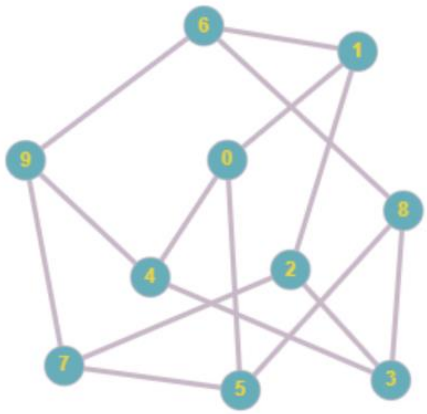
ADSA Mini Problem

Step 2: Professor Layton < Guybrush Threepwood < You :

1. Represent the relation (have seen) between players as a graph, argue about your model.



This graph represents the relation describe in the Exercise 2

We decide to use an unweighted graph in order to represent the relation between players because in this case this is not a quantifiable relation. Each node represents a player. We decide to use an undirected graph because if a first player have seen a second player, the second player have also seen the first player.

Then we decide to represent this graph with an adjacency list because it is the most convenient way to illustrate our graph for this exercise. Indeed, in this exercise, we care about the relation between players, so we need information about the vertices.

2. Thanks to a graph theory problem, present how to find a set of probable impostors.

Two players are connected if they have seen each other. We know that impostors never walk together, so our first hypothesis is that impostors never seen each other.

Considering the graph coloring algorithm, if two players are not colored with the same color, we can conclude that these players are connected and so, they cannot be both impostors. Consequently, two impostors are colored with the same color.

If we know which player is the first impostor, it is easy to figure out which player is the second impostor. We look which player are colored with the same color that the first impostor.

A graph can be colored in many ways, that is why we must run our algorithm more than one time in order to have different solutions and to set probabilities.

3. Argue about an algorithm solving your problem.

The biggest problematic is to find a good heuristic for our problem. In graph coloring, the role of the heuristic is to order vertices in some way in order to color them. Heuristic is important because results and performance of this algorithm depend on the order of the vertices.

Our first thought is to arbitrary order of the vertices. After some research, we figured out that this is a heuristic called ‘First Fit”. We decide to use it for the following reasons:

- It is very fast because the order of vertices does not depend on the characteristics of each vertex (degree, number of colored neighbors etc.). We do need a fast heuristic because we are going to run our algorithm several times in order to set probabilities.
- This heuristic is not very performant. However, in this exercise we do not want to perfectly colorize our graph, we want to set probabilities. Moreover, we can think of this like a kind of Random Forest in Machine Learning. One output of a graph coloring algorithm with this heuristic gives bad results but the combination of 50,100 or 500 outputs of the same algorithm gives much better results.

This is our algorithm for the coloring function. It is simple, so we do not have many things to argue about.

Let us just talk about the initialization of the list of color. Initially we allowed three colors to colorize the all graph because this is the optimal number of colors according to Brook's theorem. But our algorithm is greedy, it is not always procuring the optimal solution. To avoid errors during runtime we allowed four colors to color the all graph.

Input: an adjacency list L and a list O which represent the order of the vertices
Initialization:
    A dictionary D which contains for each vertex its color.
    D is initialized empty.
    A list of color.
Body:
    While there is an uncolored vertex :
        Select the first vertex in O.
        Look all the colored vertex connected to the actual vertex and take their color.
        Select the first color in the list of color which is not used by the neighbors.
        Update D with the vertex and its color.
Output: D

This next algorithm is the one we use to set our probabilities. Again, it is simple, so we do not have many things to argue one.

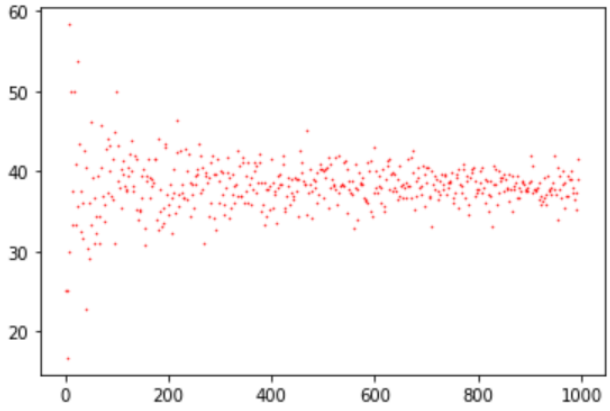
Input: an adjacency list
Initialization:
    Three empty list L1,L2 and L3.
Body:
    For j in range(0,n) :
        Randomly order the vertices and saves the order in a list O.
        Coloring function ( )
        Extend L1 with vertices which have the same color than the vertex 1 according to the previous function.
        Do the same for L2 with vertex 4 and L3 with vertex 5.

