ADSA Mini Problem report

VAREZ Arthur
Student number: 707085
Working Project Group: DIA5
22 December 2020

The project given is about famous game which became popular during this year thank's to a lot of streamers who played on it : Among Us. It's a very easy game to understand: 10 players are playing on a map. Among this 10 players there's two "imposters", whose goal is to kill all the crewmates (those who are not imposters). To achieve this goal they have severals extra tools : they can use vents to travel faster through the map and sabotage to disturb crewmates in order to split them. Meanwhile, crewmates goal is to finish tasks to do all over the map and try to not be killed by imposters.

In this project we will try to organise the next tournament for Zerator which will take place during his famous charity event : ZLAN. So this project could be divided in two parts :

- The organisation of the tournament and its simulation.
- The strategies developed as a crewmate to grab points in order to climb the ladder and to win the tournament by discovering the imposters on each game.

To do this, some rules and explanations are given :

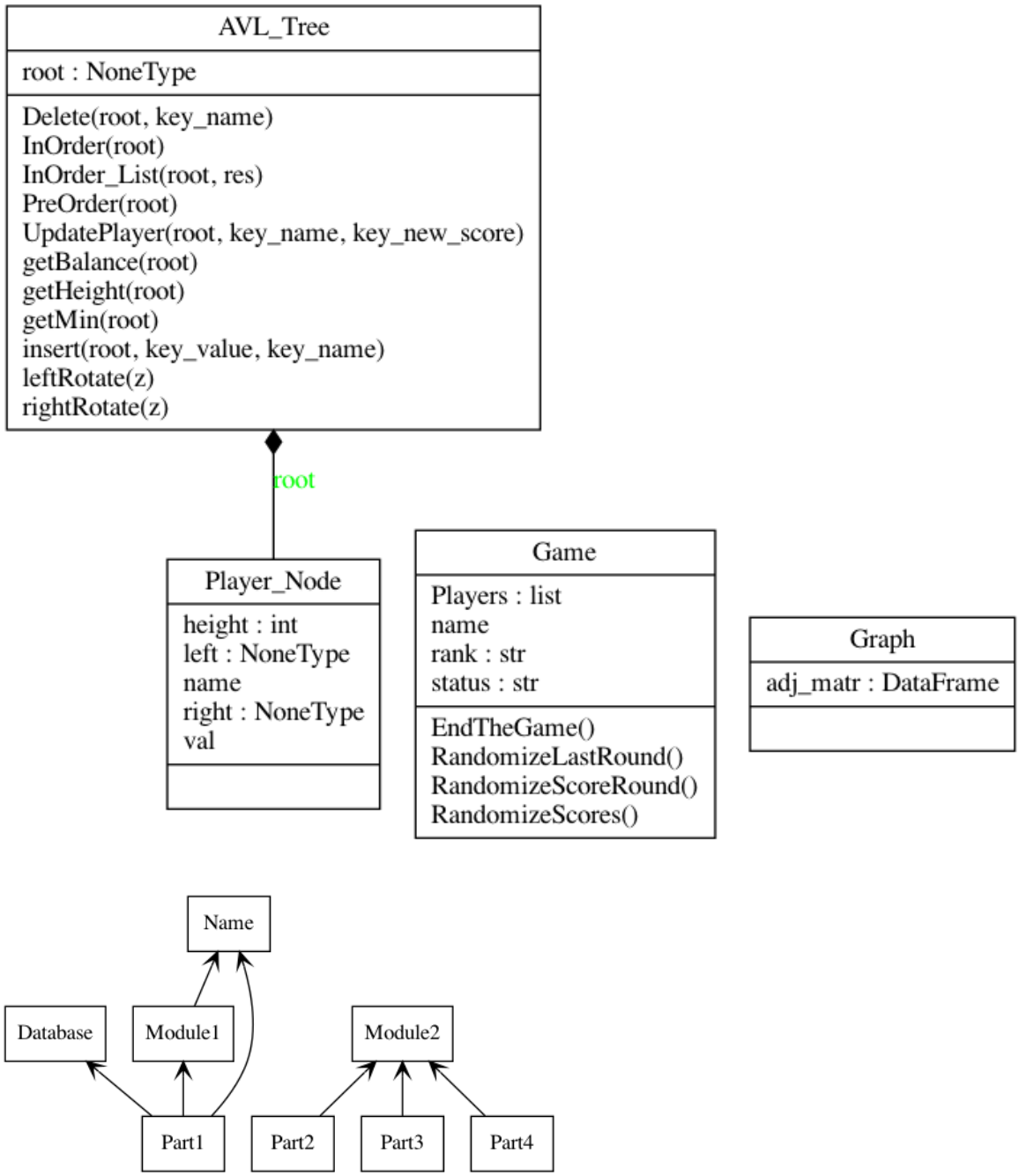
- We have a set of 100 players competing for the win.
- A game is a group of 10 players playing together with approximatively the same rank.
- A round is composed of 3 games.
- The scores of players are the mean of the points they got on the 3 games composing a round.
- At the end of the round, the 10 last players based on ranking are disqualified.
- We follow this pattern until there's until 10 players.
- Then for the last round, scores are reset and now this round is composed of 5 games. A the end of it, we get the final ranking.

