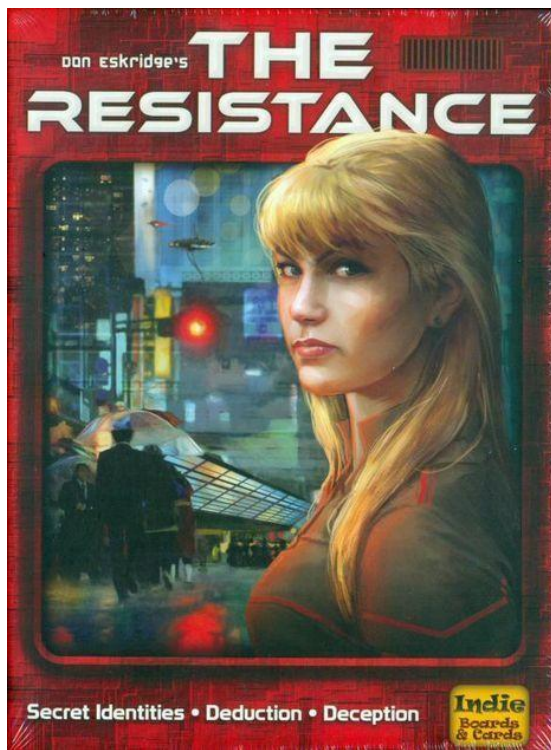


# The Resistance AI Agent

## Identifying Optimal Strategies in Imperfect Information Game

---

Xinyu Lei(22604588)



---

# Literature Review

This section generally presents the background information of this game, related developing techniques applied in the game and other relevant research. General topics included the basic information of the game “The resistance”, implementation of algorithms such as Tabu Search Algorithm.

## 1 The Resistance

### 1.1 Background of The Resistance

“The Resistance” is a card-based role-playing game suitable for 5 to 10 players. All players are divided into 2 different groups. One is the spy group and the other is the resistance member group. The resistance member group represents a group of people who want to overthrow the evil government. The spy group represents the evil government and wants to stop the resistance member group’s plot by infiltrating into the resistance member group.

### 1.2 Game Rules

The spies know about each other’s identities and the resistances don’t. There are 5 missions in one game and each mission has two statuses: success or failure. To win the game, the resistances have to achieve 3 successful missions and the spies have to fail the mission 3 times. Also, spies win if the formations of missions fail for 5 consecutive times.

Number of Resistance Members & Government Spies						
Number of players:	5	6	7	8	9	10
Resistance	3	4	4	5	6	6
Spies	2	2	3	3	3	4

(figure 1)

Number of players sent on each mission <sup>[2]</sup>				
Number of players:	5	6	7	8–10
Mission 1	2	2	2	3
Mission 2	3	3	3	4
Mission 3	2	4	3	4
Mission 4	3	3	4*	5*
Mission 5	3	4	4	5

(figure 2)

In each mission, there is a leader who is supposed to generate a team to carry on the mission. The number of people attending the mission varies depending on the game

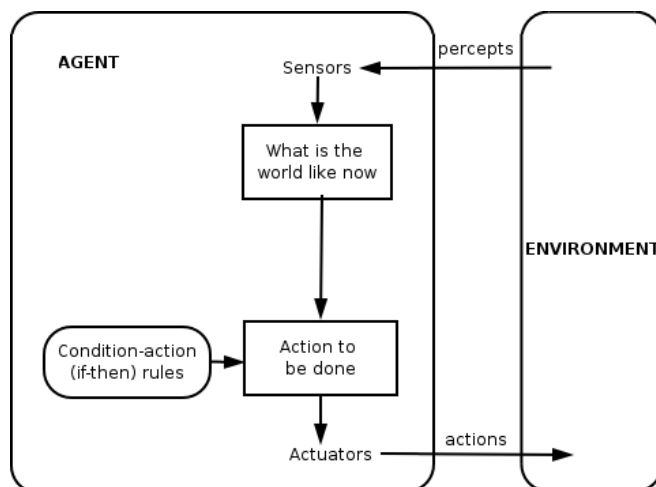
---

rules. The rest of the participants need to vote for or vote against the formation of the team. If the majority votes for it, the team is going to do the mission, if votes against, a new leader will be appointed and he has to generate a new team to wait for participants' approval or not. During the mission, the resistants have to vote for the mission by playing out the "True" card while the spies can either play out "False" to sabotage this mission or pretend themselves by playing out "True". (Wikipedia contributors, 2021)

### 1.3 Agent

In artificial intelligence, an agent is anything that can perceive the surrounding environment and automatically take action to achieve the expected goals and make progress by learning or using any knowledge.

