ADVANCED DATA STRUCTURE AND ALGORITHMS MINI-PROBLEM

STEP 1: TO ORGANIZE THE TOURNAMENT

Zerator asks you to organize the next “Among Us” tournament for the next ZLAN. The rules are as following:

- Total of 100 players
- 10 players per game
- 3 random games then
- each game regroups the players by a batch of ten following their ranking.
    - The last 10 players (in the ranking) are ejected to the tournament.
    - Do it until it remains only 10 players
- For the last 10 players, play 5 games with reinitiated ranking. Update and check the ranking of the 10 players and give the podium.

Here is the ranking model:

- Impostor: 1pts per kill, 3pts per undiscovered murder, 10pts if win
- Crewmate: 3pts if the argument unmasks an imposter, 1pts if all solo tasks are made, 5pts if win

Each time a game ended, the score of each player is the mean of all its games.

The players are stored in a structured database with a log complexity to reach an element which corresponds to a score (the most optimized structure presented in the ADSA Course).

To organize the tournament, we need a total of 100 players. And to do so we created a class Player in which we have the player's name and its score.

Since we have 100 players, we need to have to give them a random name and a random score. Therefore, we will use the two functions:

1. ***random_player_name(length = 6)*** that randomize a player's name
2. ***random_player_score ()*** that randomize player's score at each game between 0 and 12 points

Once we have the player's name and score, we need to store them in a structured database with a log complexity to reach an element which corresponds to a score and the most optimized structure for this was to store all the players in an AVL tree and there comes the class AVLTree(object) :

In this class, you'll have different functions:

1. ***insert_node(self, root, key)*** which will insert players one by one based on their score and will make sure to get the balance factor because we know an AVL tree have to be balanced in order to work and if there is an unbalanced node, it will try out the 4 cases of unbalanced nodes and perform rotation if needed

2. ***delete_node (self, root, key)*** which will delete players who have the lowest scores this function will also make sure that once players are deleted from the tree, the tree remains an AVL tree and not a tree with unbalanced nodes

These two are the main functions in this class but there are other little functions that we use in those main functions such as

- ***leftRotate, rightRotate*** in order to perform rotations
- ***getMinValueNode, getMaxValueNode*** to have the minimum and maximum value of the tree
- ***printHelper*** to print the tree

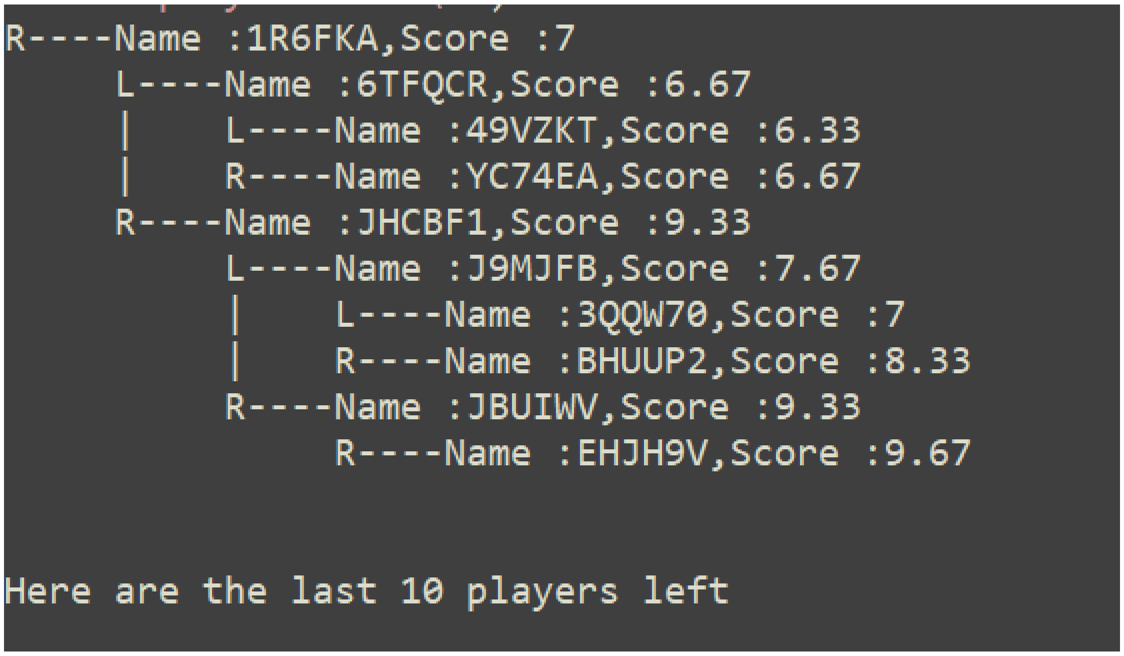
Now we will create our AVL tree using the function Creation_Tree in which we will use our class ***AVLTree*** and then insert 100 players (with their random names and scores).

