Report ADSA

STEP 1

Question 1 : “Present a data structure for the player and its score.”

We use dictionary composed of the name of the player, its score (the mean of the score of all game), and a list of the score of each game.
A dictionary is a cleaner way (compared to a list) to represent a player since the name of the parameters are defined (playername, score and games), and allows us to reach easily any of the parameters when we will construct the class Player to construct the AVL tree.
To be clear we could have used a list and just use indexes to find what we needed, but a dictionary is more comfortable and easy to read for you.

Each player is stored in a list called players before putting them in the AVL Tree. This isn’t important, and it is done to separate question 1 and question 2 in the code. We could have added them in the tree directly, for example.

Question 2 : “Propose the most optimized data structure for the tournament.”

The most optimized data structure for the tournament is an AVL tree, as we saw it during class.
Due to the balancing property, the insertion, deletion and search operations which take O(log n).
The time complexity of adding every player in the database is O(n*log(n)), since there is n insertion of players in the AVL Tree.

Question 3/4 : “Present and argue about a method which randomize players score. Present and argue about a method to update players score and the database.”

The randomizing mechanism is very simple, we always randomize a value between 0 and 12, add it to the list of games of the player, and we update its score (which is the mean of all the games he played).

We have done three things for the updating part :
- First as asked in the instruction, two distinct functions, one to randomize players score and one to update them in the database.
- Second thing, a function which takes every players in a list (from an AVL Tree) (O(n)). Then we have a loop for each player, we randomize its score, and then insert the player in a new AVL Tree (so finally O(n*log(n))).
- Finally, a function which browse every node (O(n)) in a recursive way, randomize the score and insert the node to a new AVL Tree. We then call the function on the root.left or root.right. (so finally O(n*log(n)) without a list between).

The first and the second method can be split into 2 functions. The third one is the best and most optimized one since there are no lists and it is n*log(n). It combines question 3 and 4.

In the code you can find “updateScoresRandomly()” which corresponds to the first method, randomizeScore() which corresponds to the second one and updateScoresRandomlyAndInsertInNewTree() which correspond to the third one.

Question 5 : “Present and argue about a method to create random games based on the database.”

We create a list of list of the number of games that will be played (ex : for 90 players => 9 games, so a list of 9 lists).
We create a list of indices going from 0 to 99, which will be used to select the players only once.
We select a random “number” - which is in the range of the length of the list “indices” - and take the “value” of the player in the list "indices" at the index “number”, and then we will delete the “value” from the list “indices” so that we won't choose this player again (in fact we pop the value so that it gives us the value and delete it at the same time).

We put random and different 10 players in a list and add this list to the list containing all the games and we do it until there are no players left.
This way we are sure there is no 2 players who are the same in all the games.

Question 6 : “Present and argue about a method to create games based on rankings.”

We first create a list of list of the number of games that will be played (ex : for 90 players => 9 games, so a list of 9 lists)
Thanks to an in order traversal, we create a list of players in their ascending order from the AVL Tree directly.
So we just need to navigate in the sorted list containing each player, and add them into games of 10

Question 7 : “Present and argue about a method which drop the players until the last 10 players.”

We call the function tournamentPhase1 which simulate the 12 first games :
There is our 100 players. We play the 3 first games (random games) with no eliminations. At the end of it we drop the 10 worst players.
To do so we just have to use the function which creates ranked games from in-order traversal method and take all the games except the first one.
Since we then use this list of games to launch the next game in the loop, it is interesting not to create a function which would use a certain AVL Tree traversal in order to take all the players except the 10 last or anything like that. And since no instruction has been given concerning the structure of the games, this method seems to be the best one.
Then we will play games and drop the 10 worst players until there are only 10 players left.
We then print those players to show their scores and games.

Question 8 : “Present and argue about a method which display the last 10 players and the podium.”

Nothing particular, we just print the playername of the last 10 players with their scores and games (which are the 5 games they played after the reset of their scores and games).

STEP 2

Question 1 : “Argue about the graph representation of your model.”

We think using a class “Player_graph” is the best way to have a scalable use of our model. It means that if we want to use a player in other steps of the project, it will be easy and clean to implement.
So the Player_Graph class has 2 attributes, which are the name of the player, and a list of player from “Player_Graph”, which is the list of players that player has seen.

In this class we have a method which allows us to add a player in the list of players a player has seen.

And finally we have a display method which print the name of the player and the name of the players he saw.