Usually, an agent has an "Objective function" so that when the agent achieves its goal, the function will be generating a reward point to motivate the agent's action. (Wikipedia contributors, 2021)



simple reflex agent diagram (figure 3)

#### 1.3.1 A simple Agent in the game of The Resistance

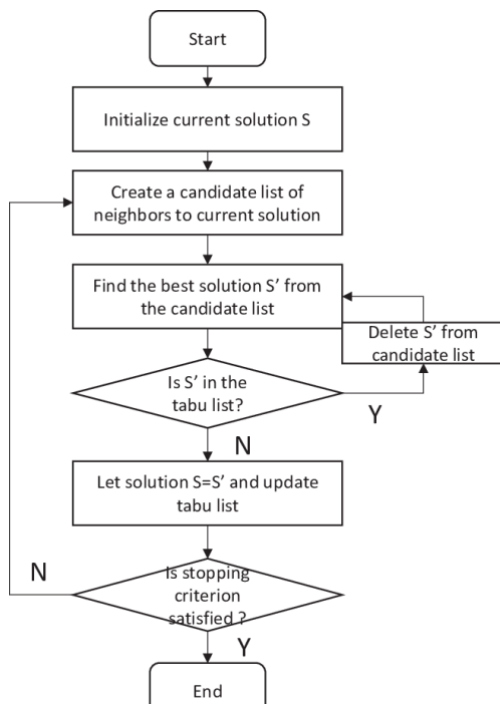
In this game, a simple agent can be introduced with human-like thinking logic. The agent can be designed by estimating the current fighting situations in one mission, then a scoring system will be added to the agent by adding or minus reward scores to mark the current situation and guide the agent to take the next action based on the scores.

## 1.4 Tabu Search Algorithm

Tabu Search Algorithm is a meta-heuristic searching technique created by Fred Glover that is used to prevent sticking in local optimal situations. This algorithm firstly creates a default doable plan and based on that, the algorithm chooses a series of specific searching movements as trials and then chooses the movements that make the value of objective functions change the most. To avoid getting stuck into poor scoring areas, this algorithm uses a flexible memorising technique to carefully explore the neighbourhood of every solution as the search progresses.

(Wikipedia contributors, 2021)

Flowchart of the tabu search algorithm  
(2021)



(figure 4)

Pseudocode of tabu search algorithm  
(Wikipedia contributors, 2021)

```
1 sBest ← s0
2 bestCandidate ← s0
3 tabuList ← []
4 tabuList.push(s0)
5 while (not stoppingCondition())
6     sNeighborhood ← getNeighbors(bestCandidate)
7     bestCandidate ← sNeighborhood[0]
8     for (sCandidate in sNeighborhood)
9         if ( (not tabuList.contains(sCandidate)) and (fitness(sCandidate) > fitness(bestCandidate)) )
10             bestCandidate ← sCandidate
11         end
12     end
13     if (fitness(bestCandidate) > fitness(sBest))
14         sBest ← bestCandidate
15     end
16     tabuList.push(bestCandidate)
17     if (tabuList.size > maxTabuSize)
18         tabuList.removeFirst()
19     end
20 end
21 return sBest
```

(figure 5)

---

#### 1.4.1 Tabu Search Algorithm in the game of The resistance

By reviewing the functionalities of a simple agent in this game as mentioned above, it is believed that the simple agent can be abstracted as a greedy algorithm. One significant disadvantage is that this algorithm only chooses the options that seem the best for the current step rather than considering what is for the best in the global situations. To be more specific, the greedy algorithm is highly likely to get stuck in a local optimal situation. Although there might be other algorithms that are ideal for the game like decision trees or genetic algorithms, the tabu search algorithm has the advantages of clear logic and lightweight code. It also has a unique technique to keep the local optimal record and jump out of the situation and keep looking for other optimates by moving elsewhere and improving the overall solution quality.

In the process of proposing a mission in the game, there is a gaming mechanism in it. One example is like choosing 3 unique agents' indexes in 7 different agents' indexes to attend the game. Hence, it is clear that the results of the game are with highly regard to such a mechanism. And such mechanisms can produce large quantities of possibilities. As in TSP problems, where there also exists a large quantity of possibilities to specify to make the best move and one good method is the tabu search algorithm. Given the facts that both TSP and the resistance game have similar possibility-decision issues to handle, it is believed that the tabu search algorithm can also be applied in this game. In this situation, the tabu search algorithm can be an ideal algorithm for sorting out the optimal next movement with a well designed heuristics. However, the tabu search algorithm's performance may vary depending on the design of heuristics, which can be a research interest for this game.

By researching, tabu list is one of the key components for this algorithm. The tabu list is the foundation of the use of short-term memory to prevent the algorithm from getting stuck into local optimal solutions. It records a fixed amount (called tabu length) of recently made moves. These elements won't be taken into consideration during the next search. The elements that are forbidden searching are also called tabu objects. (Liang, 2020) The tabu length is the max number of tabu objects that can be put into the tabu list. According to the resources, the tabu length is an important variable during the search. Both of the situations can cause the inaccuracy of the results if the length is too long or too short.

---

Tabu length can be an interesting research target in this game. It is assumed that the game system would consume too much time to run or the algorithm might be getting aborted from timeout, and it might cause the elements to be repeatedly searched if the length is too short.

## Introduction

### OVERVIEW

This report discusses the python implementation of the tabu search algorithm in the project and what factors of this algorithm that can improve the performance of the resistance agent in the game.

Our project AI mainly focuses on the situation when our AI is allocated the role of resistance by the shuffle system, and how the AI acts smartly to eventually win the game rounds by analysing every limited piece of useful information.

When exploring the template codes provided in Figure 1, we found it interesting that in the *Game.py* file, the spies always know what other agents' role is since the function always provide the list of all spies' indexes to the current spy if the current agent is spy, which means the agent has a perfect information version of the game. However, if the current AI represents a resistance member, the system won't be providing any additional information about other AI's roles, i.e. it is called imperfect information.