The rule is:

Each player will play 3 random games and its score will the mean of all its score obtained in each game (3 games here). For this, we decided that we will implement a function called ***update_player_score ()*** that will update the player's score in our AVL tree.

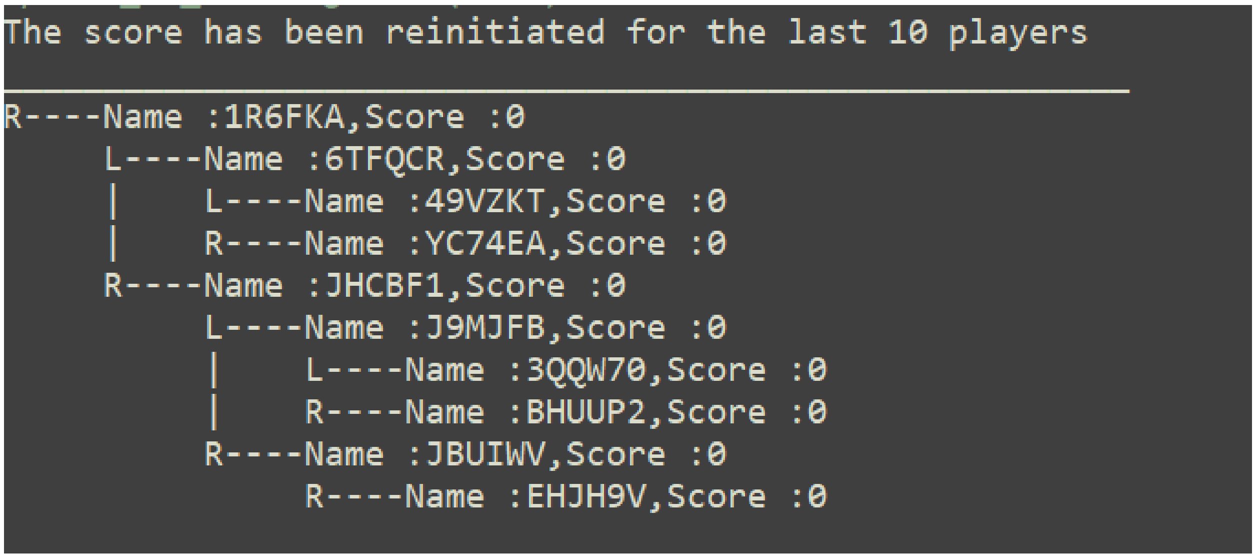
After playing the game and updating the player's score, we need to delete the last 10 players out of our AVL tree and for that we will our function called ***delete_last_ten_players ()*** that will identify the last ten players in our AVL tree through the function ***getMinValueNode()*** and then delete them so that only 90 players will be left

We will repeat the process ***update_player_score ()*** and ***delete_last_ten_players ()*** 9 times in order to have just 10 players at the end of our tournament.



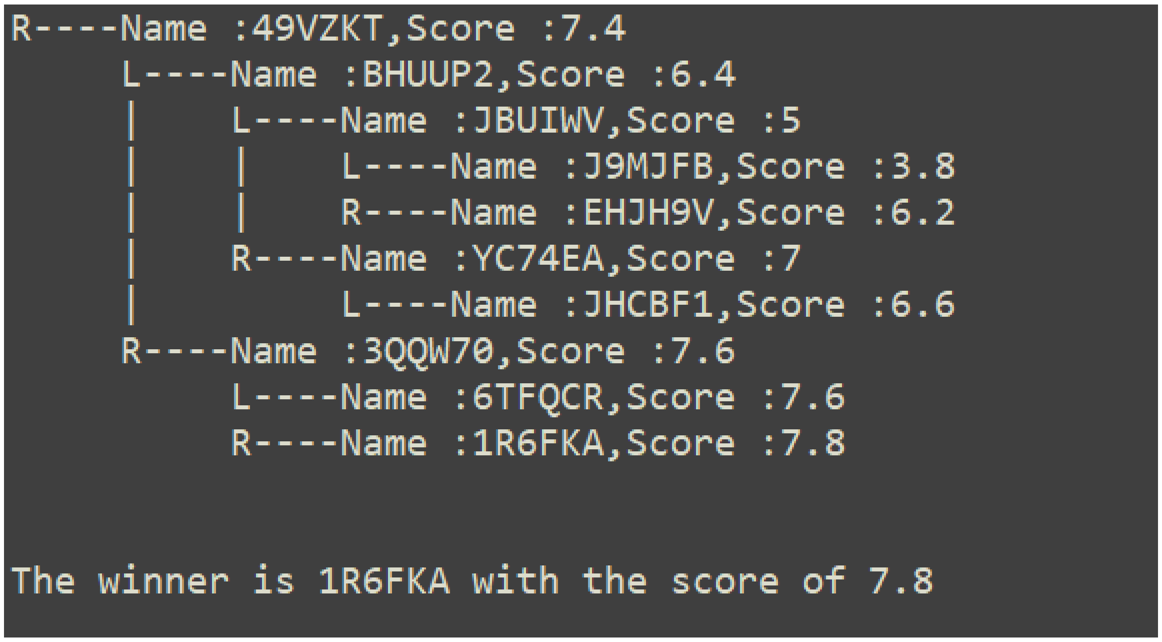
```
R----Name :1R6FKA,Score :7
     L----Name :6TFQCR,Score :6.67
     |    L----Name :49VZKT,Score :6.33
     |    R----Name :YC74EA,Score :6.67
     R----Name :JHCBF1,Score :9.33
          L----Name :J9MJFB,Score :7.67
          |    L----Name :3QQW70,Score :7
          |    R----Name :BHUUP2,Score :8.33
          R----Name :JBUIWV,Score :9.33
               R----Name :EHJH9V,Score :9.67


Here are the last 10 players left
```