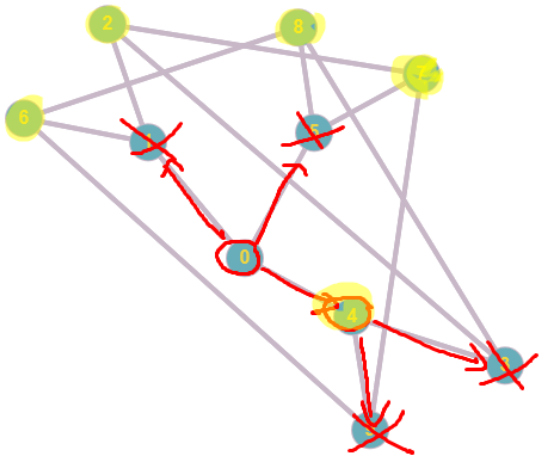
All of this constitutes a graph, where a node is a player, and the list of players he has seen are the vertices.
This graph is ultimately an undirected graph. Indeed, the relation “has seen” introduces the idea that if a player has seen another player, the other one has seen him too.

Question 2-3 : “Present how to find a set of probable impostors thanks to a graph theory problem and argue about an algorithm solving your problem.”

We found two ways using graph theory to find a set of probable impostors.

V1 method :

We can do it by elimination using a list containing the number corresponding to the players.
At the beginning the list is composed of every numbers.
We also create a list which will contain the set of impostors (this is optional since we could just display them directly)
We go to each node connected to the dead player.
While on a node, we consider it as an impostor, so the two others cannot be an impostor (so we delete them from the list), now we go to the nodes connected to our alleged impostor, and all the players that he has seen cannot be impostors too, so we delete them from the list too.
So the other nodes can be impostors, so we pair each of them with the initial node we were looking at and add them to the set of probable impostors.
And then we reset the list of numbers corresponding to the players and look at the next node connected to the dead player and repeat the process until we have looked up at each node (player) the dead player has seen.



Here is an example for the first node, where 4 is paired with the other yellowed nodes.
Then we repeat it for node 5 and 1

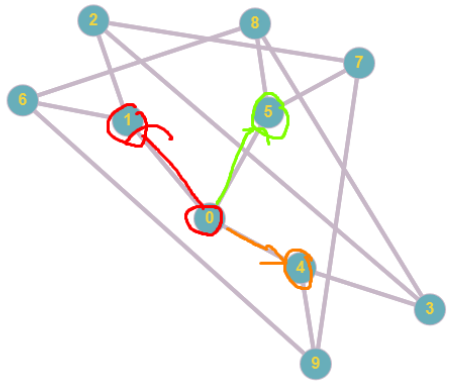
We found this method fine but it uses a list of numbers and we thought it might not be a 100% graph method.
So we propose another method which is close to a graph coloring method.

V2 method :

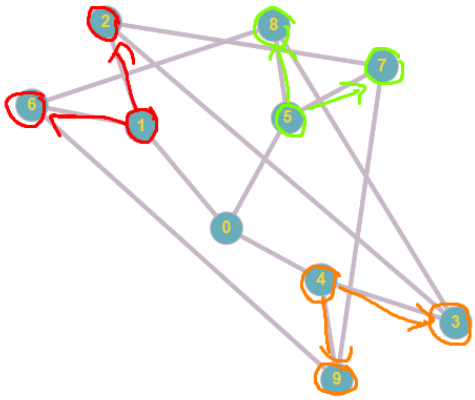
In this method we implement an algorithm which is close to a graph coloring algorithm. It is also more optimized since we don’t color the nodes and don’t comeback and look which color they are.

The idea, using the example of the problem, is (**Question 2**) :
First, since the first subjects are the Players who have seen Player 0 we can just take their corresponding node in the graph and go to any other node that is connected to it (except node 0 of course).
From these nodes we can now go one more step to all connected nodes and they will be our second suspect.
In this case it also refers to a graph coloring process.

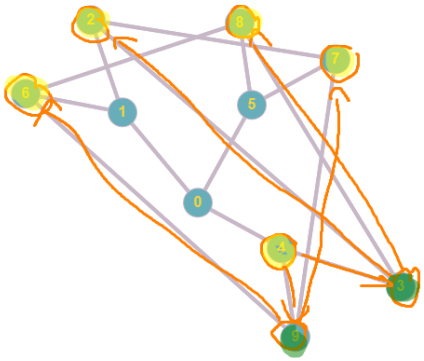
Now the algorithm (**Question 3**) :
We create a list which will contain the set of impostors (this is optional since we could just display them directly)
We look at each node the victim has seen.



For each of this node, we look at the player it has seen, and don't consider the victim of course.



Then we look at the players each of this node saw (without considering the node we come from)
Finally we can add the couple of probable impostor.



For example for the node 4
(we explicitly see how it is a graph-coloring-like process).
Where 4 is paired with each of the yellow colored nodes.

We think this is a better solution, and the best we found. It also refers to a coloring graph which is a graph theory problem we saw in class.

STEP 3

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

We model the crewmate and imposters maps by 2 weighted graphs. Each room is a node of the graph and the weights of edges are the distance between the rooms.