(Figure 6)

In this situation, the resistance member AI can be described as blind since it doesn't know who the opponent is or who the resistance member is, while the spy knows everything and it is much easier to anticipate other AIs' actions. Viewed from both

perspectives, the resistance member is in the position of having imperfect information, which requires the agent to actively search and organize the information to justify the roles of every participating AI in order to improve its probabilities to perfect the game as closely as possible. However, the spy AI already knows every participating agent's role and its job is to prevent the resistance member agent from winning, which means that the spy cuts off the step of blindly searching for all role allocation possibilities of all agents, which is an adverse situation for the resistant agents.

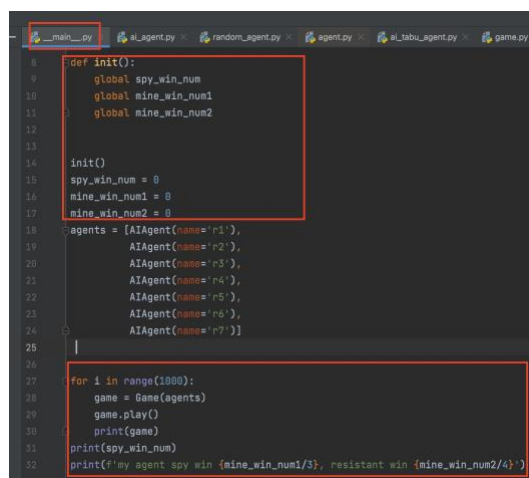
## Analysis

### Part A

To make up for the adverse situation and turn it into an advantage for a resistant agent, we decided that we started with building human-like logic to strengthen the resistant agents.

First, we need to know the performance of this default RandomAgent() by calculating the resistant winning rate and spy winning rate in the whole game. To make the data more accurate, we decided to let the game run 1000 times and keep doing so a couple of times.

(figure 7)

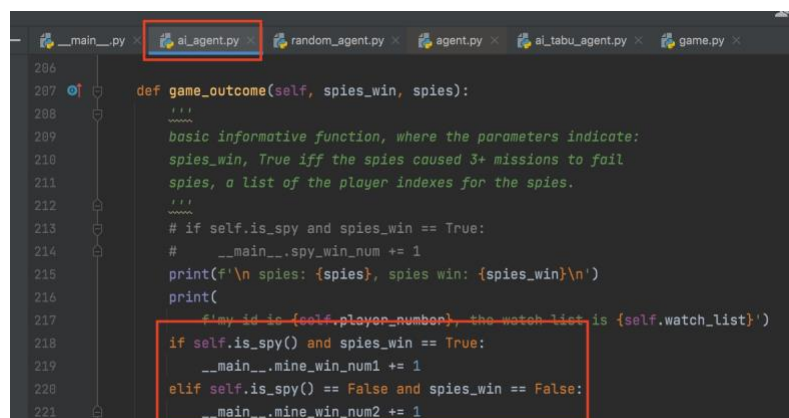


```

8 def init():
9     global spy_win_num
10    global mine_win_num1
11    global mine_win_num2
12
13
14    init()
15    spy_win_num = 0
16    mine_win_num1 = 0
17    mine_win_num2 = 0
18
19    agents = [AIAgent(name='r1'),
20              AIAgent(name='r2'),
21              AIAgent(name='r3'),
22              AIAgent(name='r4'),
23              AIAgent(name='r5'),
24              AIAgent(name='r6'),
25              AIAgent(name='r7')]
26
27
28    for i in range(1000):
29        game = Game(agents)
30        game.play()
31        print(game)
32        print(spy_win_num)
33        print('my agent spy win {mine_win_num1/3}, resistant win {mine_win_num2/4}')

```

(figure 8)



```

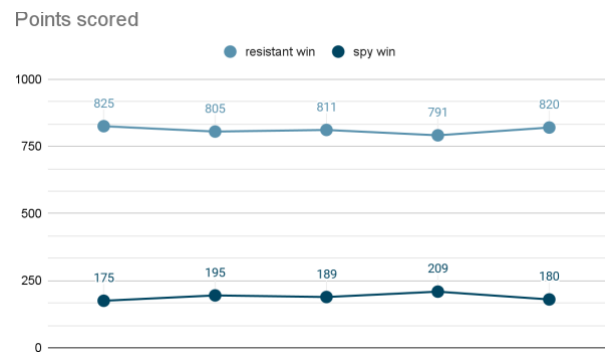
206
207
208 def game_outcome(self, spies_win, spies):
209     """
210     basic informative function, where the parameters indicate:
211     spies_win, True iff the spies caused 3+ missions to fail
212     spies, a list of the player indexes for the spies.
213     """
214     # if self.is_spy and spies_win == True:
215     #     __main__.spy_win_num += 1
216     print(f'\n spies: {spies}, spies win: {spies_win}\n')
217     print(
218         f'my id is {self.player_number}, the watch list is {self.watch_list}')
219     if self.is_spy() and spies_win == True:
220         __main__.mine_win_num1 += 1
221     elif self.is_spy() == False and spies_win == False:
222         __main__.mine_win_num2 += 1

```

---

```
spy wins: 175.0 /1000 win rate: 17.5 %  
resistant wins: 825.0 /1000 win rate: 82.5 %
```

(figure 9)



(figure 10)

Hence, we got some results from the game system in figure 10, the resistant winning average rate is 81.04 %. The spy winning rate is only 18.96 % in 1000 games, which seems that the resistant agent has very good performance. However, by inspecting the code, the spy has a 70% chance that it wouldn't betray the mission and the mission proposal has a 50 % chance to pass. It is obvious to find out that both resistant and spy agents don't have any human-likely thinking logic, which is a good point to work on.