For the last 10 players, we will use the function ***reinitiate_last_ten_players_score ()*** which will reinitiate the players' score at 0



```
The score has been reinitiated for the last 10 players
_______________________________________________________
R----Name :1R6FKA,Score :0
     L----Name :6TFQCR,Score :0
     |    L----Name :49VZKT,Score :0
     |    R----Name :YC74EA,Score :0
     R----Name :JHCBF1,Score :0
          L----Name :J9MJFB,Score :0
          |    L----Name :3QQW70,Score :0
          |    R----Name :BHUUP2,Score :0
          R----Name :JBUIWV,Score :0
               R----Name :EHJH9V,Score :0
```

After reinitiate the players' score, we need to play 5 random games and its score will the mean of all its score obtained in each game (5 games here) and based on the ranking the player who has the highest score will be the winner of the tournament.



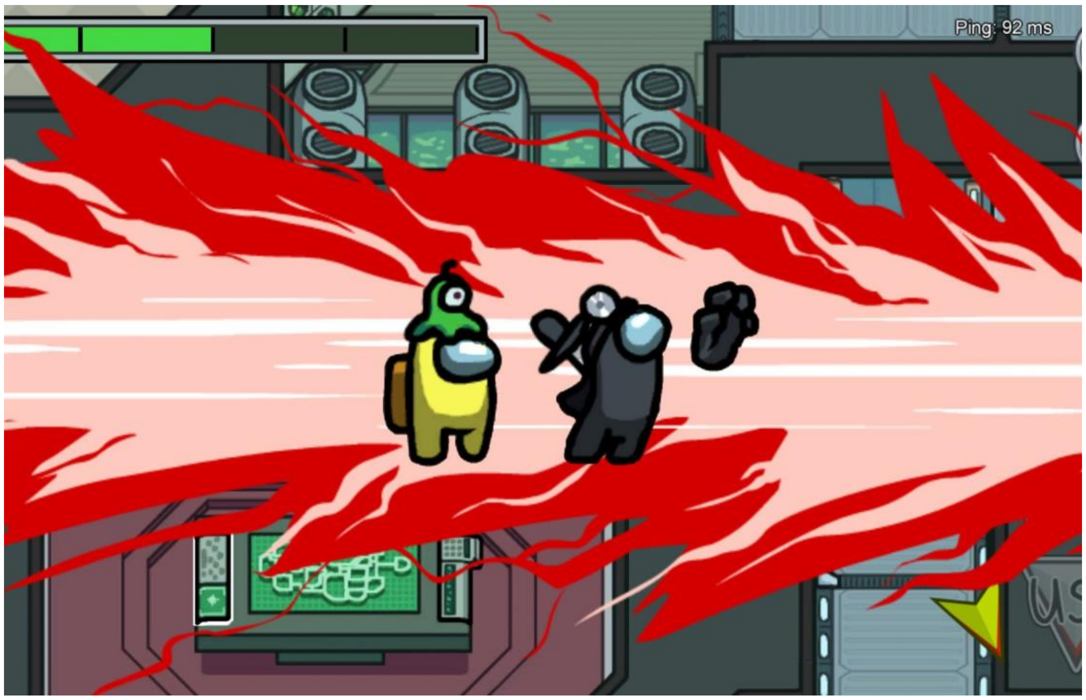
```
R----Name :49VZKT,Score :7.4
     L----Name :BHUUP2,Score :6.4
     |    L----Name :JBUIWV,Score :5
     |    |    L----Name :J9MJFB,Score :3.8
     |    |    R----Name :EHJH9V,Score :6.2
     |    R----Name :YC74EA,Score :7
     |         L----Name :JHCBF1,Score :6.6
     R----Name :3QQW70,Score :7.6
          L----Name :6TFQCR,Score :7.6
          R----Name :1R6FKA,Score :7.8


The winner is 1R6FKA with the score of 7.8
```

Here you can see the player 1R6FKA is the winner after scoring 7.8 points.