The results of these medialisations are:

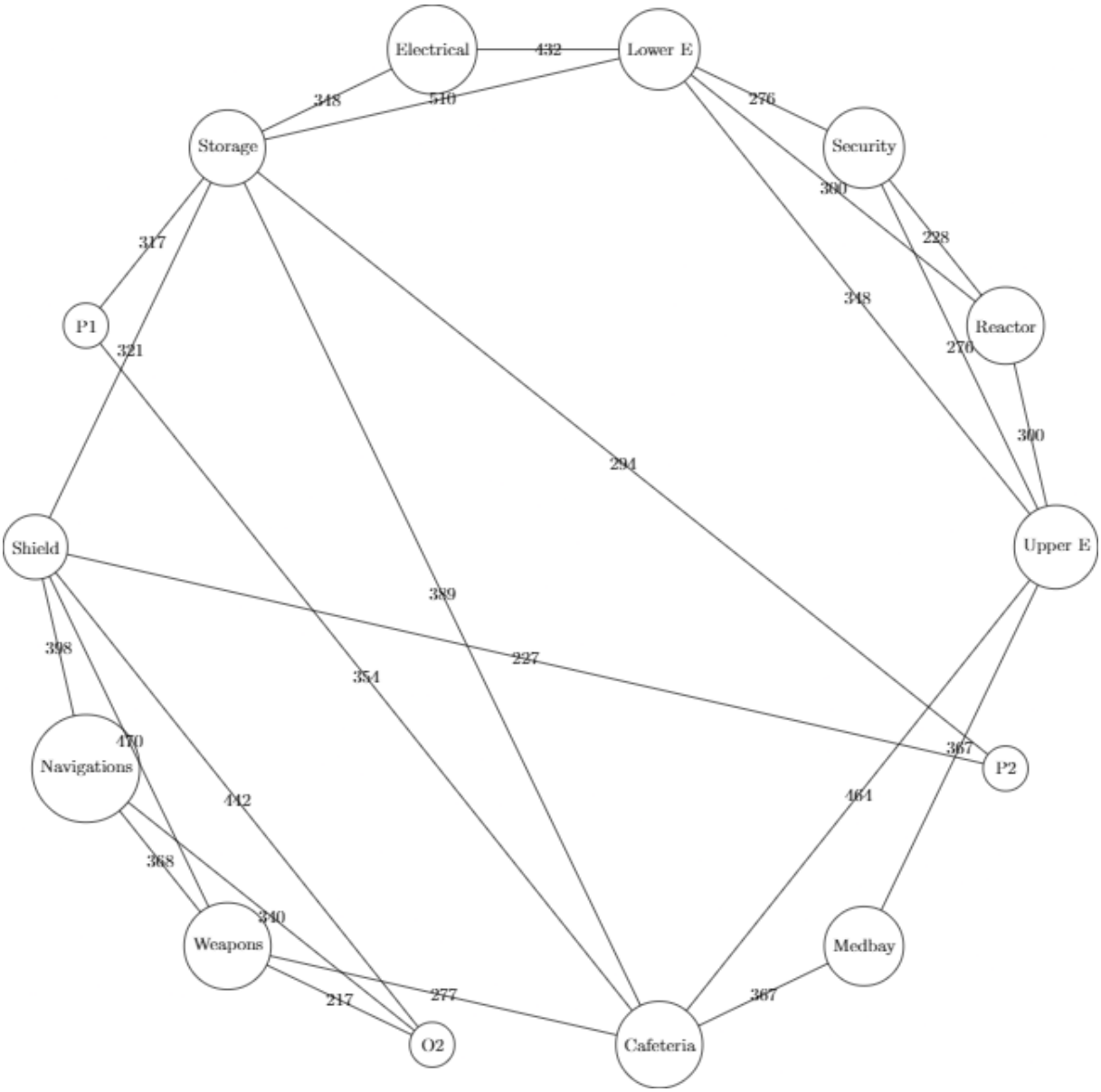


Figure 1 - Crewmate graph

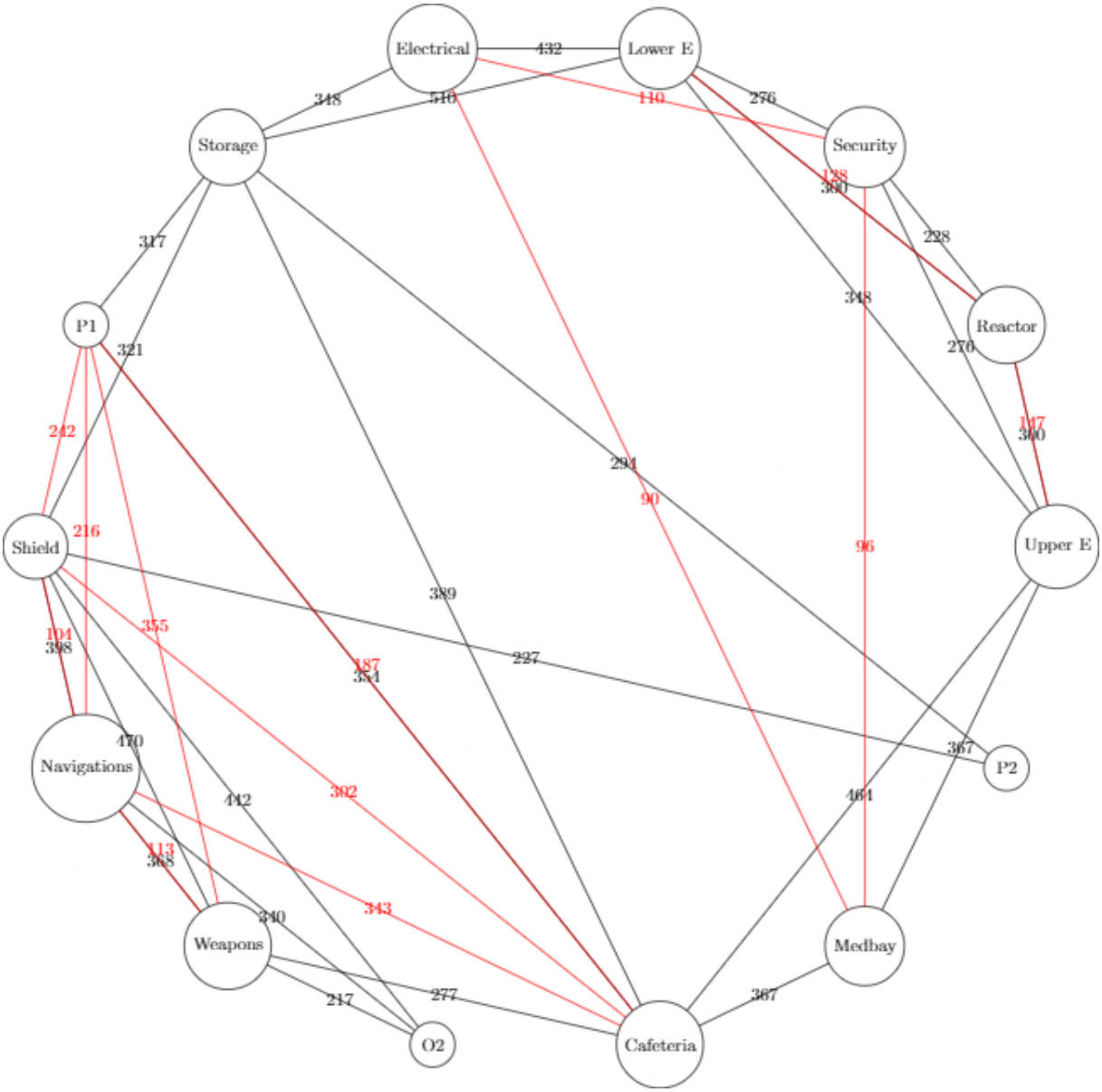


Figure 2 - Imposters graph

We can see that imposters get an advantage from the crewmates. They can go directly from a room to another using edge that crewmate cannot access. Plus, they have sometimes quicker edges.

Question 2: Argue about a pathfinding algorithm to implement.

We decided to implement the Floyd Warshall algorithm to get the time to travel to any pair of rooms for both models.

Question 3.

	UpperE	Reactor	Security	LowerE	Electrical	Storage	P1	Shield	Navigations	Weapons	O2	Cafeteria	Medbay	P2
UpperE	0.0	300.0	276.0	348.0	780.0	853.0	818.0	1174.0	1109.0	741.0	958.0	464.0	367.0	1147.0
Reactor	300.0	0.0	228.0	300.0	732.0	810.0	1118.0	1131.0	1409.0	1041.0	1258.0	764.0	667.0	1104.0
Security	276.0	228.0	0.0	276.0	708.0	786.0	1094.0	1107.0	1385.0	1017.0	1234.0	740.0	643.0	1080.0