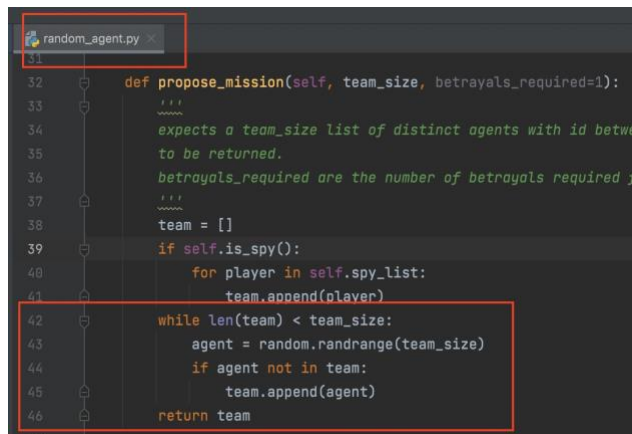
## Part B

Other than random, we added the basic logic that the resistant agent will keep a dictionary to themselves to record their opponents move. The resistant agent is playing an imperfect version of the game for it doesn't know who the other agents are except by



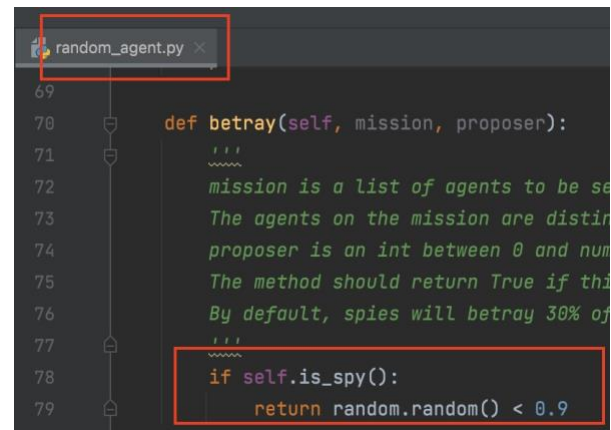
observing their actions during each mission and using some technical mechanism to determine their roles. Hence, the missing information can be gathered by storing observed data into the dictionary and analyzing it before making decisions regarding the next move.

To make the resistant agent get the best performance, it is important that the spy agent needs to be compatible. So we made improvements on spy logic.



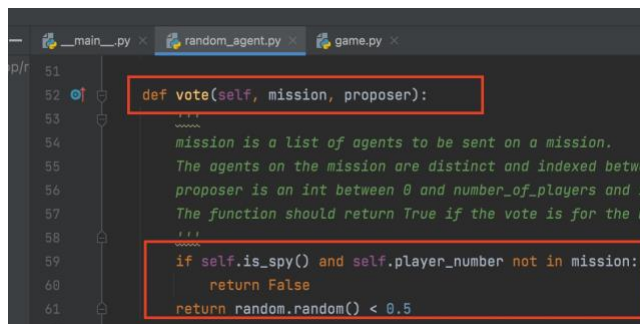
```
31 random_agent.py
32 def propose_mission(self, team_size, betrayals_required=1):
33     """
34     expects a team_size list of distinct agents with id between
35     to be returned.
36     betrayals_required are the number of betrayals required for
37     """
38     team = []
39     if self.is_spy():
40         for player in self.spy_list:
41             team.append(player)
42     while len(team) < team_size:
43         agent = random.randrange(team_size)
44         if agent not in team:
45             team.append(agent)
46     return team
```

(figure 11 )



```
69 random_agent.py
70 def betray(self, mission, proposer):
71     """
72     mission is a list of agents to be sent on a mission.
73     The agents on the mission are distinct and indexed between
74     proposer is an int between 0 and number_of_players and i
75     The method should return True if the agent betrays the mission
76     By default, spies will betray 30% of the time
77     """
78     if self.is_spy():
79         return random.random() < 0.9
```

(figure 12)

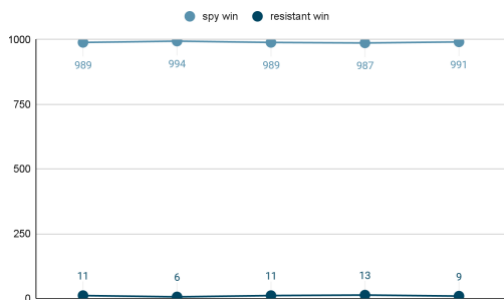


```
51 main.py random_agent.py game.py
52 def vote(self, mission, proposer):
53     """
54     mission is a list of agents to be sent on a mission.
55     The agents on the mission are distinct and indexed between
56     proposer is an int between 0 and number_of_players and i
57     The function should return True if the vote is for the mission
58     """
59     if self.is_spy() and self.player_number not in mission:
60         return False
61     return random.random() < 0.5
```

(figure 13)

From the 2 figures shown above, we added logic that during the process of proposing a mission, if the current agent is a spy, then it would propose all its spy team members into the mission to increase the spy winning chance. (figure 11)The second logic is that if a spy agent is on a mission, it has a 90% chance to betray the team. (figure 12) The third logic is that if the current agent is a spy and it's not on a mission, it will vote negatively(figure 13).

