). Greens and greys are data structures that hold information about the green and grey agents respectively. The interval_size parameter is calculated from the interval_bounds input that the user provided before starting the game. The blue agent displays on screen, a few options for the player to chose from, along with the associated uncertainty value and energy loss associated with the choice. The agent also displays useful metrics like voting_percent (percentage of voters currently in the green population), in order to aid the player to make their move. When the player makes a move using the blue agent, it loses some of its energy level based on the table coded in game. It is assumed that either the player will play a move that will have some effect on the green population, or will have the choice to bring in a grey agent, that will make decision on behalf of the blue agent. According to the project spec-sheet, the grey agent can be either friendly and make decision favourable for the blue agent or can be a spy for red team, in which case the move will have certain repercussions for blue team. More on the grey agents will be discussed later in the section dedicated to them. When the blue agent makes it’s turn, it has an uncertainty value/factor associated with the particular move it makes, that will determine the effect on the green node’s population. As discussed earlier, the blue agent’s job is to drag down the uncertainty level to minimum, i.e., convince them to vote (As stated in my assumption).

In case the user decides to play as red agent-> the blue agent has the option to run on automation and make the best possible move for its immediate future.

->Designing the red agent

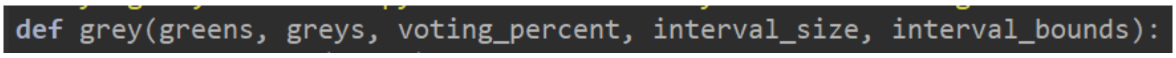
Similar to the blue agent, the red agent takes a few parameters as well –



```
#Red Agent
def red(mode, greens, red_followers, voting_percent, interval_size, interval_bounds, energy_level):
```

The parameters were mostly similar to the blue agent, while some of them being implemented later on for a purpose. The red_followers variable is decreased every time the red makes a move, according to the penalty schemed coded in the project, associated with every choice. The red agent is also passed on with the energy_level of the blue agent, to facilitate the gradual increase of blue's energy level over a few turns made by the red. Similar to the blue agent, the red agent is also capable of being played by a human player or in automation. The main job of the red agent is to push up the uncertainty, thereby, increasing the number of non-voter nodes (as mentioned previously in my assumption). Moreover, the functioning of red agent us very similar to the blue agent in implementation terms.

->Designing the grey agent



```
def grey(greens, greys, voting_percent, interval_size, interval_bounds):
```

The grey agent is called on by the blue agent, both when a human player is playing it or whether in automation. On implementation level, the grey agent is passed on with the greys dictionary, which has the data of all the grey agents introduced in the game at the beginning.

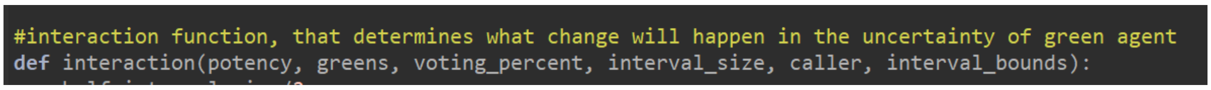
The grey agent implementer randomly selects one of the tokens from this dictionary: when the token is "grey_good", the grey agent takes decision in favour of the blue agent, i.e., to pull down the uncertainty of the green nodes. The magnitude by which the grey agent pulls down the uncertainty of the green nodes varies and is randomly selected. The magnitude will be the same as the uncertainty values table of the blue agent, just that the grey agent chooses one of it at random.

In the earlier implementation, I coded the grey agent to be making decisions not on random but from analysing the voting_percent, just as the blue agent does. But the implementation became too complicated and I dropped the plan.

Likewise, if the token fetched randomly from the dictionary turns out to be a "grey-bad", it will make moves in a similar way, just that it supports the red-agent now, and 'pushes' the uncertainty higher, in order for the green nodes to decide to not vote. The grey agent disappears once it's work has been done.

Once the number of grey_agents has been used up/exhausted, no more can it be called by the blue agent (both manual and automated). User will be notified with the warning that they are out of grey-agents, if they continue to use the option, their move will be wasted and the turn will keep moving on.