    For each list L1,L2 and L3 :
        Count the number of occurrences of each element.
        Divide this number by n and multiply by 100.
        Save the results in a dictionary.
Output: 3 dictionaries.

One of the challenges here is to find the correct n which represent the number of times this algorithm will be run.

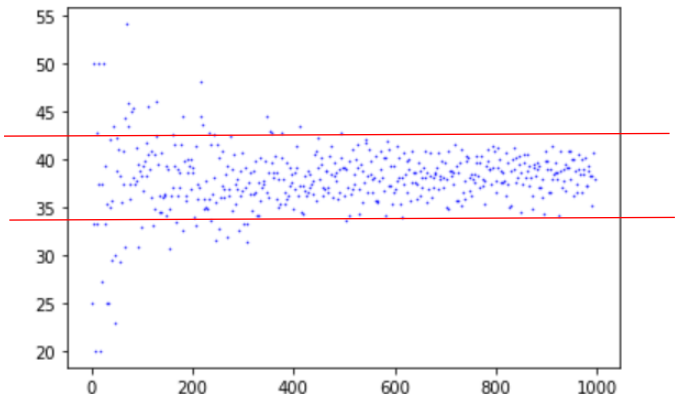
In order to find n, we slightly modify the previous algorithm by adding some code :

- A loop FOR in order to range n from 1 to 1000.
- Some list to save results after each iteration of the loop FOR that we added.



This graph represents the evolution for n=1 to n=1000 of the probability of vertex 3 being the impostor if vertex 1 is the first impostor.

It seems that the probability converges when n is superior to 400.



This graph represents the evolution for n=1 to n=1000 of the probability of vertex 9 being the impostor if vertex 1 is the first impostor.

It seems that the probability converges when n is superior to 400.

It appears in both cases that our precision is about 10 percent.

We decide to set n at 400.

4. Implement the algorithm and show a solution.

```
After 400 iterations :


These are the probabilities when the first importor is the player One :
{0: 0.0, 1: 100.0, 2: 0.0, 3: 36.75, 4: 37.75, 5: 37.0, 6: 0.0, 7: 37.5, 8: 40.25, 9: 40.5}


These are the probabilities when the first importor is the player Four :
{0: 0.0, 1: 37.75, 2: 40.5, 3: 0.0, 4: 100.0, 5: 39.0, 6: 36.5, 7: 35.5, 8: 40.25, 9: 0.0}


These are the probabilities when the first importor is the player Five :
{0: 0.0, 1: 37.0, 2: 36.75, 3: 38.5, 4: 39.0, 5: 100.0, 6: 40.0, 7: 0.0, 8: 0.0, 9: 36.75}
```

These are our results.

When the first impostor is player One, the most probable second impostor is either player Eight (40.25%) or player Nine (40.5%). Our precision is more or less than 10 percent, so we cannot conclude with accuracy on who is the second impostor. However, we can guarantee the innocence of player Two and player Six in this configuration.

Besides when the impostor is player Four, player Eight (40.25 %) and player Two (40.5 %) have the highest odds to be the second impostor. In this case we can guarantee the innocence of player Three and player Nine.

Moreover, if the first impostor is the player Five, player Six (40%) or player Four (39%) is the most likely second impostor. Here we can guarantee the innocence of player Seven and Eight.

Finally, player Eight appears twice as a good suspect. If we do not take account of our error, it might be our first suspect.

Step 3: I don't see him, but I can give proofs he vents!

1. Presents and argue about the two models of the map.

We decide to use a weighted graph in order to represent the map. Each room of the map is a vertex and two vertices are connected by a weighted edge. The edge represents the corridor and the weight represents the distance in centimeters (or the time in seconds) between the rooms. The graph is undirected because corridors do not have directions.

In this exercise we deal with two configurations, the player is either a crewmate or an impostor. That is why we have two weighted graphs, one for each case.

Then we decide to represent each graph by a distance matrix because it is the most convenient way to illustrate our graph for this exercise. Indeed, in this exercise, we need to find the distance between two rooms, so we need information about the vertices.

In order to improve the comprehension of our code and to make the outputs more esthetic, we decide to use a data frame from the library “pandas” rather than a matrix. The advantage of a data frame is that it is a kind of matrix with labeled rows and columns.