By testing, the average spy winning rate increased to 98% as is shown below (figure 14), which increased 79.04% by comparing with the default spy agent.



(figure 14)

Under this circumstance, it is natural to optimize the resistance logic. We designed a scoring dictionary called `watch_list` in `ai_agent.py`.

```
def new_game(self, number_of_players, player_number, spy_list):
    """
    initialises the game, informing the agent of the
    number_of_players, the player_number (an id number for the age
    and a list of agent indexes which are the spies, if the agent
    """
    self.number_of_players = number_of_players
    self.player_number = player_number
    self.spy_list = spy_list

    temp_list = []
    round = 0
    mission_abort = 0
    for i in range(number_of_players):
        temp_list.append(i)
    watch_list = dict.fromkeys(temp_list, 0)
    self.watch_list = watch_list
    self.round = round
    self.mission_abort = mission_abort
    print(f"My id is {self.player_number}")
    if (self.player_number in self.spy_list):
        print("I'm a spy")
```

(figure 15)

```
def mission_outcome(self, mission, proposer, betrayals, mission_success):
    print(f'mission list is {mission}')
    if self.is_spy() != True and betrayals != 0:
        for opponent in mission:
            if opponent != self.player_number: # no need to add value fo
                self.watch_list[opponent] += 0.2
            if opponent != self.player_number and self.round == 0:
                self.watch_list[opponent] += 0.2
```

(figure 16)

```

44 def propose_mission(self, team_size, betrayals_required=1):
45     team = []
46     # find the biggest number in watch_list
47     max = 0
48     for player in self.watch_list:
49         if max < self.watch_list[player]:
50             max = self.watch_list[player]
51     # try to filter those players who have the highest mark and vote them out
52     guess_spy_list = []
53     for player in self.watch_list:
54         if self.watch_list[player] == max and max != 0:
55             guess_spy_list.append(player)
56     # add logic into the agent when to propose its allies
57     if self.is_spy():
58         for allies in self.spy_list:
59             team.append(allies)
60         while len(team) < team_size:
61             agent = random.randrange(team_size)
62             if agent not in team:
63                 team.append(agent)
64         return team
65     else:
66         team.append(self.player_number)
67         counter = 0 # used to stop infinite loop
68         while len(team) < team_size:
69             counter += 1
70             agent = random.randrange(team_size)
71             if agent not in team and agent not in guess_spy_list and counter <= 40:
72                 team.append(agent)
73             continue
74             if counter > 40:
75                 team.append(agent)
76         return team

```

(figure 17)

```

71 def vote(self, mission, proposer):
72     vote_false = 0.5
73     max_key = -1
74     if self.is_spy() and self.player_number not in mission:
75         return False
76     elif self.is_spy() and self.mission_abort >= 3:
77         return False
78     elif self.is_spy() == False and self.round == 0:
79         return True
80     # find the biggest number in watch_list
81     max = 0
82     for player in self.watch_list:
83         if max < self.watch_list[player]:
84             max = self.watch_list[player]
85     # try to filter those players who have the highest mark and vote them out
86     guess_spy_list = []
87     for player in self.watch_list:
88         if self.watch_list[player] == max and max != 0:
89             guess_spy_list.append(player)
90     print(f'spy list is {guess_spy_list}')
91     if self.is_spy() != True and max != 0:
92         if proposer in guess_spy_list:
93             return False
94         for player in guess_spy_list:
95             if player in mission and self.mission_abort < 5:
96                 vote_false = 0.1
97                 break
98             elif player in mission:
99                 vote_false = 1
100                 break
101     return random.random() < vote_false

```

(figure 18)

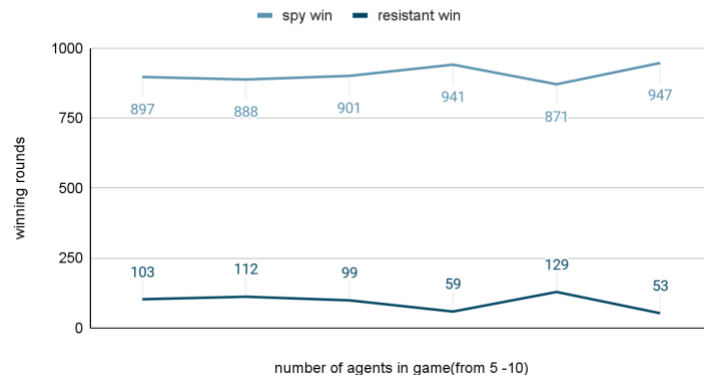
When one mission ends, if the current agent is resistant and loses this round, it means there is at least one spy in this mission. However, since the resistant agent has the property of imperfect information, it is impossible to know which agent betrayed. So this current resistant agent will suspect all participating agents except itself and give them a well-designed penalty score in the watch\_list (figure 16) to show the possibilities of being suspected by the current resistant agent.