->Designing the interaction function



```
#interaction function, that determines what change will happen in the uncertainty of green agent
def interaction(potency, greens, voting_percent, interval_size, caller, interval_bounds):
```

The interaction function takes in the 'potency' or the uncertainty from any of the agents (red, blue or grey) and given all the variables, calculates what the next set of attributes would be for the green nodes. Essentially it is to implement the interaction between the blue agent-green agent, the red agent-green agent, and the grey agent- green agent. The function also makes sure that the uncertainty associated with each of the green nodes lies within the interval specified earlier by the player, before the start of the game.

->Designing the green function to emulate the interaction of green nodes amongst themselves and their connected neighbours.

The green function gets its own turn every round, and it basically emulates the exchange of uncertainties and opinions amongst the green nodes themselves. The connections between the specific green nodes are determined by the Graph generated earlier and will be used throughout the game. This green-to-green interaction is undirected, means any of the two connecting nodes can change the opinion of the other. The calculation in this part is determined by what uncertainty the two nodes have and how far their uncertainty values are from the median of the uncertainty interval.

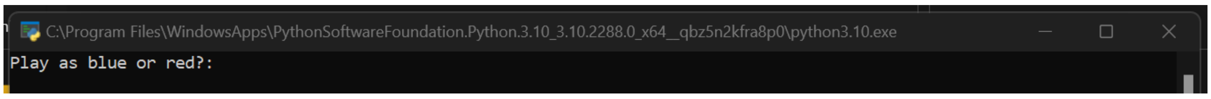
Essentially, the interaction between red/blue and green is Directed, i.e., only the red and the blue can change the opinion of the green agent and not vice versa. While, the interaction between green-green node is undirected and the two nodes can influence each other's uncertainties and opinions.

->The display() function

The display function displays the required graph and produces the output on the screen.

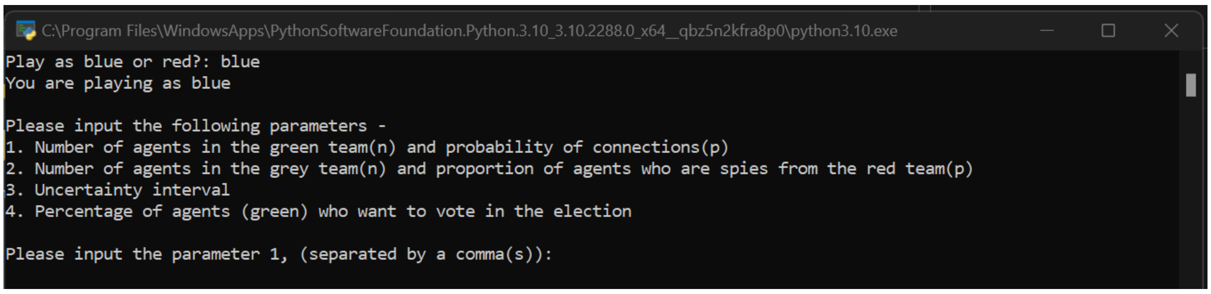
Instructions to play the game –

When you run the game.py file, the game will ask you about your choice (whether to play as blue agent or red agent).



```
C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.10_3.10.2288.0_x64__qbz5n2kfra8p0\python3.10.exe
Play as blue or red?:
```