	Upper Engine	Reactor	Security	Lower Engine	Medbay	Electrical	Cafeteria	Storage	Weapons	O2	Navigation	Shields	Admin	Communication
Upper Engine	0.0	5.5	5.0	6.5	6.0	0.0	8.3	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Reactor	5.5	0.0	4.0	5.5	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Security	5.0	4.0	0.0	5.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Lower Engine	6.5	5.5	5.0	0.0	0.0	7.7	0.0	8.4	0.0	0.0	0.0	0.0	0.0	0.0
Medbay	6.0	0.0	0.0	0.0	0.0	0.0	6.4	0.0	0.0	0.0	0.0	0.0	0.0	0.0
Electrical	0.0	0.0	0.0	7.7	0.0	0.0	0.0	5.9	0.0	0.0	0.0	0.0	0.0	0.0
Cafeteria	8.3	0.0	0.0	0.0	6.4	0.0	0.0	7.0	4.5	0.0	0.0	0.0	6.3	0.0
Storage	0.0	0.0	0.0	8.4	0.0	5.9	7.0	0.0	0.0	0.0	0.0	5.7	5.7	5.3
Weapons	0.0	0.0	0.0	0.0	0.0	0.0	4.5	0.0	0.0	3.5	6.3	6.9	0.0	0.0
O2	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	3.5	0.0	5.9	7.1	0.0	0.0
Navigation	0.0	0.0	0.0	0.0	0.0	0.0	0.0	0.0	6.3	5.9	0.0	6.8	0.0	0.0
Shields	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.7	6.9	7.1	6.8	0.0	0.0	4.0
Admin	0.0	0.0	0.0	0.0	0.0	0.0	6.3	5.7	0.0	0.0	0.0	0.0	0.0	0.0
Communication	0.0	0.0	0.0	0.0	0.0	0.0	0.0	5.3	0.0	0.0	0.0	4.0	0.0	0.0

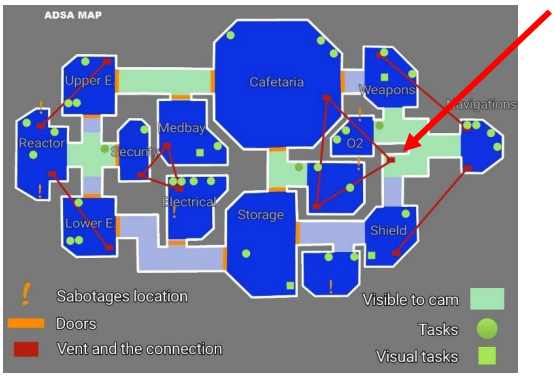
Data Frame of the distances if the player is a crewmate

This data frame is our first one. We can see that the time between the room “Security” and “Medbay” is 0. It makes sense because there is no corridor between these two rooms. The distance between “Upper Engine” and “Cafeteria” is 8.3 centimeters, so a player takes 8.3 seconds to travel from “Upper Engine” to “Cafeteria”.

	Upper Engine	Reactor	Security	Lower Engine	Medbay	Electrical	Cafeteria	Storage	Weapons	O2	Navigation	Shields	Admin	Communication
Upper Engine	0.000000	0.000001	5.000000	6.500000	6.000000	0.000000	8.300000	0.0	0.000000	0.0	0.000000	0.000000	0.000000	0.0
Reactor	0.000001	0.000000	4.000000	0.000001	0.000000	0.000000	0.000000	0.0	0.000000	0.0	0.000000	0.000000	0.000000	0.0
Security	5.000000	4.000000	0.000000	5.000000	0.000001	0.000001	0.000000	0.0	0.000000	0.0	0.000000	0.000000	0.000000	0.0
Lower Engine	6.500000	0.000001	5.000000	0.000000	0.000000	7.700000	0.000000	8.4	0.000000	0.0	0.000000	0.000000	0.000000	0.0
Medbay	6.000000	0.000000	0.000001	0.000000	0.000000	0.000001	6.400000	0.0	0.000000	0.0	0.000000	0.000000	0.000000	0.0
Electrical	0.000000	0.000000	0.000001	7.700000	0.000001	0.000000	0.000000	5.9	0.000000	0.0	0.000000	0.000000	0.000000	0.0
Cafeteria	8.300000	0.000000	0.000000	0.000000	6.400000	0.000000	0.000000	7.0	4.500000	4.4	4.000000	3.100000	0.000001	0.0
Storage	0.000000	0.000000	0.000000	8.400000	0.000000	5.900000	7.000000	0.0	0.000000	0.0	0.000000	5.700000	5.700000	5.3
Weapons	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	4.500000	0.0	0.000000	3.5	0.000001	6.900000	4.500000	0.0
O2	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	4.400000	0.0	3.500000	0.0	5.900000	7.100000	4.400000	0.0
Navigation	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	4.000000	0.0	0.000001	5.9	0.000000	0.000001	4.000000	0.0
Shields	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	3.100000	5.7	6.900000	7.1	0.000001	0.000000	3.100000	4.0
Admin	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000001	5.7	4.500000	4.4	4.000000	3.100000	0.000000	0.0
Communication	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	0.000000	5.3	0.000000	0.0	0.000000	4.000000	0.000000	0.0

Data Frame of distances if the player is an impostor

This is our data frame if the player is an impostor. An impostor can travel through a vent and it takes no time (0 second), so we upgrade the first data frame in order to add distance between rooms with vents. We decide to set the distance between rooms with vents at 0.001 rather than 0 in order to be able to difference where there is a vent and where there is no path between the rooms.