LowerE	348.0	300.0	276.0	0.0	432.0	510.0	827.0	831.0	1229.0	1089.0	1273.0	812.0	715.0	804.0
Electrical	780.0	732.0	708.0	432.0	0.0	348.0	665.0	669.0	1067.0	1014.0	1111.0	737.0	1104.0	642.0
Storage	853.0	810.0	786.0	510.0	348.0	0.0	317.0	321.0	719.0	666.0	763.0	389.0	756.0	294.0
P1	818.0	1118.0	1094.0	827.0	665.0	317.0	0.0	638.0	999.0	631.0	848.0	354.0	721.0	611.0
Shield	1174.0	1131.0	1107.0	831.0	669.0	321.0	638.0	0.0	398.0	470.0	442.0	710.0	1077.0	227.0
Navigations	1109.0	1409.0	1385.0	1229.0	1067.0	719.0	999.0	398.0	0.0	368.0	340.0	645.0	1012.0	625.0
Weapons	741.0	1041.0	1017.0	1089.0	1014.0	666.0	631.0	470.0	368.0	0.0	217.0	277.0	644.0	697.0
O2	958.0	1258.0	1234.0	1273.0	1111.0	763.0	848.0	442.0	340.0	217.0	0.0	494.0	861.0	669.0
Cafeteria	464.0	764.0	740.0	812.0	737.0	389.0	354.0	710.0	645.0	277.0	494.0	0.0	367.0	683.0
Medbay	367.0	667.0	643.0	715.0	1104.0	756.0	721.0	1077.0	1012.0	644.0	861.0	367.0	0.0	1050.0
P2	1147.0	1104.0	1080.0	804.0	642.0	294.0	611.0	227.0	625.0	697.0	669.0	683.0	1050.0	0.0