Now, as we have defined and clarified the processing to follow to simulate the tournament, we can go deeper into the code explanation.

Firstly we will discuss about the organisation of my code and why i decided to do as this. I divided this project into 8 python files which are described below :

- Part1,Part2,Part3 and Part4 : python files on which the solution of each belonging part is running.
- Module1 : Module containing a class named Game. This class is useful for the first part.
- Module2 : Contains the Graph class which help us to to develop solution for Part1, Part2 and Part3. There are also severals methods allowing to load graph and to well define them.
- Name : Module on which there is a method to generate random names and an other one to generate scores.
- Database : In this file I've created two classes in order to construct the database used in the first part.
- Text files (graph,map_crewmate,map_imposter) : Txt files, loaded in the code by several methods, describing some of the data-structures we will use in this project.

Here is the UML of the project to see the code organisation and how the modules work together:



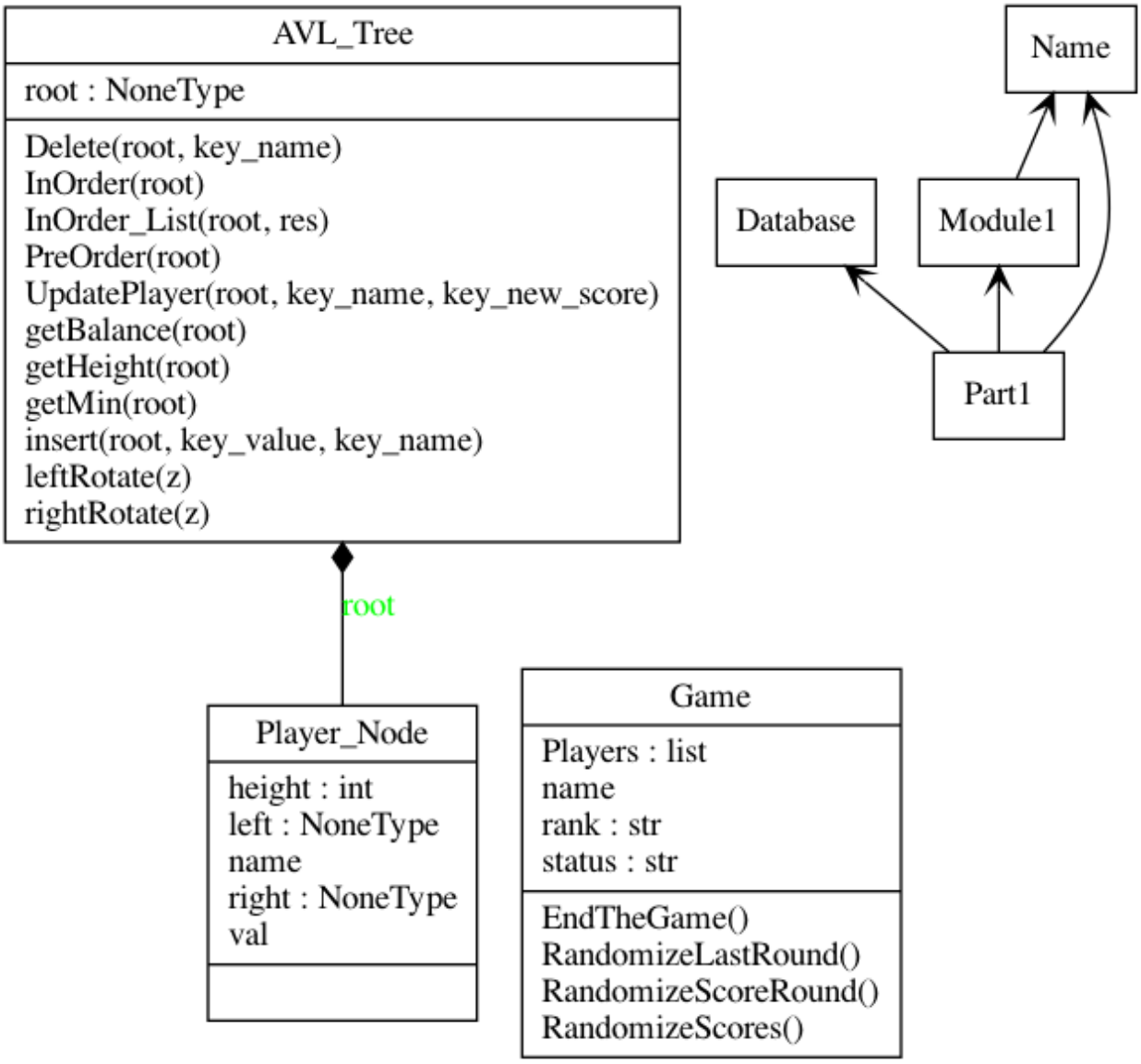
As this little introduction has been made to explain the context and the way I choose to do this project, we can now discuss about the solution I brought for each part.