This vent is quite challenging because it is in a corridor. One possible solution is to add a new node in the graph in order to represent this vent. But in our graph, a vertex represent a room and we do not want to change that. So, we just calculate the distance like we do before by measuring.

This vent creates a path between rooms which were not connected before. In this case we simply upgrade the data frame.

This vent also creates a new path between rooms which were already connected. In this case we choose the smaller value of the distance between the path with the vent and the one without. We do that because we are looking for the smallest distance so there is no interest to use a vent if there is already a shorter path.

For example, the time between the room “Security” and the room “Medbay” was 0 in the last matrix. Otherwise in this one, the time is 0.001 seconds because there is a vent between these rooms.

2. Argue about a pathfinding algorithm to implement.

We use the “Floyd Warshall” algorithm which computes the distance between all pairs of vertices.

This is the algorithm we use. It takes one of the previous data frames as parameter.

```
Input : distance matrix "Map"
Initialization :
Create two matrices of zeros D and P
For each cell (i,j) in Map :
        If i != j and Map[i,j] == 0:
                D[i,j] = infinity
                P[i,j] = 'None'
        If (i,j) is an edge of Map:
                D[i,j]  = weight(i,j)
                P[i,j]  = name of the room i
        Else:
                P[i,j]  = 'None'
Body :
For k from 1 to the number of rooms:
        For i from 1 to the number of rooms:
                For j from 1 to number of rooms:
                        If D[i,j] > D[i,k] + D[k,j] :
                                D[i,j] = D[i,k] + D[k,j]
                                P[i,j] = name of the room k
Output : D and P
```

We create two more data frames, and we initialize both. The first one is a slightly different version of the data frame of the distances. In this version the distance between two unconnected rooms is initialized to 1000.

We do that because we are looking for the shortest path and it is not possible to have a path equal or higher than 1000 in this graph. The first time that the algorithm will find a path between two unconnected rooms, 1000 will be replaced by the actual distance of this path.

The second data frame illustrates the path from one room to another. The element which represents two unconnected rooms are initialized to ‘None’. If there is a path between two rooms, the element is initialized to the name of the room where we start the path.

Then for each room we calculate the distance between this room and another. If this value is inferior at the distance stored in the first data frame, we replaced the distance by the actual value.

3. Implement the method and show the time to travel for any pair of rooms for both models.

This is the results when the player is a crewmate.

	Upper Engine	Reactor	Security	Lower Engine	Medbay	Electrical	Cafeteria	Storage	Weapons	O2	Navigation	Shields	Admin	Communication
Upper Engine	0.0	5.5	5.0	6.5	6.0	14.2	8.3	14.9	12.8	16.3	19.1	19.7	14.6	20.2
Reactor	5.5	0.0	4.0	5.5	11.5	13.2	13.8	13.9	18.3	21.8	24.6	19.6	19.6	19.2
Security	5.0	4.0	0.0	5.0	11.0	12.7	13.3	13.4	17.8	21.3	24.1	19.1	19.1	18.7
Lower Engine	6.5	5.5	5.0	0.0	12.5	7.7	14.8	8.4	19.3	21.2	20.9	14.1	14.1	13.7
Medbay	6.0	11.5	11.0	12.5	0.0	19.3	6.4	13.4	10.9	14.4	17.2	17.8	12.7	18.7
Electrical	14.2	13.2	12.7	7.7	19.3	0.0	12.9	5.9	17.4	18.7	18.4	11.6	11.6	11.2
Cafeteria	8.3	13.8	13.3	14.8	6.4	12.9	0.0	7.0	4.5	8.0	10.8	11.4	6.3	12.3
Storage	14.9	13.9	13.4	8.4	13.4	5.9	7.0	0.0	11.5	12.8	12.5	5.7	5.7	5.3
Weapons	12.8	18.3	17.8	19.3	10.9	17.4	4.5	11.5	0.0	3.5	6.3	6.9	10.8	10.9
O2	16.3	21.8	21.3	21.2	14.4	18.7	8.0	12.8	3.5	0.0	5.9	7.1	14.3	11.1
Navigation	19.1	24.6	24.1	20.9	17.2	18.4	10.8	12.5	6.3	5.9	0.0	6.8	17.1	10.8
Shields	19.7	19.6	19.1	14.1	17.8	11.6	11.4	5.7	6.9	7.1	6.8	0.0	11.4	4.0
Admin	14.6	19.6	19.1	14.1	12.7	11.6	6.3	5.7	10.8	14.3	17.1	11.4	0.0	11.0
Communication	20.2	19.2	18.7	13.7	18.7	11.2	12.3	5.3	10.9	11.1	10.8	4.0	11.0	0.0

For example, the time to travel from the room “Upper Engine” to the room “Electrical” is 14.2 seconds.