Figure 3- Crewmate time travel

	UpperE	Reactor	Security	LowerE	Electrical	Storage	P1	Shield	Navigations	Weapons	O2	Cafeteria	Medbay	P2
UpperE	0.0	147.0	276.0	275.0	386.0	734.0	651.0	766.0	807.0	741.0	958.0	464.0	367.0	993.0
Reactor	147.0	0.0	228.0	128.0	338.0	638.0	798.0	913.0	954.0	888.0	1105.0	611.0	324.0	932.0
Security	276.0	228.0	0.0	276.0	110.0	458.0	650.0	765.0	806.0	740.0	957.0	463.0	96.0	752.0
LowerE	275.0	128.0	276.0	0.0	386.0	510.0	827.0	831.0	935.0	1016.0	1233.0	739.0	372.0	804.0
Electrical	386.0	338.0	110.0	386.0	0.0	348.0	644.0	669.0	773.0	734.0	951.0	457.0	90.0	642.0
Storage	734.0	638.0	458.0	510.0	348.0	0.0	317.0	321.0	425.0	538.0	755.0	389.0	438.0	294.0
P1	651.0	798.0	650.0	827.0	644.0	317.0	0.0	242.0	216.0	329.0	546.0	187.0	554.0	469.0
Shield	766.0	913.0	765.0	831.0	669.0	321.0	242.0	0.0	104.0	217.0	434.0	302.0	669.0	227.0
Navigations	807.0	954.0	806.0	935.0	773.0	425.0	216.0	104.0	0.0	113.0	330.0	343.0	710.0	331.0
Weapons	741.0	888.0	740.0	1016.0	734.0	538.0	329.0	217.0	113.0	0.0	217.0	277.0	644.0	444.0
O2	958.0	1105.0	957.0	1233.0	951.0	755.0	546.0	434.0	330.0	217.0	0.0	494.0	861.0	661.0
Cafeteria	464.0	611.0	463.0	739.0	457.0	389.0	187.0	302.0	343.0	277.0	494.0	0.0	367.0	529.0
Medbay	367.0	324.0	96.0	372.0	90.0	438.0	554.0	669.0	710.0	644.0	861.0	367.0	0.0	732.0
P2	993.0	932.0	752.0	804.0	642.0	294.0	469.0	227.0	331.0	444.0	661.0	529.0	732.0	0.0