After all that, we were discussing about creating just one main function and we thought about creation the function ***Tournament ()*** which is the main and the last function of this program.

This function will do the tournament between 100 players until it remains only 10 players and then for the last players play the 5 games and give the podium to the winner.

STEP 2: PROFESSOR LAYTON < GUYBRUSH THREEPWOOD < YOU



You're not only the tournament organizer, you're also a player. Thus, you must find the best strategies to grab points to climb the ladder.

Most of the time, the two Impostors are among Crewmates. They never walk together. Thus, the information about players which are seen together may help to find the Impostor. After the first kill's report, the following information presents the players which see each other's:

- Player 0 has seen player 1, 4 and 5
- Player 1 has seen player 0, 2 and 6
- Player 2 has seen player 1, 3 and 7
- Player 3 has seen player 2, 4 and 8
- Player 4 has seen player 0, 3 and 9
- Player 5 has seen player 0, 7 and 8
- Player 6 has seen player 1, 8 and 9
- Player 7 has seen player 2, 5 and 9
- Player 8 has seen player 3, 5 and 6
- Player 9 has seen player 4, 6 and 7.

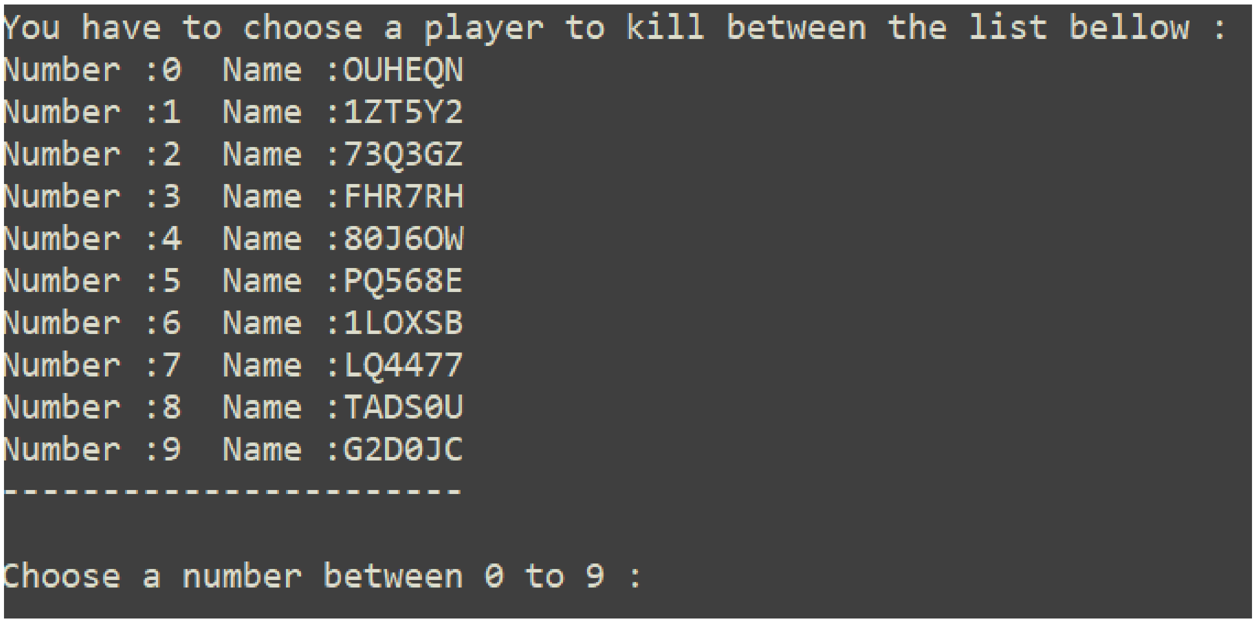
1. Represent the relation (have seen) between players as a graph, argue about your model.

- To represent the relation between players, we made a list in the class Player where we will find the three players that each player will cross.
- For the representation of the relations between players, we had a choice to go either with Incidence (matrix) or adjacency (matrix/list) representation. We decided to go with the adjacency representation because it considers the relation between nodes more than the distance between nodes and here distance between nodes is not useful.
- The next step was to implement a program that assigns each player a list of players that he crossed called ***voisins ().*** To do so, we will use a list called b where we will find all the players of the game. For each player, we initialize a list that will be filled with its possible neighbors. This list will first be filled with players who do not have neighbors. If the number of players without a neighbor is less than the number of places available in the player's list of neighbors, then we add to b the players who already have a neighbor. And if we see that despite this the size of b is still lower than the number of places available in the player's neighbor list, then we add players who already have 2 neighbors. You must be careful to not fill in the current player's list b because he cannot be his own neighbor for this reason a condition is added. Once the b-list is well filled, we will randomly draw a player from this list, and we will consider him as a neighbor of the current player, and we add the current player to the list of neighbors of the randomly chosen player. After doing this, we noticed that we could still have errors when filled in the lists of players because it may be that the random chosen to fill the lists that some players. To avoid this, we add a condition that allows to assign to the list of the current player the players who have the least number of neighbors.