We also know the path from one room to another:

	Upper Engine	Reactor	Security	Lower Engine	Medbay	Electrical	Cafeteria	Storage	Weapons	O2	Navigation	Shields	Admin	Communication
Upper Engine	None	Upper Engine	Upper Engine	Upper Engine	Upper Engine	Lower Engine	Upper Engine	Lower Engine	Cafeteria	Weapons	Weapons	Weapons	Cafeteria	Storage
Reactor	Reactor	None	Reactor	Reactor	Upper Engine	Lower Engine	Upper Engine	Lower Engine	Cafeteria	Weapons	Weapons	Storage	Storage	Storage
Security	Security	Security	None	Security	Upper Engine	Lower Engine	Upper Engine	Lower Engine	Cafeteria	Weapons	Weapons	Storage	Storage	Storage
Lower Engine	Lower Engine	Lower Engine	Lower Engine	None	Upper Engine	Lower Engine	Upper Engine	Lower Engine	Cafeteria	Shields	Shields	Storage	Storage	Storage
Medbay	Medbay	Upper Engine	Upper Engine	Upper Engine	None	Storage	Medbay	Cafeteria	Cafeteria	Weapons	Weapons	Weapons	Cafeteria	Storage
Electrical	Lower Engine	Lower Engine	Lower Engine	Electrical	Storage	None	Storage	Electrical	Storage	Shields	Shields	Storage	Storage	Storage
Cafeteria	Cafeteria	Upper Engine	Upper Engine	Upper Engine	Cafeteria	Storage	None	Cafeteria	Cafeteria	Weapons	Weapons	Weapons	Cafeteria	Storage
Storage	Lower Engine	Lower Engine	Lower Engine	Storage	Cafeteria	Storage	Storage	None	Cafeteria	Shields	Shields	Storage	Storage	Storage
Weapons	Cafeteria	Cafeteria	Cafeteria	Cafeteria	Cafeteria	Storage	Weapons	Cafeteria	None	Weapons	Weapons	Weapons	Cafeteria	Shields
O2	Weapons	Weapons	Weapons	Shields	Weapons	Shields	Weapons	Shields	O2	None	O2	O2	Weapons	Shields
Navigation	Weapons	Weapons	Weapons	Shields	Weapons	Shields	Weapons	Shields	Navigation	Navigation	None	Navigation	Weapons	Shields
Shields	Weapons	Storage	Storage	Storage	Weapons	Storage	Weapons	Shields	Shields	Shields	Shields	None	Storage	Shields
Admin	Cafeteria	Storage	Storage	Storage	Cafeteria	Storage	Admin	Admin	Cafeteria	Weapons	Weapons	Storage	None	Storage
Communication	Storage	Storage	Storage	Storage	Storage	Storage	Storage	Communication	Shields	Shields	Shields	Communication	Storage	None

For example, the path between the room “Upper Engine” and the room “Electrical” is :

“Upper Engine” ⇨ “Lower Engine” ⇨ “Electrical”

And the path between the room “Security” and the room “Weapons” is :

“Security” ⇨ “Upper Engine” ⇨ “Cafeteria” ⇨ “Weapon”

The next results are the results when the player is an impostor.