By keep suspecting the agents according to this logic, when the next mission comes, the current resistant agent will search the whole watch\_list and choose the key (the index of the agent) that its value is the max among all elements in the watch\_list dictionary(figure 17), and try to stop this agent getting into this mission through the index because it has the highest score and it has the biggest possibility to be a spy. (figure 18)

Since we add the watch\_list into the resistant agent, and according to the mechanism, ideally it can detect the suspect agents that threw negative votes to destroy the mission by accumulating penalty points. And according to the running results, the average spy winning rate dropped down to 90.74% between 5 to 10 agents, and the average resistant agent increased to 9.25%, which increased 7.25% by comparing with the former running result.

---

average spy win rate: 90.74%, resistant win rate 9.25%



(figure 19 )

However, we believe the resistant agent performance is still at an initial stage. Looking from the output of the watch\_list,(figure 19) we found that the self-suspecting mechanism isn't accurate because it contains not only spies but also resistants in it, where there shouldn't be any resistants ideally.

Given the testing result that the resistant agents don't have a high accuracy, we assume that tabu search algorithm may be helpful, which is the next section we will be discussing.

## Part C

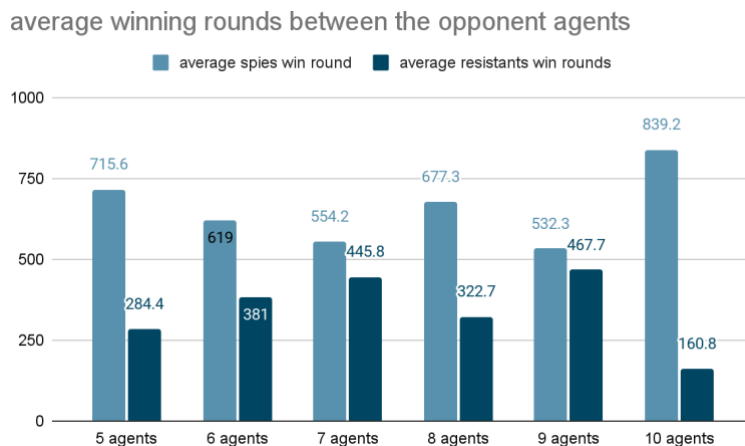
Next, We implemented the tabu search algorithm based on human-likely thinking logic. This algorithm is used to help the resistance agents while they need to select a group of people to go on a mission. The goal here is to minimize the chance of putting one or more spies into the mission list, therefore maximize the resistance win rate. The implementation is based on the pseudocode (figure 5)from this page and some modifications:

Tabu search algorithm does the following:

1. Take a random input as its starting point
2. Evaluate the input based on the heuristic we provides
3. Starting to swap the element within the input and the candidate (the other possibilities)
4. Evaluate the new result after swapping

5. If the new result is better, store the result.
6. repeat step 3 to 5
7. After swapping every possibility, compare the lowest result among the past round with the initial input, if the new one is better, change the return value (i.e., best solution) to the newer one and store the swapping into tabu list to avoid doing it again.
8. Checking the length of the tabu list. If it's too long, pop out the first one added into the list.
9. repeat step 3 - 8 until reaches termination condition.

After the implementation of this algorithm, we found that the average win rate of resistance agents greatly improved from 9.25% to 34.38%. By inspecting the output from the tabu search algorithm, it's not hard to find that the team list produced have higher accuracy compared to just look at the watch list (i.e., fewer spies added into the mission), which is the result of minimizing local optima.



(figure 20)

avg spy win round: 656.2, win rate: 65.62%

avg resistant win round: 343.8 win rate 34.38 %

## C.1 Setting Variables Impact

Even though the tabu search algorithm improved our resistance agent performance, whether the algorithm will have the same performance while competing with different strategies spy agents remains unknown. Plus, while implementing the algorithm, the

---

length of the labu list and the number of searching rounds are flexible, which means there are no specific requirements for those two numbers and can be potential unstable factors for the algorithm's performance.

The first question is how the number of rounds and the length of the tabu list will impact our performance. To answer this question, it is necessary to divide it into two parts: the impact of the number of rounds, and the impact of length of the tabu list.

### C.1.1

The number of rounds controls how many times the algorithm repeats its logic, which does not change any logic of the algorithm itself. If the algorithm stops too early, since tabu search is a hill climb algorithm, it may stop half way of finding the optimal solution. If the algorithm runs too many times, which is unnecessary and slows the agent down. Our goal is to find an optimal round number that doesn't impact both finding the solution and the performance of running the algorithm.

To answer this question, we used the same spy logic throughout the testing process. We will explain this idea by showing an example: playing a round game with 7-agents. There are seven players(four resistance members and three spies). All of them are using the same spy and resistance logic. So they are basically demonstrating whether the resistant part or the spy part of the agent is stronger, which is shown by counting the winning number for resistance and spy in 1000 games respectively. Before the experiment starts, the average of those numbers is resistance: (466,464,479,439,462) ->462,spy: (534,536,521,561,538) ->538. The ROUND number for this result is 50. The number 50 will be changed to both lower and higher numbers and the corresponding win rate will be recorded as well.