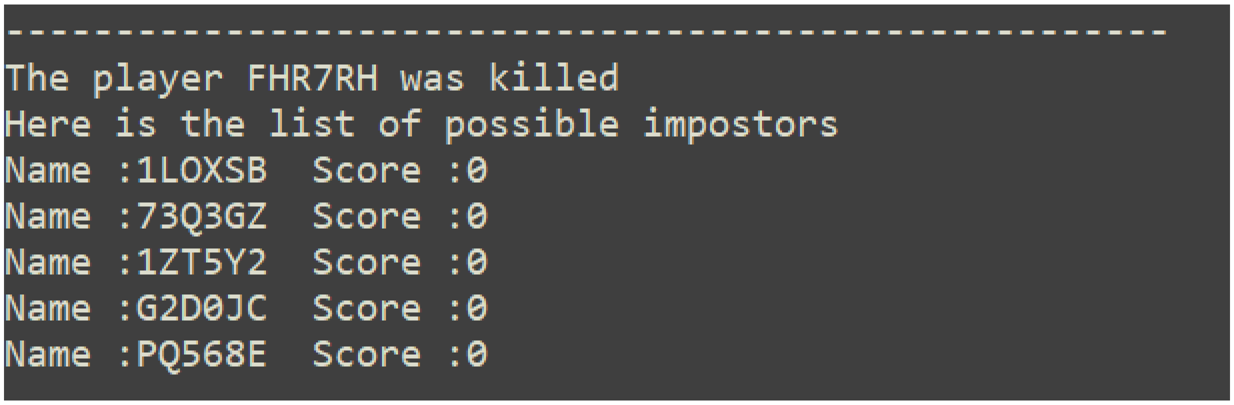
2. Thanks to a graph theory problem, present how to find a set of probable impostors.

- In order to find the likely set of impostors, we first look at the players who crossed the dead, then we initialize a set called Probable_impostors where we will first put the 3 players crossed by the dead. We add to this list all the other players who have not crossed any of his 3 players.
- Let's use the example given in the statement to better explain our program. In this case, it is the 0 player who died, and this player crossed players 1,4,5 so these three players are put directly in the Probable_imposors list. We then look at all the other players in the game, if among the players that they crossed this find one of the 3 players mentioned earlier then we do not consider them as possible impostor. In the case of the example, all the other players have crossed at least one of the players who are already in the list of Probable_impostors so we don't have any other players to put in the list. In other cases, there may be players who have not crossed any of these 3 players and in this case, we add them to our list.

Here is a display of the result:

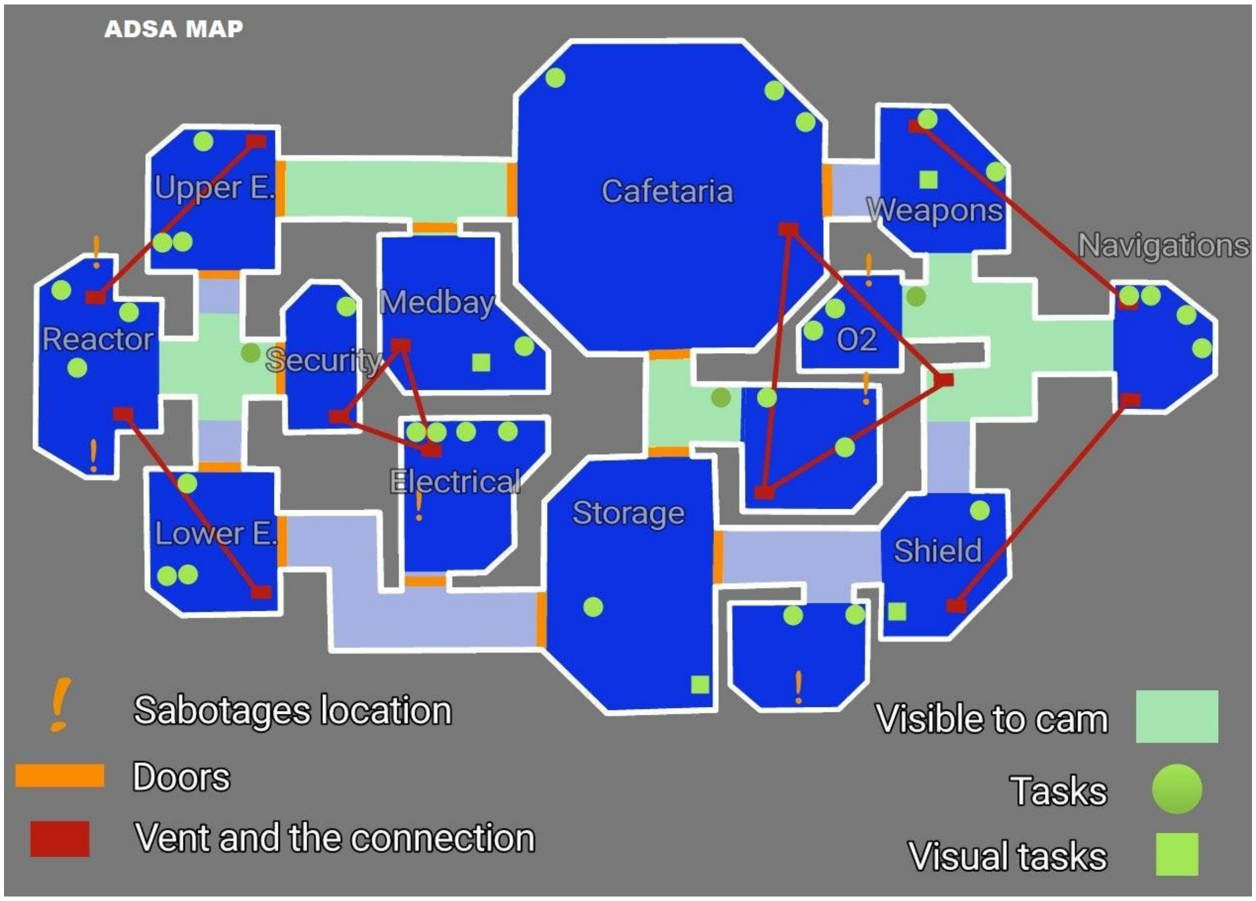


```
You have to choose a player to kill between the list bellow :
Number :0  Name :OUHEQN
Number :1  Name :1ZT5Y2
Number :2  Name :73Q3GZ
Number :3  Name :FHR7RH
Number :4  Name :80J6OW
Number :5  Name :PQ568E
Number :6  Name :1LOXSB
Number :7  Name :LQ4477
Number :8  Name :TADS0U
Number :9  Name :G2D0JC
----------------------

Choose a number between 0 to 9 :
```



```
----------------------------------------------------
The player FHR7RH was killed
Here is the list of possible impostors
Name :1LOXSB  Score :0
Name :73Q3GZ  Score :0
Name :1ZT5Y2  Score :0
Name :G2D0JC  Score :0
Name :PQ568E  Score :0
```

STEP 3: I DON'T SEE HIM, BUT I CAMJ GIVE PROOFS HE VENTS!



Considering that a player (a crewmate) can only walk through the map, but an impostor can also travel with vent, it is important to compute the time to travel between each room for crewmates and impostors. A room is represented by its center (you don't have to be precise).

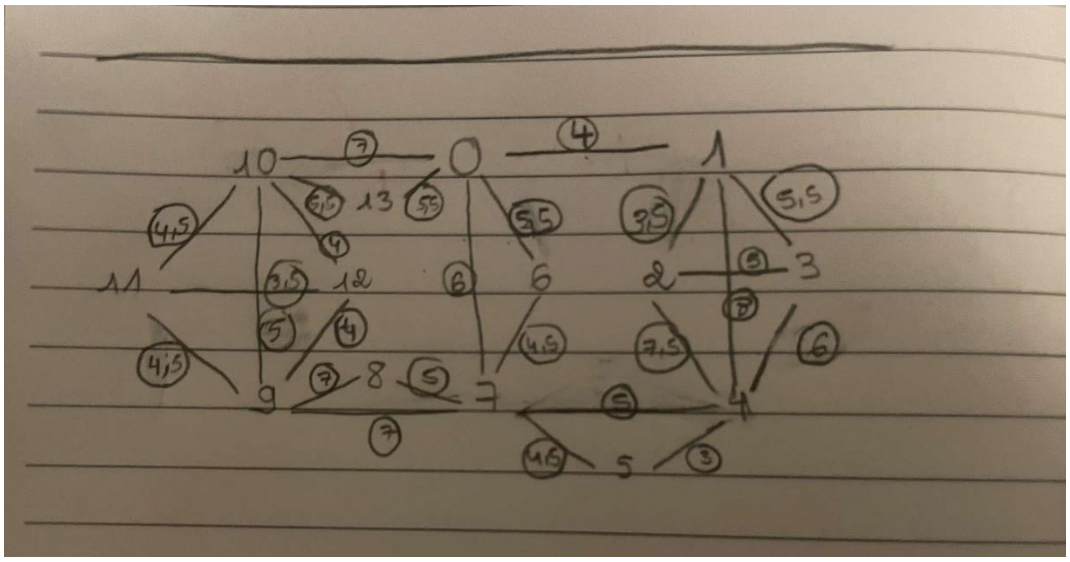
A room has a link with another room if there is a corridor between them. The time to travel 1cm is 1sec. You can draw a graph to model the ADSA MAP.