	Upper Engine	Reactor	Security	Lower Engine	Medbay	Electrical	Cafeteria	Storage	Weapons	O2	Navigation	Shields	Admin	Communication
Upper Engine	0.000	0.001	4.001	0.002	4.002	4.002	8.300	8.402	11.402	12.700	11.401	11.400	8.301	13.702
Reactor	0.001	0.000	4.000	0.001	4.001	4.001	8.301	8.401	11.403	12.701	11.402	11.401	8.302	13.701
Security	4.001	4.000	0.000	4.001	0.001	0.001	6.401	5.901	9.503	10.801	9.502	9.501	6.402	11.201
Lower Engine	0.002	0.001	4.001	0.000	4.002	4.002	8.302	8.400	11.404	12.702	11.403	11.402	8.303	13.700
Medbay	4.002	4.001	0.001	4.002	0.000	0.001	6.400	5.901	9.502	10.800	9.501	9.500	6.401	11.201
Electrical	4.002	4.001	0.001	4.002	0.001	0.000	6.401	5.900	9.503	10.801	9.502	9.501	6.402	11.200
Cafeteria	8.300	8.301	6.401	8.302	6.400	6.401	0.000	5.701	3.102	4.400	3.101	3.100	0.001	7.100
Storage	8.402	8.401	5.901	8.400	5.901	5.900	5.701	0.000	5.702	9.202	5.701	5.700	5.700	5.300
Weapons	11.402	11.403	9.503	11.404	9.502	9.503	3.102	5.702	0.000	3.500	0.001	0.002	3.102	4.002
O2	12.700	12.701	10.801	12.702	10.800	10.801	4.400	9.202	3.500	0.000	3.501	3.502	4.400	7.502
Navigation	11.401	11.402	9.502	11.403	9.501	9.502	3.101	5.701	0.001	3.501	0.000	0.001	3.101	4.001
Shields	11.400	11.401	9.501	11.402	9.500	9.501	3.100	5.700	0.002	3.502	0.001	0.000	3.100	4.000
Admin	8.301	8.302	6.402	8.303	6.401	6.402	0.001	5.700	3.102	4.400	3.101	3.100	0.000	7.100
Communication	13.702	13.701	11.201	13.700	11.201	11.200	7.100	5.300	4.002	7.502	4.001	4.000	7.100	0.000

In this case, the time to travel from the room “Upper Engine” to the room “Electrical” is 4.002 seconds. An impostor saves 10 seconds. The travel is faster because the impostor used two vents, we deduce that thanks to the decimals in 0.002.

	Upper Engine	Reactor	Security	Lower Engine	Medbay	Electrical	Cafeteria	Storage	Weapons	O2	Navigation	Shields	Admin	Communication
Upper Engine	None	Upper Engine	Reactor	Reactor	Security	Security	Upper Engine	Lower Engine	Shields	Cafeteria	Shields	Cafeteria	Cafeteria	Storage
Reactor	Reactor	None	Reactor	Reactor	Security	Security	Upper Engine	Lower Engine	Shields	Cafeteria	Shields	Cafeteria	Cafeteria	Storage
Security	Reactor	Security	None	Reactor	Security	Security	Medbay	Electrical	Shields	Cafeteria	Shields	Cafeteria	Cafeteria	Storage
Lower Engine	Reactor	Lower Engine	Reactor	None	Security	Security	Reactor	Lower Engine	Shields	Cafeteria	Shields	Cafeteria	Cafeteria	Storage
Medbay	Security	Security	Medbay	Security	None	Medbay	Medbay	Electrical	Shields	Cafeteria	Shields	Cafeteria	Cafeteria	Storage
Electrical	Security	Security	Electrical	Security	Electrical	None	Medbay	Electrical	Shields	Cafeteria	Shields	Cafeteria	Cafeteria	Storage
Cafeteria	Cafeteria	Upper Engine	Medbay	Reactor	Cafeteria	Medbay	None	Admin	Shields	Cafeteria	Shields	Cafeteria	Cafeteria	Shields
Storage	Lower Engine	Lower Engine	Electrical	Storage	Electrical	Storage	Admin	None	Shields	Shields	Shields	Storage	Storage	Storage
Weapons	Shields	Shields	Shields	Shields	Shields	Shields	Shields	Shields	None	Weapons	Weapons	Navigation	Shields	Shields
O2	Cafeteria	Cafeteria	Cafeteria	Cafeteria	Cafeteria	Cafeteria	O2	Shields	O2	None	Weapons	Navigation	O2	Shields
Navigation	Shields	Shields	Shields	Shields	Shields	Shields	Shields	Shields	Navigation	Weapons	None	Navigation	Shields	Shields
Shields	Cafeteria	Cafeteria	Cafeteria	Cafeteria	Cafeteria	Cafeteria	Shields	Shields	Navigation	Navigation	Shields	None	Shields	Shields
Admin	Cafeteria	Cafeteria	Cafeteria	Cafeteria	Cafeteria	Cafeteria	Admin	Admin	Shields	Admin	Shields	Admin	None	Shields
Communication	Storage	Storage	Storage	Storage	Storage	Storage	Shields	Communication	Shields	Shields	Shields	Communication	Shields	None

Let’s see the from the room “Upper Engine” to the room “Electrical” in this case:

“Upper Engine” ⇨ “Reactor” ⇨ “Security” ⇨ “Electrical”

This time the path goes across 4 rooms. It is one more room than the previous path but thanks to the vents, this path is the shortest.