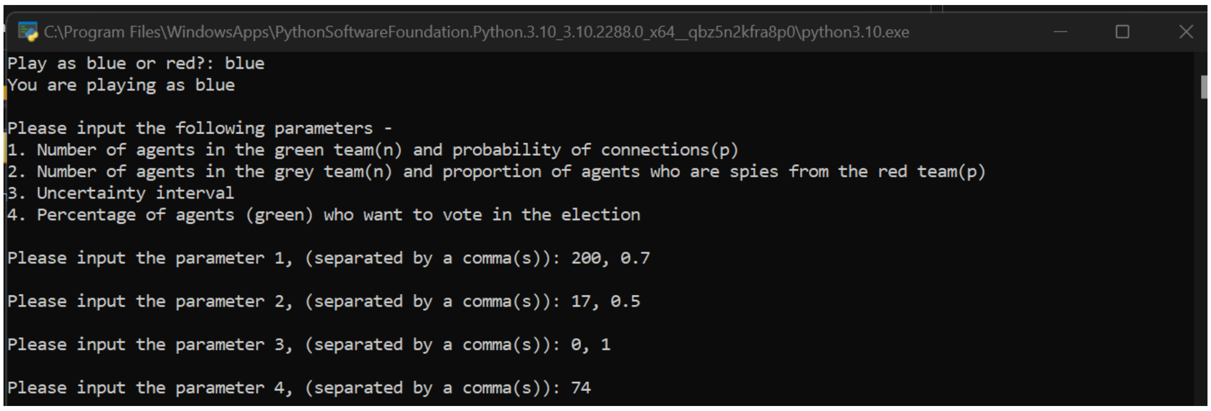
If you respond with blue/ red, it will further ask you to enter these parameters-



```
C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.10_3.10.2288.0_x64__qbz5n2kfra8p0\python3.10.exe
Play as blue or red?: blue
You are playing as blue

Please input the following parameters -
1. Number of agents in the green team(n) and probability of connections(p)
2. Number of agents in the grey team(n) and proportion of agents who are spies from the red team(p)
3. Uncertainty interval
4. Percentage of agents (green) who want to vote in the election

Please input the parameter 1, (separated by a comma(s)):
```

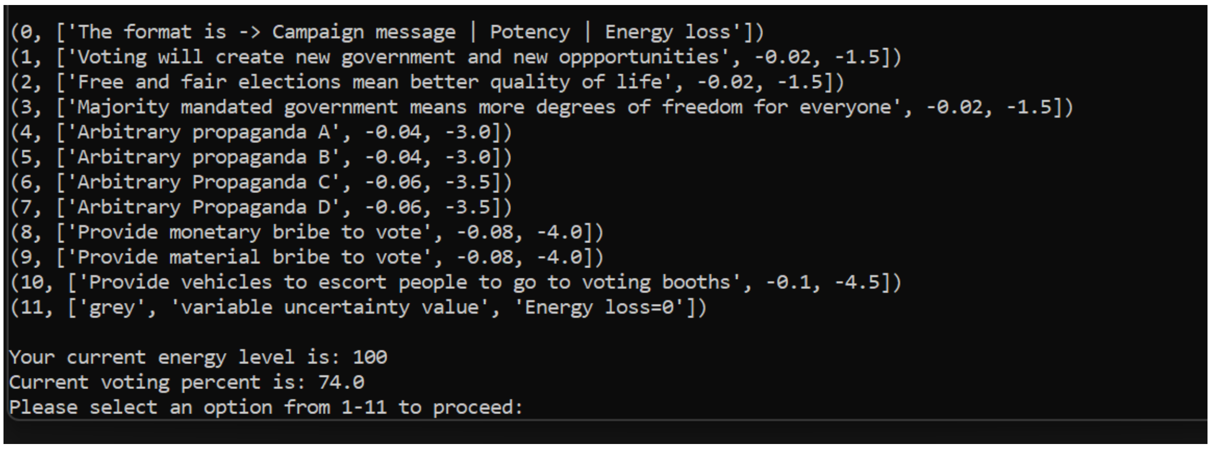
A valid set of inputs includes, but is not limited to –

Also, P.S. – that due to the way of construction of my project, it requires the lower bound of uncertainty interval to be greater than 0. Upper bound can be anything above that.