Figure 4- Imposter time travel

	UpperE	Reactor	Security	LowerE	Electrical	Storage	P1	Shield	Navigations	Weapons	O2	Cafeteria	Medbay	P2
UpperE	nan	51.0 %	0.0 %	21.0 %	51.0 %	14.0 %	20.0 %	35.0 %	27.0 %	0.0 %	0.0 %	0.0 %	0.0 %	13.0 %
Reactor	51.0 %	nan	0.0 %	57.0 %	54.0 %	21.0 %	29.0 %	19.0 %	32.0 %	15.0 %	12.0 %	20.0 %	51.0 %	16.0 %
Security	0.0 %	0.0 %	nan	0.0 %	84.0 %	42.0 %	41.0 %	31.0 %	42.0 %	27.0 %	22.0 %	37.0 %	85.0 %	30.0 %
LowerE	21.0 %	57.0 %	0.0 %	nan	11.0 %	0.0 %	0.0 %	0.0 %	24.0 %	7.0 %	3.0 %	9.0 %	48.0 %	0.0 %
Electrical	51.0 %	54.0 %	84.0 %	11.0 %	nan	0.0 %	3.0 %	0.0 %	28.0 %	28.0 %	14.0 %	38.0 %	92.0 %	0.0 %
Storage	14.0 %	21.0 %	42.0 %	0.0 %	0.0 %	nan	0.0 %	0.0 %	41.0 %	19.0 %	1.0 %	0.0 %	42.0 %	0.0 %
P1	20.0 %	29.0 %	41.0 %	0.0 %	3.0 %	0.0 %	nan	62.0 %	78.0 %	48.0 %	36.0 %	47.0 %	23.0 %	23.0 %
Shield	35.0 %	19.0 %	31.0 %	0.0 %	0.0 %	0.0 %	62.0 %	nan	74.0 %	54.0 %	2.0 %	57.0 %	38.0 %	0.0 %
Navigations	27.0 %	32.0 %	42.0 %	24.0 %	28.0 %	41.0 %	78.0 %	74.0 %	nan	69.0 %	3.0 %	47.0 %	30.0 %	47.0 %
Weapons	0.0 %	15.0 %	27.0 %	7.0 %	28.0 %	19.0 %	48.0 %	54.0 %	69.0 %	nan	0.0 %	0.0 %	0.0 %	36.0 %
O2	0.0 %	12.0 %	22.0 %	3.0 %	14.0 %	1.0 %	36.0 %	2.0 %	3.0 %	0.0 %	nan	0.0 %	0.0 %	1.0 %
Cafeteria	0.0 %	20.0 %	37.0 %	9.0 %	38.0 %	0.0 %	47.0 %	57.0 %	47.0 %	0.0 %	0.0 %	nan	0.0 %	23.0 %
Medbay	0.0 %	51.0 %	85.0 %	48.0 %	92.0 %	42.0 %	23.0 %	38.0 %	30.0 %	0.0 %	0.0 %	0.0 %	nan	30.0 %
P2	13.0 %	16.0 %	30.0 %	0.0 %	0.0 %	0.0 %	23.0 %	0.0 %	47.0 %	36.0 %	1.0 %	23.0 %	30.0 %	nan

Figure 5 - Celerity of imposters compare to cewmate

We can see for example that the imposters can go from the Medbay to Security 85% quicker that the crewmate.