4. Display the interval of time for each pair of room where the traveler is an impostor.

	Upper Engine	Reactor	Security	Lower Engine	Medbay	Electrical	Cafeteria	Storage	Weapons	O2	Navigation	Shields	Admin	Communication
Upper Engine	0.000	5.499	0.999	6.498	1.998	10.198	0.000	6.498	1.398	3.600	7.699	8.300	6.299	6.498
Reactor	5.499	0.000	0.000	5.499	7.499	9.199	5.499	5.499	6.897	9.099	13.198	8.199	11.298	5.499
Security	0.999	0.000	0.000	0.999	10.999	12.699	6.899	7.499	8.297	10.499	14.598	9.599	12.698	7.499
Lower Engine	6.498	5.499	0.999	0.000	8.498	3.698	6.498	0.000	7.896	8.498	9.497	2.698	5.797	0.000
Medbay	1.998	7.499	10.999	8.498	0.000	19.299	0.000	7.499	1.398	3.600	7.699	8.300	6.299	7.499
Electrical	10.198	9.199	12.699	3.698	19.299	0.000	6.499	0.000	7.897	7.899	8.898	2.099	5.198	0.000
Cafeteria	0.000	5.499	6.899	6.498	0.000	6.499	0.000	1.299	1.398	3.600	7.699	8.300	6.299	5.200
Storage	6.498	5.499	7.499	0.000	7.499	0.000	1.299	0.000	5.798	3.598	6.799	0.000	0.000	0.000
Weapons	1.398	6.897	8.297	7.896	1.398	7.897	1.398	5.798	0.000	0.000	6.299	6.898	7.698	6.898
O2	3.600	9.099	10.499	8.498	3.600	7.899	3.600	3.598	0.000	0.000	2.399	3.598	9.900	3.598
Navigation	7.699	13.198	14.598	9.497	7.699	8.898	7.699	6.799	6.299	2.399	0.000	6.799	13.999	6.799
Shields	8.300	8.199	9.599	2.698	8.300	2.099	8.300	0.000	6.898	3.598	6.799	0.000	8.300	0.000
Admin	6.299	11.298	12.698	5.797	6.299	5.198	6.299	0.000	7.698	9.900	13.999	8.300	0.000	3.900
Communication	6.498	5.499	7.499	0.000	7.499	0.000	5.200	0.000	6.898	3.598	6.799	0.000	3.900	0.000

As we saw before, an impostor saves environ 10 seconds when he travels from the “Upper Engine” to the room “Electrical”.

It seems that paths with vents cannot be longer than paths without. An impostor is never disadvantaged. Indeed, there is no negative value in this data frame. However, an impostor is not always advantaged because for some paths there is no improvement. For example, the path between “Communication” and “Lower Engine”.

Step 4: Secure the last tasks :

1. Presents and argue about the model of the map.

We decided to use the data frame that we already used for part 3. However, in this part we don’t need to take care of the vents. Indeed, if an impostor uses a vent, he will be unmasked.

2. Thanks to a graph theory problem, present how to find a route passing through each room only one time.

Our graph is undirected, and we want to visit each vertex, starting from one room and ending at the same. What we need to find is a Hamiltonian circuit. The answer to this question is the answer of the Hamiltonian problem.

3. Argue about an algorithm solving your problem.

The brute force algorithm is not an option here, indeed there are more than three billion possible paths. We need to use a heuristic.

We decided to use the nearest neighbor algorithm to solve this problem because we want to find the shortest Hamiltonian circuit. This is a greedy algorithm which means that the algorithm takes what looks best in the short run, whether or not it is best in the long run. However, in our situation this algorithm fits fine because it will always be shorter to take one corridor rather than two.

Input : A distance matrix "Map", the room from where we started "R"
Initialization :
    Create a list L.
    Set the first element of the list at R.
    Create a variable which represent the actual room. Set this variable at R.
Body :
    While L doesn't contain once all the rooms and twice the room from where we started :
        Look at the neighbors of the actual room.
        If there are no neighbors :
            Exit the while loop.
        Look at the closest neighbors.
        While the closest neighbor is the room from where we started :
            Consider the next closest neighbor.
        Delete all path between the closest neighbor and all the others room.
        Add the closest neighbor to L.
        Set the actual room at the closest neighbor.
Output : The list L.

We apply this algorithm at each room of algorithm because the initial position influences the output.

We start with a list which contains only the initial room. This list represents the Hamiltonian circuit in our graph.