In the following parts, console outputs and images will be join to this report to help you to better visualise how algorithms work and to show my personal process to deal with the subject.

First Part : How to organise the Tournament

Files required for this part : Part1.py, Module1.py, Database.py, Name.py, Game.py.

Here is the UML related to this part (generated with pyreverse):



The first step to organise the tournament is to define an optimised data structure to store the players and their scores. Firstly I though of dictionaries to represent players and their scores. But as we need a data structure seen in our courses with a log complexity, I decided to go for an AVL Tree.

So I created a class named Player_Node which contains the informations about a player (score, name and height). Every instance of this class could be connected by other instances of this same class according linking between players. There are some overided methods such as str method to display a node and eq method to compare two nodes.

Now as we defined the class presented above we can discuss about the data-structure I used for this part : an AVL tree. This trees are self-balancing binary search trees, which allow to have a log n complexity for searching, inserting and deleting. Here, nodes of my Tree are Player_Node instances of the class defined previously. The linking is done between nodes by the attribute “name” in order to keep flexibility when we want to update the scores (attribute val in Player_Node class).

Let’s discuss about the methods implemented in this class too:

- getBalance : return the balance factor of the tree
- getHeight : return the height of the tree
- getMin : return the lowest element of the tree
- leftRotate : does a left rotation of the tree (used for rebalance the tree)
- rightRotate : does a right rotation of the tree (used for rebalance the tree)
- insert : insert an element in the tree by giving a score and a name. The algorithm travel through the tree to find where to insert the future node being created. Then, thank’s to the function we defined above, the tree is rebalanced.

- Delete : Same logic as insert algorithm but we delete an node instead of inserting
- UpdatePlayer : given a Node_Player, the algorithm search into the AVLTree in order to update the score of the searched node.
- InOrder : Display the elements of the tree in order
- InOrderList : Almost the same as InOrder but here instead of displaying the nodes of the tree, we return them. It allows us to work on the players for the simulation of the tournament.
- PreOrder : Display the nodes in this order: root node is visited first, then the left subtree and finally the right subtree.