50->1

spy: (631,640,624,646,629) ->634 resistance: (369,360,376,354,371) ->366

50->10

spy: (563,551,524,554,560) ->550.4 resistance: (437,449,476,446,440) ->449.6

50->15

spy: (548,500,521,527,542) ->527.6 resistance: (452,500,479,473,458) ->472.4

50->20

spy: (545,538,509,549,581) ->544.4 resistance: (455,462,491,451,419) ->455.6

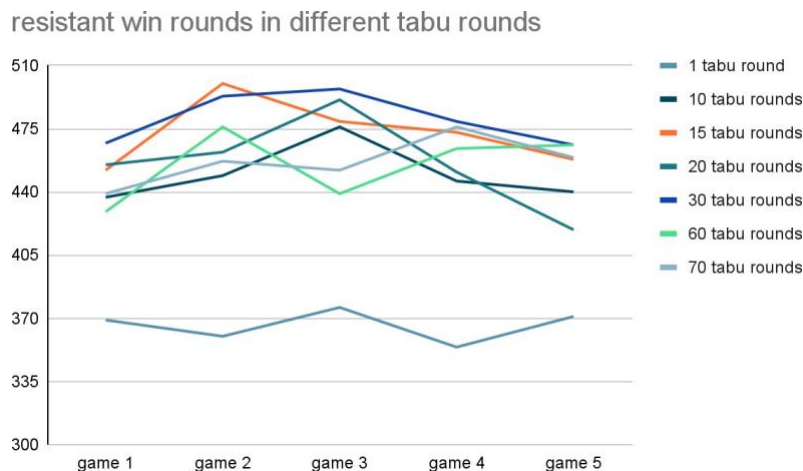
50->30

spy: (533,507,503,521,534) ->519.6 resistance: (467,493,497,479,466) ->480.4

50->60

spy: (571,524,561,536,534) ->545.2 resistance: (429,476,439,464,466) ->454.8  
 50->70  
 spy: (561,543,548,524,541) ->543.4 resistance: (439,457,452,476,459) ->456.6

The result showed us that the bigger round numbers have an effect within the range of 1-10. However, after 10 rounds, improvement is limited to a small amount and even with few reduced performance situations. (figure 21) The number is confirming our assumption that too many rounds are not helpful under the use of the resistance game. The experiment also has a slower result producing speed when the round number changes to 100 or 200, with no improvement in performance at all. From this point, the round number will be set to 30, which is a sweet spot between 10 and 50, with a highest performance as well.



(figure 21)

### C.1.2

The length of the tabu list controls how many items that will not be examined while searching. Too many items in this list may result in longer searching time or no item to search at all when all the items are in the tabu list. Insufficient length numbers in the list will result in repeated search. Both of those results are not preferred to occur.

The goal at this stage is to find the optimal tabu list length that is not too small or too large.

To answer this question, we used the same spy logic throughout the testing process and testing through a 7-agents game. Round number is set to 30 according to the research in Part 1. Before the experiment starts, the average winning rounds for both spy and the resistant before testing are: resistance: (488,477,473,448,461) ->469.4,spy: (512,523,527,552,539) ->530.6. During the experiment, the tabu list length is flexible in each testing. The default length of the tabu list is based on the length of half the mission team list and round it to an integer. The length integer will be changed to both lower and higher numbers and the corresponding winning rate will be recorded as well.

Minimizing the tabu list length:

Dynamic -> 1

spy: (545,523,522,560,550) ->540 resistance: (455,477,478,440,450) ->460

Dynamic -> 2

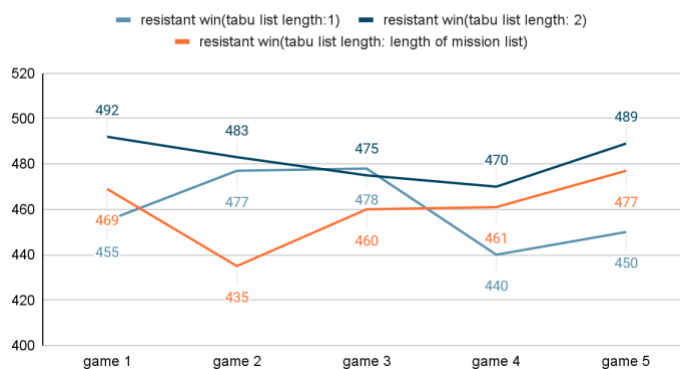
spy: (508,517,525,530,511) ->518.2 resistance: (492,483,475,470,489) ->481.8

Maximizing the tabu list length:

Dynamic -> length of mission list

spy: (531,565,540,539,523) ->539.6 resistance: (469,435,460,461,477) ->460.4

resistant win rounds with different tabu list lengths



(figure 22)

As the results indicate above, again, the changes in performance are not significant. Too big or small configurations for the tabu list length are not helpful in terms of improving the algorithm overall performance. However, the sweet spot found for the current agent is to set the tabu list length to 2, which always keeps two items that were searched before.



---

## C.2 Heuristic

Heuristic, which is the accuracy of the watchlist the agent used, will have a large impact on the tabu search algorithm. Tabu search will rely on the provided heuristic and evaluate each solution it has searched according to this “marking key”. For example, if a player has done suspicious behaviours like being a member of the team and the mission fails, this player will be given higher marks in the watchlist.