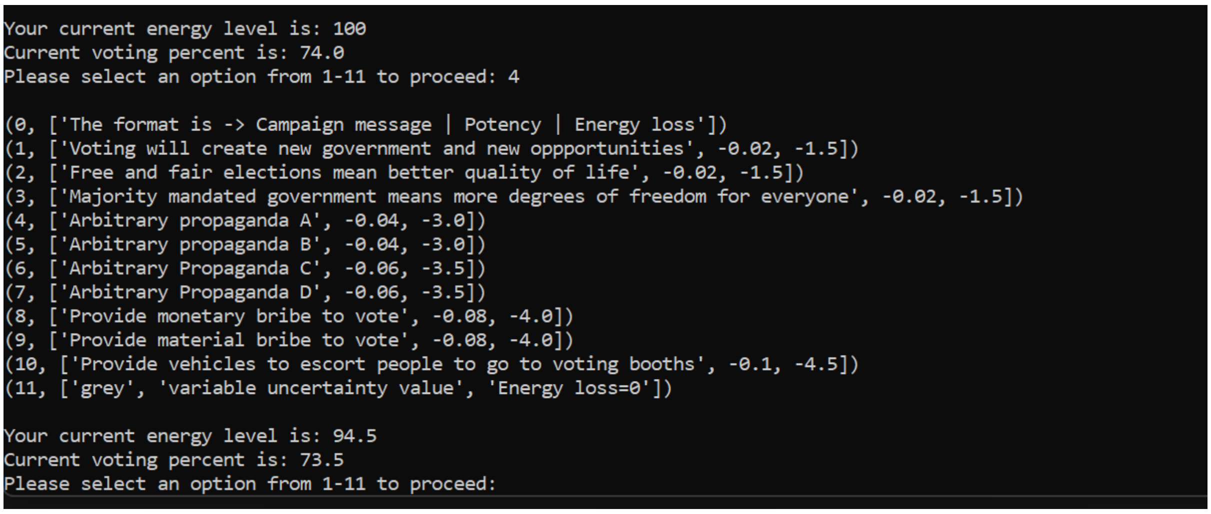
```
C:\Program Files\WindowsApps\PythonSoftwareFoundation.Python.3.10_3.10.2288.0_x64__qbz5n2kfra8p0\python3.10.exe
Play as blue or red?: blue
You are playing as blue

Please input the following parameters -
1. Number of agents in the green team(n) and probability of connections(p)
2. Number of agents in the grey team(n) and proportion of agents who are spies from the red team(p)
3. Uncertainty interval
4. Percentage of agents (green) who want to vote in the election

Please input the parameter 1, (separated by a comma(s)): 200, 0.7

Please input the parameter 2, (separated by a comma(s)): 17, 0.5

Please input the parameter 3, (separated by a comma(s)): 0, 1

Please input the parameter 4, (separated by a comma(s)): 74
```

The game then presents with a suite of options for the player to play as the agent of their choice (blue in this case.) –



```
(0, ['The format is -> Campaign message | Potency | Energy loss'])
(1, ['Voting will create new government and new oppportunities', -0.02, -1.5])
(2, ['Free and fair elections mean better quality of life', -0.02, -1.5])
(3, ['Majority mandated government means more degrees of freedom for everyone', -0.02, -1.5])
(4, ['Arbitrary propaganda A', -0.04, -3.0])
(5, ['Arbitrary propaganda B', -0.04, -3.0])
(6, ['Arbitrary Propaganda C', -0.06, -3.5])
(7, ['Arbitrary Propaganda D', -0.06, -3.5])
(8, ['Provide monetary bribe to vote', -0.08, -4.0])
(9, ['Provide material bribe to vote', -0.08, -4.0])
(10, ['Provide vehicles to escort people to go to voting booths', -0.1, -4.5])
(11, ['grey', 'variable uncertainty value', 'Energy loss=0'])

Your current energy level is: 100
Current voting percent is: 74.0
Please select an option from 1-11 to proceed:
```

Selecting one of the options (From 1-11), will initiate the game and after the player takes their turn, the red agent and the green agent also take their turn and return back to the player for further input (P.S. my choice was option 4 in this case, and the associated energy level and voting percent have changed)-



```
Your current energy level is: 100
Current voting percent is: 74.0
Please select an option from 1-11 to proceed: 4

(0, ['The format is -> Campaign message | Potency | Energy loss'])
(1, ['Voting will create new government and new oppportunities', -0.02, -1.5])
(2, ['Free and fair elections mean better quality of life', -0.02, -1.5])
(3, ['Majority mandated government means more degrees of freedom for everyone', -0.02, -1.5])
(4, ['Arbitrary propaganda A', -0.04, -3.0])
(5, ['Arbitrary propaganda B', -0.04, -3.0])
(6, ['Arbitrary Propaganda C', -0.06, -3.5])
(7, ['Arbitrary Propaganda D', -0.06, -3.5])
(8, ['Provide monetary bribe to vote', -0.08, -4.0])
(9, ['Provide material bribe to vote', -0.08, -4.0])
(10, ['Provide vehicles to escort people to go to voting booths', -0.1, -4.5])
(11, ['grey', 'variable uncertainty value', 'Energy loss=0'])

Your current energy level is: 94.5
Current voting percent is: 73.5
Please select an option from 1-11 to proceed:
```