Next step is to create games and how to randomise scores for a game and after for a round.

To achieve this goal, I've to introduce my Game class, implemented in Module1.

This class can be used to generate random games but also ranking games. Given a name, which in my case is randomise thank's to the method generate_word of the Name module, and a list of Players_Nodes if you want, a game is created based on ranking or not if you precise it into the parameters.

Once we've created a game, we can randomise the score of each player with a random float generator and actualise the element of the current game depending of the configuration (You can randomise the score for a game, a round or also the final rank).

Here is an example of the output you can have:

```
In [69]: runfile('/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced Data/Project/Part1.py', wdir='/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced
Data/Project')
Reloaded modules: Name, Module1, Database
Name = test, Rank = No rank, Status = In progress

List of Players :

Name : Player1, score : 0.0
Name : Player10, score : 0.0
Name : Player100, score : 0.0
Name : Player11, score : 0.0
Name : Player12, score : 0.0
Name : Player13, score : 0.0
Name : Player14, score : 0.0
Name : Player15, score : 0.0
Name : Player16, score : 0.0
Name : Player17, score : 0.0

Name = test, Rank = No rank, Status = In progress

List of Players :

Name : Player1, score : 3.0
Name : Player10, score : 6.666666666666667
Name : Player100, score : 9.0
Name : Player11, score : 5.0
Name : Player12, score : 7.0
Name : Player13, score : 5.666666666666667
Name : Player14, score : 6.666666666666667
Name : Player15, score : 3.3333333333333335
Name : Player16, score : 9.333333333333334
Name : Player17, score : 5.666666666666667
```

Let's discuss about updating players' scores:

Thank's to how I've implement my database and my Game class, I was able to simplify the way to update scores in the database. In fact the database is an AVLTree with Node_Players objects. The list of players I've been using during the project are composed of the same elements used in the tree. So when I change the scores of the players when creating a game and randomising the scores, it also update the corresponding node of the AVLTree database.

Using the same example than before, here is the output when a display of the tree is asked:

```
In [74]: Database.InOrder(Database.root)
name : Player1,score : 3.0
name : Player10,score : 6.666666666666667
name : Player100,score : 9.0
name : Player11,score : 5.0
name : Player12,score : 7.0
name : Player13,score : 5.666666666666667
name : Player14,score : 6.666666666666667
name : Player15,score : 3.3333333333333335
name : Player16,score : 9.333333333333334
name : Player17,score : 5.666666666666667
name : Player18,score : 0.0
name : Player19,score : 0.0
name : Player2,score : 0.0
name : Player20,score : 0.0
name : Player21,score : 0.0
name : Player22,score : 0.0
name : Player23,score : 0.0
name : Player24,score : 0.0
name : Player25,score : 0.0
name : Player26,score : 0.0
name : Player27,score : 0.0
name : Player28,score : 0.0
name : Player29,score : 0.0
name : Player3,score : 0.0
name : Player30,score : 0.0
name : Player31,score : 0.0
name : Player32,score : 0.0
name : Player33,score : 0.0
name : Player34,score : 0.0
name : Player35,score : 0.0
name : Player36,score : 0.0
name : Player37,score : 0.0
name : Player38,score : 0.0
name : Player39,score : 0.0
name : Player4,score : 0.0
name : Player40,score : 0.0
name : Player41,score : 0.0
name : Player42,score : 0.0
```

Creation of ranking and random games

For this two methods of Part1.py module below, the class and its methods described above are essential.

CreateRandomGame **:**

This method is used only once but has a key importance. In the beginning, the database is created and populated/initialised with the method PopulateDB which assigns a score of 0.0 at each player (float is really important here, I've tried to figure out why players had only int type scores for a long long time ...). Then the first game needs to be a random one because all players have a null score.

The concept is the following:

- Players objects of the database are returned with InOrderList method of AVLTree class

- Checking if there are enough players in the DB to create a Game. If so, a set object is defined(to strictly have different players) and given as parameter to create a Game object.