Impostor can also take the vent; the map shows the link between each vent. Taking a vent do not take time.

To unmask impostors, you have the idea to compare the time to travel between any pair of room in two cases: if you are a crewmate; if you are an impostor.

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

- The two models of the map will be Crewmates and Impostors. To build the model of the graph we decided to implement one graph for the Crewmates who can walk through the map and the other one for Impostors who can also walk through the map and use the vent.
- Since a room is represented by its center, we took the measurement from one room to another through a ruler and the measurement is approximatively accurate. This is the graph we obtained after taking each measurement between each room:



This is the order we took:

0 = Cafeteria

1 = Weapons

2 = O2

3 = Navigation

4 = Shield

5 = Communication

6 = Admin

7 = Storage

8 = Electrical

9 = Lower E

10 = Upper E

11 = Reactor

12 = Security

13 = Medbay

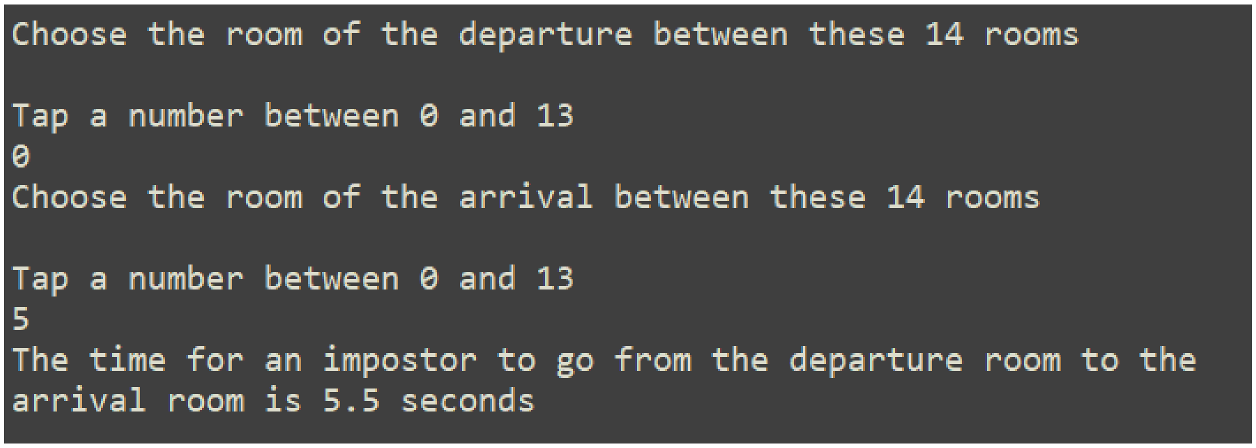
For the Crewmates map, we linked one room to another room according to the given map, we did the same thing for impostors, but we added extra connections between rooms because of vents.

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

After implementing these two graphs, we defined a function called floydWarshall (graph, n) which purpose is to determine the shortest paths between one room to another room.

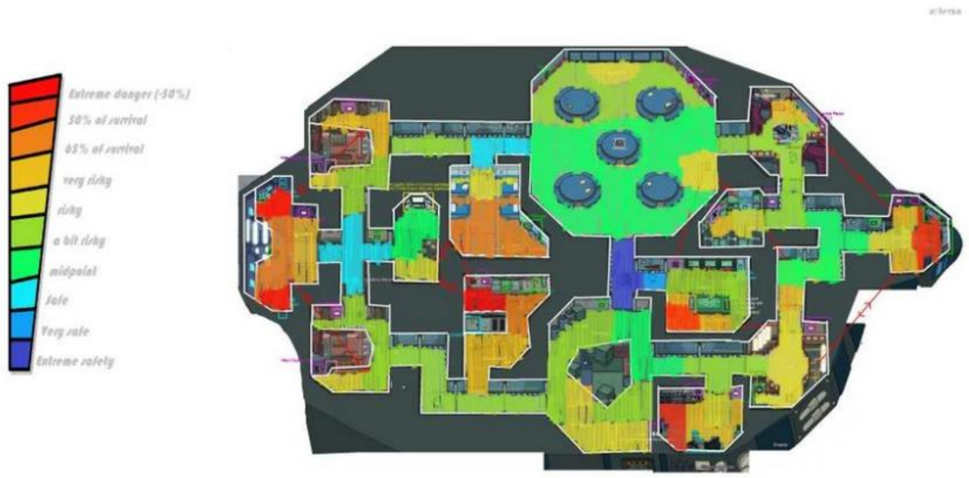
It will display the result in cm and the time to travel 1cm is 1sec so logically the displayed distance is the time (in sec) that it takes to travel from one room to another. Also, in the code you have the option to choose the room of departure and the room of arrival.