However, this logic will take innocent resistance players into account as well. The watchlist will also be useless while all the members in a mission team have the same mark in the heuristic and the tabu algorithm will also fail to search for an optimal solution under circumstances like this. And further testing proved that by having the tabu search return an empty list while every key in the watch list has the equal non zero value(i.e., 0:0.2, 1:0.2, 2:0.2, 3:0.2), and there is no such key that has the highest mark among all elements.

To test the accuracy and the logic of heuristic’s impact on the tabu search algorithm, the base winning performance of the resistance agent is 476 among 1000 games. If we add logic like: if the mission failed, the proposer itself will also be suspected by the current resistance agent and the current resistance agent will add the marks in its own watchlist:

```
self.watch_list[proposer] += 0.1
```

The result showed a resistance agent winning 64 games among 1000, which showed a massive hit in performance. The reason here is still because of the imprecise logic. By looking at agent.py, we knew the game rules that there are 3 spies and 4 resistance agents in a 7-agents game. Hence, there are 4/7 chances that the proposer is a resistance agent during the game. By brute forcibly accumulating the suspecting marks to all proposers is highly likely to affect an innocent resistance agent and make a false evaluation by the tabu search algorithm.

In this case, it would be reasonable that we could add another logic to restrict the suspect of proposers in the possibility of 3/7 since the possibility of a spy proposer is only 3/7:

```
if mission_success == False and random.random() < 0.428:  
    self.watch_list[proposer] += 0.1
```

---

By testing, the resistant agents' winning rounds increased to 396 among 1000 rounds of games, which is a huge leap by comparison with the testing before, which in that situation, the resistance agents only won 64 rounds among 1000 rounds of games. But still, there is still a gap between this testing result and the testing that without the extra logic because that one won 476 rounds, which means the resistance agents has an extra 8% performance.

Besides the logic itself, our heuristic also has taken the advantage of aggressive logic spies. In the previous testing, the spy agent logic is to always vote negative when they are on a mission and always vote against a mission proposal that does not include them.

Furthermore, the current spy agent will also add all its spy allies into the mission team when it is a leader. However, once the spy agent's logic has been completely redesigned, the resistance agents' performance may also be reduced.

This time, the experiment will be held in another python file and the goal is to make the spy less aggressive to see what the impact is on the heuristic and tabu search algorithm.

To improve the spy agent, the first part is to change the logic of voting in a mission team. The spy can be less aggressive by simply reducing the negative voting rate (i.e., voting fails half the time). If the number of spies in this team is higher than the number that require the mission to fail, vote pass, otherwise vote fail. This logic is trying to confuse the other resistance member like the case:

(note: 1 and 2 are spies ), (1, 2) -> success (1, 3) -> fail (2, 3) -> fail

But the actual spies here are 1 and 2. However, 3 will have the most suspicious mark in the watch list since 3 failed missions twice. In addition, the spy can also propose a team with only one to two spies into the mission at 70% of the time, which aims to increase resistance's confusion.

Under such brand new logic with the implementation of the spy agent, if resistance agents using the current heuristic and using tabu search to play against spies, the winning rate of resistance dropped to 202 rounds in 1000 rounds of the games. By comparison with the original, which won 476 rounds, it is again, a massive hit in performance.

The less aggressive spy makes the watchlist heuristic mechanism even less accurate and confuses the following tabu search as well, which results in the loss of performance.

---

### C.3 Future Work

At the final stage of the agent implementation and experimentation, our agent's main aspect of improvement is heuristic. Heuristic now is still based on various conditions and states. The agent can evaluate and take actions based on the limited information provided. Inaccurate or insufficient heuristic will have the performance being affected since all the following algorithm is based on the provided heuristic, which makes the improvement of the performance very limited.

To completely change the situation, the agent needs to be able to study and improve the heuristic itself based on algorithms and models, instead of adding more algorithms based on a single base heuristic. Possible approaches may apply to machine learning or deep learning, which may effectively bring the performance of the agent to a whole new level.

---

## Reference

1. Liang, F. (2020, July 27). Optimization Techniques — Tabu Search - Towards Data Science. Medium. <https://towardsdatascience.com/optimization-techniques-tabu-search-36f197ef8e25>
2. (n.d.). [https://www.researchgate.net/figure/Flowchart-of-tabu-search-algorithm\\_fig1\\_320508257](https://www.researchgate.net/figure/Flowchart-of-tabu-search-algorithm_fig1_320508257))
3. Wikipedia contributors. (2021, May 9). Tabu search. Wikipedia. [https://en.wikipedia.org/wiki/Tabu\\_search](https://en.wikipedia.org/wiki/Tabu_search)
4. Wikipedia contributors. (2021, May 26). The Resistance (game). Wikipedia. [https://en.wikipedia.org/wiki/The\\_Resistance\\_\(game\)](https://en.wikipedia.org/wiki/The_Resistance_(game))
5. Wikipedia contributors. (2021, October 18). Intelligent agent. Wikipedia. [https://en.wikipedia.org/wiki/Intelligent\\_agent](https://en.wikipedia.org/wiki/Intelligent_agent)