CreateRankingGame :

Firstly we need to define what is a rank. I decided to define ranks like that :

Rank1 : 10 best Players based on score

Rank2 : 10-20 Players

Rank3 : 20-30 Players

Rank4 : 30-40 Players

Rank5 : 40-50 Players

Rank6 : 50-60 Players

Rank7 : 60-70 Players

Rank8 : 70-80 Players

Rank9 : 80-90 Players

Rank10 : 10 worst Players

Now we have defined this, let's focus on the algorithm. Given the database and rank in the ones above (and a name, if not a name will be generate), a game with the ranking of the rank specified is created following the previous model. An extra method, GetRanks, is used to help the current one. Its job is to split the database into ranks. So thank's to it, a dictionary is returned with this pattern {"Ranki"= [ListofPlayers] }. Now, with the parameter rank, we only have to call the dictionary with this parameter as a key to get the list of Players in order to create a Game object.

Some outputs to visualise:

```
In [79]: runfile('/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced Data/Project/Part1.py', wdir='/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced
Data/Project')
Reloaded modules: Name, Module1, Database
Name = fisabu, Rank = No rank, Status = In progress

List of Players :

Name : Player96, score : 0.0
Name : Player67, score : 0.0
Name : Player11, score : 0.0
Name : Player47, score : 0.0
Name : Player56, score : 0.0
Name : Player85, score : 0.0
Name : Player87, score : 0.0
Name : Player30, score : 0.0
Name : Player62, score : 0.0
Name : Player94, score : 0.0

In [80]:
```

Random Game

```
In [83]: runfile('/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced Data/Project/Part1.py', wdir='/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced
Data/Project')
Reloaded modules: Name, Module1, Database
Name = fynezi, Rank = Rank3, Status = In progress

List of Players :

Name : Player51, score : 8.0
Name : Player65, score : 8.0
Name : Player84, score : 8.0
Name : Player99, score : 8.0
Name : Player15, score : 7.666666666666667
Name : Player75, score : 7.666666666666667
Name : Player80, score : 7.666666666666667
Name : Player97, score : 7.666666666666667
Name : Player12, score : 7.333333333333333
Name : Player39, score : 7.333333333333333

In [84]:
```

Ranking Game with Rank3 after the first Round

Playing until the 10 last players:

Previously, we've defined how to create random and ranking games. Now we have to define methods to play a round. The main issue here is that we have to create an undefined number of games for a round in which every player can only be in one group of players.

For the first round we need to play random games. So it leads me to create the method Play1stRound. This method creates a list of random games. We grab the list of players in the database and until this list is not empty, we create a game selecting randomly players of that collection and popping them out to be sure to not take them again. Once we've created all the games we can randomise scores of the players for a round, using RandomizeScoreRound function of Game class.