Here is an example and result:



```
Choose the room of the departure between these 14 rooms

Tap a number between 0 and 13
0
Choose the room of the arrival between these 14 rooms

Tap a number between 0 and 13
5
The time for an impostor to go from the departure room to the
arrival room is 5.5 seconds
```

STEP 4: SECURE THE LAST TASKS



Only few tasks remain, and you will win as a crewmate. You decide to finish the last tasks forming a pack with all remaining player. Indeed, in a pack, impostors cannot kill anyone, they will be unmasked.

The map is ADSA MAP. You need to go the quickest possible to finish all the remaining tasks before impostors distract the pack to its route. Thus, you decide to browse the map room by room, and to finish the task in the current room. A room will be visited only one time.

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

- In this step, we use the graph that we implemented in the step 3 which represented the path that crewmates could take.
- This graph was represented by an adjacency matrix

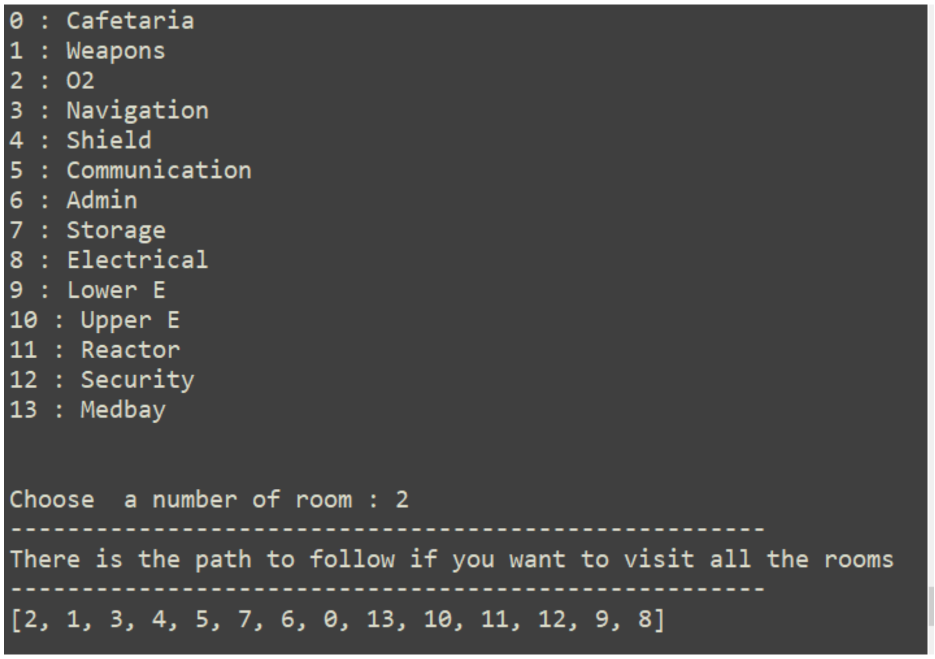
2.Thanks to a graph theory problem, present how to find a route passing through each room only one time.

- We tried the Prism algorithm but then we realized that id didn't achieve the desired result. Indeed, the prism algorithm like Krushal algorithm allows us to have the shortest paths without doing a cycle.
- Meanwhile we want to go through all the rooms only once. To resolve this problem, the algorithm that seemed the most logical was the Hamilton one.
- To use the Hamilton algorithm, we transformed our graph which was a matrix into a dictionary that has for each element, the different rooms it has a direct access.

3.Argue about an algorithm solving your problem.

- After implementing the Hamilton algorithm, we see that if we take a certain room as the departure room, we won't be able to do a proper cycle. For these rooms, we put them in a list called ***listeimpossible***. When the user enters one of these rooms as a departure room, a message is displayed informing that it's not possible.

Here is the display of the result:



```
0 : Cafetaria
1 : Weapons
2 : O2
3 : Navigation
4 : Shield
5 : Communication
6 : Admin
7 : Storage
8 : Electrical
9 : Lower E
10 : Upper E
11 : Reactor
12 : Security
13 : Medbay


Choose  a number of room : 2
-----------------------------------------------------
There is the path to follow if you want to visit all the rooms
-----------------------------------------------------
[2, 1, 3, 4, 5, 7, 6, 0, 13, 10, 11, 12, 9, 8]
```