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

	UpperE	Reactor	Security	LowerE	Electrical	Storage	P1	Shield	Navigations	Weapons	O2	Cafeteria	Medbay	P2
UpperE	[0.0, 0.0[	[147.0, 300.0[	[276.0, 276.0[	[275.0, 348.0[	[386.0, 780.0[	[734.0, 853.0[	[651.0, 818.0[	[766.0, 1174.0[	[807.0, 1109.0[	[741.0, 741.0[	[958.0, 958.0[	[464.0, 464.0[	[367.0, 367.0[	[993.0, 1147.0[
Reactor	[147.0, 300.0[	[0.0, 0.0[	[228.0, 228.0[	[128.0, 300.0[	[338.0, 732.0[	[638.0, 810.0[	[798.0, 1118.0[	[913.0, 1131.0[	[954.0, 1409.0[	[888.0, 1041.0[	[1105.0, 1258.0[	[611.0, 764.0[	[324.0, 667.0[	[932.0, 1104.0[
Security	[276.0, 276.0[	[228.0, 228.0[	[0.0, 0.0[	[276.0, 276.0[	[110.0, 708.0[	[458.0, 786.0[	[650.0, 1094.0[	[765.0, 1107.0[	[806.0, 1385.0[	[740.0, 1017.0[	[957.0, 1234.0[	[463.0, 740.0[	[96.0, 643.0[	[752.0, 1080.0[
LowerE	[275.0, 348.0[	[128.0, 300.0[	[276.0, 276.0[	[0.0, 0.0[	[386.0, 432.0[	[510.0, 510.0[	[827.0, 827.0[	[831.0, 831.0[	[935.0, 1229.0[	[1016.0, 1089.0[	[1233.0, 1273.0[	[739.0, 812.0[	[372.0, 715.0[	[804.0, 804.0[
Electrical	[386.0, 780.0[	[338.0, 732.0[	[110.0, 708.0[	[386.0, 432.0[	[0.0, 0.0[	[348.0, 348.0[	[644.0, 665.0[	[669.0, 669.0[	[773.0, 1067.0[	[734.0, 1014.0[	[951.0, 1111.0[	[457.0, 737.0[	[90.0, 1104.0[	[642.0, 642.0[
Storage	[734.0, 853.0[	[638.0, 810.0[	[458.0, 786.0[	[510.0, 510.0[	[348.0, 348.0[	[0.0, 0.0[	[317.0, 317.0[	[321.0, 321.0[	[425.0, 719.0[	[538.0, 666.0[	[755.0, 763.0[	[389.0, 389.0[	[438.0, 756.0[	[294.0, 294.0[
P1	[651.0, 818.0[	[798.0, 1118.0[	[650.0, 1094.0[	[827.0, 827.0[	[644.0, 665.0[	[317.0, 317.0[	[0.0, 0.0[	[242.0, 638.0[	[216.0, 999.0[	[329.0, 631.0[	[546.0, 848.0[	[187.0, 354.0[	[554.0, 721.0[	[469.0, 611.0[
Shield	[766.0, 1174.0[	[913.0, 1131.0[	[765.0, 1107.0[	[831.0, 831.0[	[669.0, 669.0[	[321.0, 321.0[	[242.0, 638.0[	[0.0, 0.0[	[104.0, 398.0[	[217.0, 470.0[	[434.0, 442.0[	[302.0, 710.0[	[669.0, 1077.0[	[227.0, 227.0[
Navigations	[807.0, 1109.0[	[954.0, 1409.0[	[806.0, 1385.0[	[935.0, 1229.0[	[773.0, 1067.0[	[425.0, 719.0[	[216.0, 999.0[	[104.0, 398.0[	[0.0, 0.0[	[113.0, 368.0[	[330.0, 340.0[	[343.0, 645.0[	[710.0, 1012.0[	[331.0, 625.0[
Weapons	[741.0, 741.0[	[888.0, 1041.0[	[740.0, 1017.0[	[1016.0, 1089.0[	[734.0, 1014.0[	[538.0, 666.0[	[329.0, 631.0[	[217.0, 470.0[	[113.0, 368.0[	[0.0, 0.0[	[217.0, 217.0[	[277.0, 277.0[	[644.0, 644.0[	[444.0, 697.0[
O2	[958.0, 958.0[	[1105.0, 1258.0[	[957.0, 1234.0[	[1233.0, 1273.0[	[951.0, 1111.0[	[755.0, 763.0[	[546.0, 848.0[	[434.0, 442.0[	[330.0, 340.0[	[217.0, 217.0[	[0.0, 0.0[	[494.0, 494.0[	[861.0, 861.0[	[661.0, 669.0[
Cafeteria	[464.0, 464.0[	[611.0, 764.0[	[463.0, 740.0[	[739.0, 812.0[	[457.0, 737.0[	[389.0, 389.0[	[187.0, 354.0[	[302.0, 710.0[	[343.0, 645.0[	[277.0, 277.0[	[494.0, 494.0[	[0.0, 0.0[	[367.0, 367.0[	[529.0, 683.0[
Medbay	[367.0, 367.0[	[324.0, 667.0[	[96.0, 643.0[	[372.0, 715.0[	[90.0, 1104.0[	[438.0, 756.0[	[554.0, 721.0[	[669.0, 1077.0[	[710.0, 1012.0[	[644.0, 644.0[	[861.0, 861.0[	[367.0, 367.0[	[0.0, 0.0[	[732.0, 1050.0[
P2	[993.0, 1147.0[	[932.0, 1104.0[	[752.0, 1080.0[	[804.0, 804.0[	[642.0, 642.0[	[294.0, 294.0[	[469.0, 611.0[	[227.0, 227.0[	[331.0, 625.0[	[444.0, 697.0[	[661.0, 669.0[	[529.0, 683.0[	[732.0, 1050.0[	[0.0, 0.0[

Figure 6- The interval of time for each pair of room where the traveler is an impostor

STEP 4

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

The model of the map is the same than for step3 crewmate map.