Here are the outputs:

```
In [90]: runfile('/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced Data/Project/Part1.py', wdir='/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced
Data/Project')
Reloaded modules: Name, Module1, Database
Name = kicacy, Rank = No rank, Status = In progress

List of Players :

Name : Player67, score : 1.0
Name : Player97, score : 5.666666666666667
Name : Player71, score : 7.0
Name : Player72, score : 8.0
Name : Player19, score : 8.666666666666666
Name : Player21, score : 8.0
Name : Player26, score : 7.0
Name : Player60, score : 3.0
Name : Player63, score : 9.0
Name : Player93, score : 11.666666666666666

Name = qecyvi, Rank = No rank, Status = In progress

List of Players :

Name : Player35, score : 5.0
Name : Player13, score : 8.0
Name : Player80, score : 8.0
Name : Player5, score : 6.0
Name : Player25, score : 9.333333333333334
Name : Player92, score : 6.666666666666667
Name : Player95, score : 7.0
Name : Player31, score : 8.666666666666666
Name : Player66, score : 8.666666666666666
Name : Player36, score : 6.0

Name = levabe, Rank = No rank, Status = In progress

Name = teguvy, Rank = No rank, Status = In progress

List of Players :

Name : Player15, score : 1.6666666666666667
Name : Player16, score : 4.666666666666667
Name : Player18, score : 8.666666666666666
Name : Player91, score : 8.0
Name : Player37, score : 5.333333333333333
Name : Player41, score : 7.333333333333333
Name : Player55, score : 7.0
Name : Player59, score : 2.0
Name : Player62, score : 11.666666666666666
Name : Player85, score : 8.0

Name = guhuja, Rank = No rank, Status = In progress

List of Players :

Name : Player11, score : 6.333333333333333
Name : Player17, score : 3.0
Name : Player24, score : 8.333333333333334
Name : Player38, score : 5.333333333333333
Name : Player43, score : 2.6666666666666665
Name : Player52, score : 4.0
Name : Player77, score : 4.666666666666667
Name : Player84, score : 5.666666666666667
Name : Player90, score : 5.666666666666667
Name : Player98, score : 4.0

Name = levabe, Rank = No rank, Status = In progress

List of Players :

Name : Player45, score : 4.0
Name : Player82, score : 9.666666666666666
Name : Player47, score : 6.666666666666667
Name : Player49, score : 6.666666666666667
Name : Player58, score : 5.0
Name : Player27, score : 7.333333333333333
Name : Player29, score : 9.666666666666666
Name : Player65, score : 8.0
Name : Player70, score : 4.666666666666667
Name : Player73, score : 9.333333333333334

Name = vajuko, Rank = No rank, Status = In progress

List of Players :

Name : Player1, score : 8.333333333333334
Name : Player96, score : 6.0
Name : Player54, score : 8.666666666666666
Name : Player99, score : 6.0
Name : Player12, score : 7.333333333333333
Name : Player22, score : 7.333333333333333
Name : Player28, score : 6.333333333333333
Name : Player30, score : 5.333333333333333
Name : Player78, score : 1.3333333333333333
Name : Player39, score : 5.333333333333333

Name = wezuje, Rank = No rank, Status = In progress

Name = kunoha, Rank = No rank, Status = In progress

List of Players :

Name : Player50, score : 4.333333333333333
Name : Player53, score : 3.6666666666666665
Name : Player56, score : 4.666666666666667
Name : Player7, score : 5.666666666666667
Name : Player79, score : 3.0
Name : Player8, score : 6.666666666666667
Name : Player86, score : 11.0
Name : Player88, score : 5.0
Name : Player9, score : 4.666666666666667
Name : Player94, score : 4.666666666666667

Name = lociwi, Rank = No rank, Status = In progress

List of Players :

Name : Player2, score : 6.666666666666667
Name : Player33, score : 4.333333333333333
Name : Player34, score : 7.333333333333333
Name : Player48, score : 2.6666666666666665
Name : Player57, score : 6.0
Name : Player6, score : 5.333333333333333
Name : Player61, score : 4.0
Name : Player81, score : 7.666666666666667
Name : Player83, score : 10.666666666666666
Name : Player89, score : 4.333333333333333

Name = wezuje, Rank = No rank, Status = In progress

List of Players :

Name : Player14, score : 8.0
Name : Player64, score : 2.3333333333333335
Name : Player69, score : 7.0
Name : Player20, score : 4.666666666666667
Name : Player23, score : 5.0
Name : Player3, score : 8.0
Name : Player32, score : 5.333333333333333
Name : Player87, score : 6.666666666666667
Name : Player46, score : 6.333333333333333
Name : Player51, score : 4.666666666666667

Name = kymupe, Rank = No rank, Status = In progress

List of Players :

Name : Player10, score : 7.0
Name : Player75, score : 8.0
Name : Player74, score : 9.0
Name : Player76, score : 5.0
Name : Player100, score : 5.0
Name : Player4, score : 7.666666666666667
Name : Player40, score : 7.333333333333333
Name : Player42, score : 2.3333333333333335
Name : Player44, score : 4.666666666666667
Name : Player68, score : 8.0

In [91]:
```

Next step is generating rounds based on ranking because we have scores not null for every players. To do so, I created the method Play_Ranking_Round. This one use the same principle as PlayRandomRound but applied on ranking game. We just have to be careful with our ranks. So we can assume that the number of ranks is lenght(DataBase) divided by 10. Now we have our ranks to apply the method CreateRankingGame.