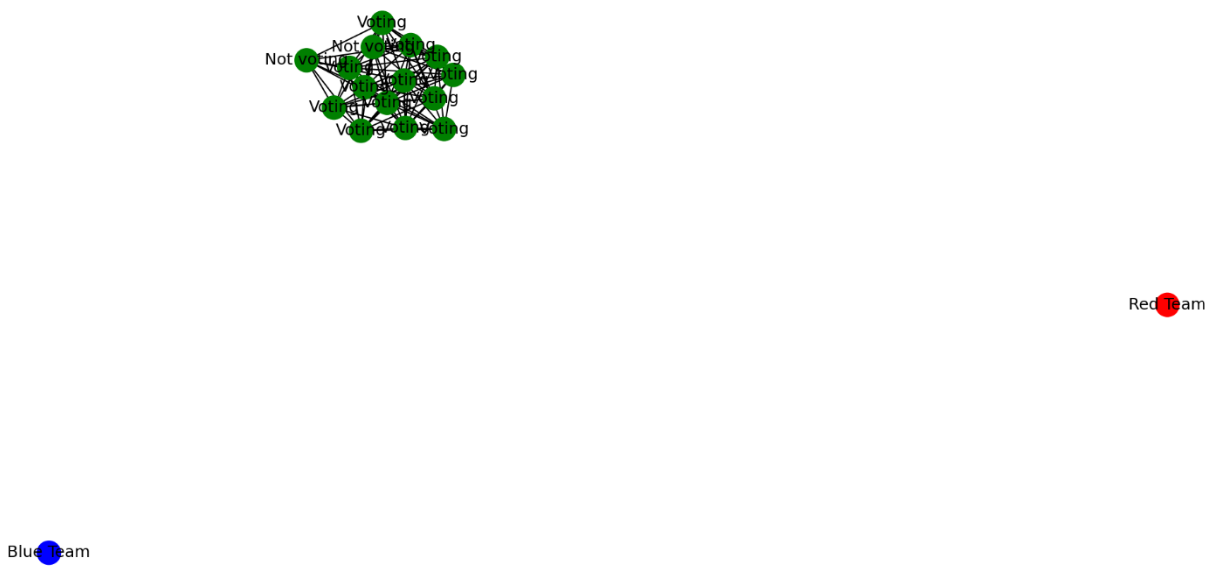
The game will continue to take inputs until the blue is drained out of it's energy level or red loses all of its followers.

The same game can be initiated by choosing to play as the red agent, in which case the player will be presented with a different set of options.

(P.S. the potency of the messages of the blue agent team are negative values as they are trying to pull down the uncertainty, while the potency of the red agent team are positive values, as they try to push the uncertainty of the green nodes towards the upper bound of the interval).

At the end of all the turns, the user will be presented with a graph that depicts which of the green nodes eventually want to vote and which nodes don't. –

(P.S. this is a graph with inputs with smaller values, i.e. ->(15 green nodes with connection probability of 0.7, 4 grey agents with half of them being bad, uncertainty interval [0, 1], percentage of voters in the beginning being 74%))



The project incorporates a lot of randomness/random variables so values may differ every time.

1. How does the game change if you have a tight uncertainty interval at the beginning of the game?

->Tighter uncertainty intervals exhaust the game relatively quickly, given the implementation of the project.

2. How does the game change if you have a broad uncertainty interval at the beginning of the game?

->Broader uncertainty intervals make the game last a little longer, in context of my implementation of the project.

3. In order for the Red agent to win, what is the best strategy?

->If playing as the red agent manually, the best strategy for it to win, in context of my project, is to set the initial voting percentage below 50%.

5. In order for the Blue agent to win, what is the best strategy?

-> If playing as the blue agent manually, the best strategy for it to win, in context of my project, is to set the initial voting percentage above 50%, and using highly potent messages to tackle the efforts of the red team and keeping the uncertainty to minimum as possible