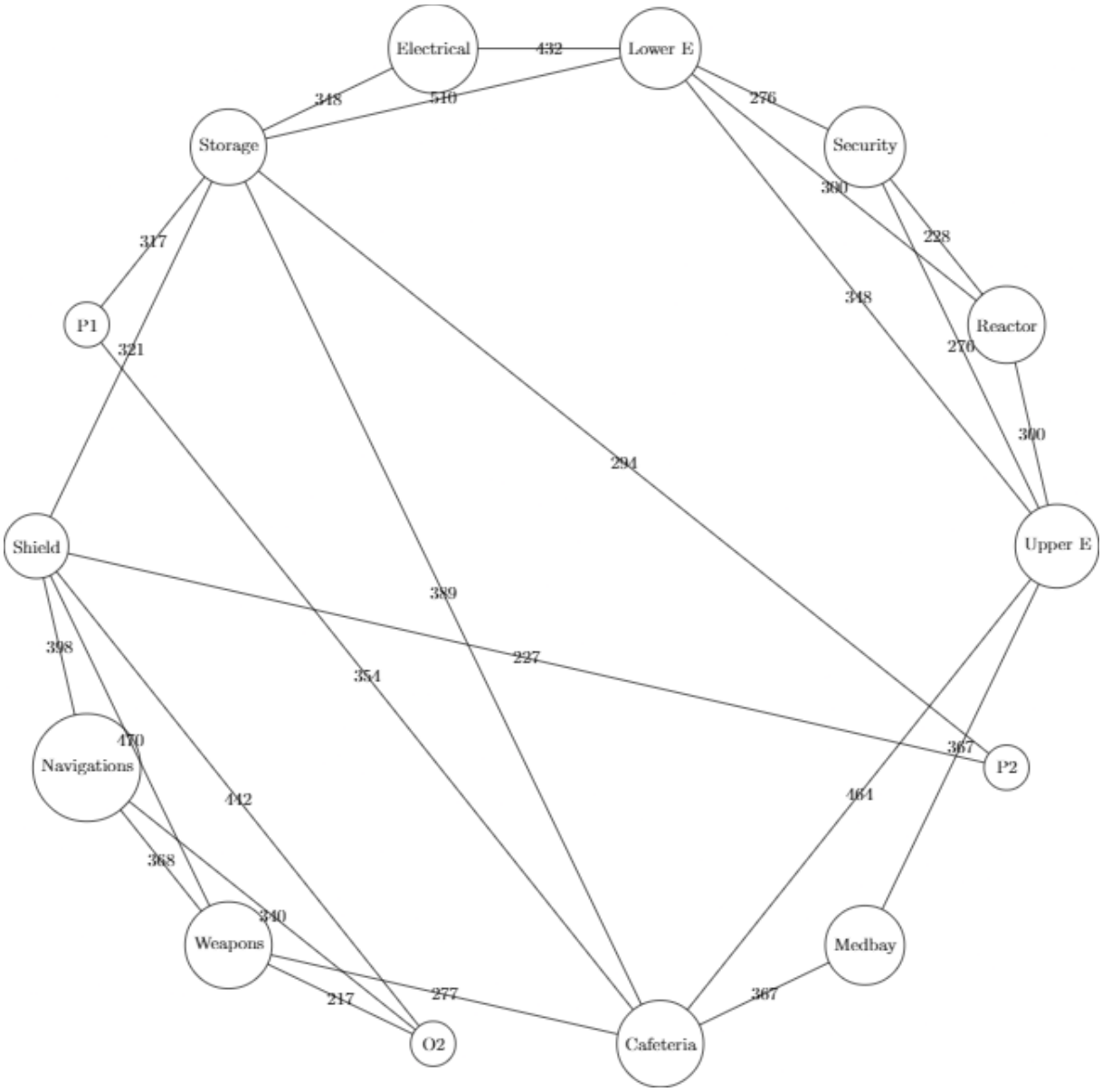


Figure 7- Map

Question 2-3-4: (Pack because two different model see next)

We make two resolution algorithms.

First resolution algorithm:

We considered the Hamiltonian problem. Going to all the nodes only one time.

So, we decided to implement a recursive Hamiltonian solver.

```
[['P2', 'Shield'],
 ['Shield', 'Navigations'],
 ['Navigations', 'O2'],
 ['O2', 'Weapons'],
 ['Weapons', 'Cafeteria'],
 ['Cafeteria', 'P1'],
 ['P1', 'Storage'],
 ['Storage', 'Electrical'],
 ['Electrical', 'LowerE'],
 ['LowerE', 'Reactor'],
 ['Reactor', 'Security'],
 ['Security', 'UpperE'],
 ['UpperE', 'Medbay']]
```

Figure 8- Hamiltonian path result

We get a minimum time travel of 4081s.

Second resolution algorithm:

The first algorithm is good. Yet in the case of more rooms, example 100 city it will not be efficient (long running time to get the result).

So, we decided to implement a second algorithm. It gives an approximation of the shorter path to visit every room.

For this algorithm we consider a complete graph where time travels are the results of step 3.

The algorithm is based on Prism. It begins with the circuit of the Prim graph and avoid cities when the direction of the current edge is an already visited city. Like this, we avoid passing twice in the same room.

```
Out[12]: [['Weapons', 'O2'],
          ['O2', 'Navigations'],
          ['Navigations', 'Cafeteria'],
          ['Cafeteria', 'P1'],
          ['P1', 'Storage'],
          ['Storage', 'P2'],
          ['P2', 'Shield'],
          ['Shield', 'Electrical'],
          ['Electrical', 'Medbay'],
          ['Medbay', 'UpperE'],
          ['UpperE', 'Security'],
          ['Security', 'Reactor'],
          ['Reactor', 'LowerE']]
```

Figure 9- Path result of the second algorithm

```
Entrée [14]: pathdistance(PrimCyclePath)
    Out[14]: 15552.0

Entrée [15]: pathdistance(ResultPath)
    Out[15]: 5338.0
```

Figure 10 - Comparison of time travel between Prim Cycle and our second algorithm

As you can see our second algorithm give a path 3 times shorter than the Prim Cycle path algorithm.

The time travel of the path given by the second algorithm is more important than the first. (4081s vs 5338s).
Yet it can help if the problem is bigger because its complexity is lower so the running time will be shorter.