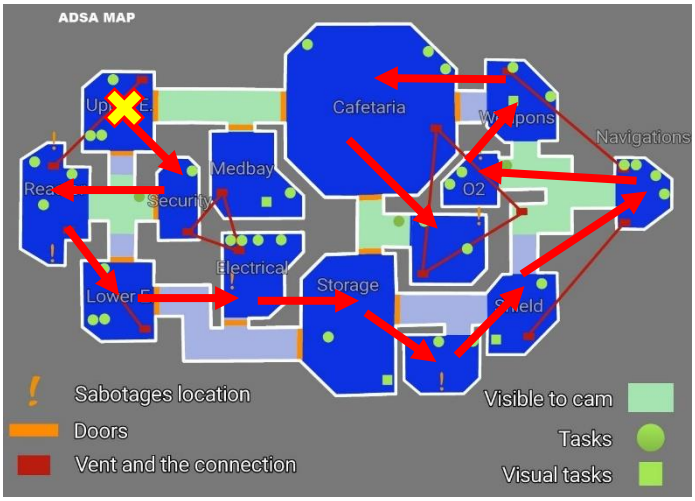
Then we check our data frame at the line which represents the actual room. We isolate this line and we sort it. After that we delete all the element equal to 0 (we only keep the room connected to the actual room). At this time, we have a sorted series which contains only the room connected to the actual room.

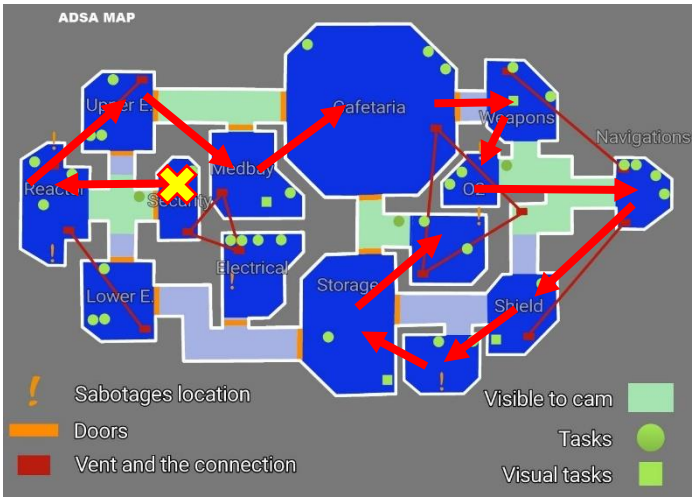
In this series we select the first element which are not the initial room. This element is the next room in our circuit. In our data frame we set to 0 all the distance between this element and the other rooms because a room can be visited twice. Finally, we add this element to the list which represents the circuit and we move to this element.

4. Implement the algorithm and show a solution.

We compute our algorithm starting from each room. Apparently, there is no Hamiltonian circuit in our graph.

This is some of the solutions we found out.

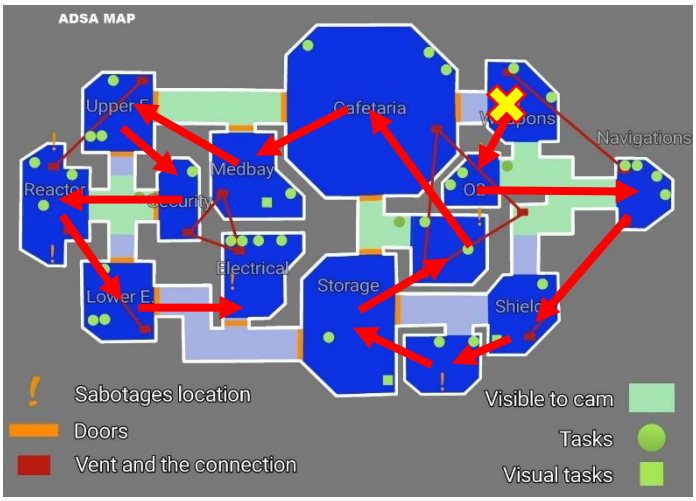




Find a Hamiltonian circuit starting from "Upper Engine"

Find a Hamiltonian circuit starting from "Security"

It seems that the room "Administrator" is problematic because the algorithm often ends at this room.



Find a Hamiltonian circuit starting from "Weapons"

There is no Hamiltonian circuit, but it seems that we can find Hamiltonian paths. We slightly modified our algorithm in order to find them. We find two Hamiltonian paths in our graph, the first one starts in the room “Weapons” and ends in the room “Electrical”. This path takes 72.1 seconds to be covered. This is the one illustrated previously.

The second one starts in the room “Medbay” and ends in the room “Admin”. This path takes 70.4 seconds to be covered.

The last one starts in the room “Admin” and ends in the room “Electrical”. This path takes 70.3 seconds to be covered. It is the shortest Hamiltonian path according to our algorithm.

Input: an adjacency list
Initialization:
    Three empty list L1,L2 and L3.
Body:
    For j in range(0,n) :
        Randomly order the vertices and saves the order in a list O.
        Coloring function ( )
        Extend L1 with vertices which have the same color than the vertex 1 according to the previous function.
        Do the same for L2 with vertex 4 and L3 with vertex 5.

    For each list L1,L2 and L3 :
        Count the number of occurrences of each element.
        Divide this number by n and multiply by 100.
        Save the results in a dictionary.
Output: 3 dictionaries.