Here are other outputs for this method:

```
In [97]: runfile('/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced Data/Project/Part1.py', wdir='/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced
Data/Project')
Reloaded modules: Name, Module1, Database
Name = pemina, Rank = Rank10, Status = In progress

List of Players :

Name : Player97, score : 8.333333333333334
Name : Player43, score : 9.666666666666668
Name : Player48, score : 8.666666666666668
Name : Player8, score : 10.333333333333332
Name : Player95, score : 8.666666666666668
Name : Player81, score : 9.333333333333334
Name : Player26, score : 7.333333333333334
Name : Player50, score : 7.666666666666666
Name : Player82, score : 10.0
Name : Player64, score : 7.666666666666666

Name = piguzo, Rank = Rank9, Status = In progress

List of Players :

Name : Player58, score : 10.0
Name : Player76, score : 6.0
Name : Player83, score : 10.666666666666668
Name : Player9, score : 8.333333333333332
Name : Player19, score : 13.0
Name : Player67, score : 8.666666666666666
Name : Player80, score : 5.333333333333333
Name : Player92, score : 7.666666666666666
Name : Player18, score : 6.666666666666667
Name : Player42, score : 9.333333333333334

Name = befate, Rank = Rank8, Status = In progress

Name = befate, Rank = Rank8, Status = In progress

List of Players :

Name : Player5, score : 13.333333333333332
Name : Player75, score : 11.0
Name : Player41, score : 9.333333333333332
Name : Player51, score : 10.333333333333332
Name : Player69, score : 8.333333333333332
Name : Player89, score : 13.666666666666668
Name : Player98, score : 13.333333333333332
Name : Player25, score : 15.0
Name : Player34, score : 8.0
Name : Player45, score : 12.666666666666666

Name = haloli, Rank = Rank7, Status = In progress

List of Players :

Name : Player4, score : 11.333333333333332
Name : Player56, score : 10.333333333333332
Name : Player59, score : 13.666666666666666
Name : Player61, score : 9.0
Name : Player73, score : 6.666666666666667
Name : Player84, score : 12.0
Name : Player16, score : 7.333333333333334
Name : Player29, score : 11.333333333333334
Name : Player32, score : 6.333333333333334
Name : Player46, score : 10.333333333333334

Name = zekewa, Rank = Rank6, Status = In progress

Name = zekewa, Rank = Rank6, Status = In progress

List of Players :

Name : Player60, score : 13.666666666666668
Name : Player86, score : 12.666666666666668
Name : Player100, score : 10.333333333333332
Name : Player52, score : 12.0
Name : Player65, score : 10.333333333333332
Name : Player68, score : 14.0
Name : Player70, score : 13.666666666666668
Name : Player99, score : 11.666666666666666
Name : Player22, score : 12.333333333333332
Name : Player35, score : 11.333333333333332

Name = pazijy, Rank = Rank5, Status = In progress

List of Players :

Name : Player1, score : 11.333333333333332
Name : Player17, score : 12.666666666666668
Name : Player20, score : 16.333333333333336
Name : Player44, score : 12.666666666666668
Name : Player14, score : 13.0
Name : Player27, score : 16.333333333333332
Name : Player30, score : 11.333333333333334
Name : Player36, score : 16.0
Name : Player49, score : 13.666666666666668
Name : Player55, score : 9.666666666666668

Name = ziveje, Rank = Rank4, Status = In progress

List of Players :

Name : Player28, score : 10.666666666666666
Name : Player66, score : 13.0
Name : Player77, score : 16.333333333333336
Name : Player78, score : 11.666666666666668
Name : Player2, score : 14.0
Name : Player24, score : 12.333333333333332
Name : Player3, score : 9.333333333333332
Name : Player40, score : 11.0
Name : Player54, score : 10.666666666666666
Name : Player91, score : 13.0

Name = tutolu, Rank = Rank3, Status = In progress

List of Players :

Name : Player71, score : 11.666666666666668
Name : Player79, score : 14.666666666666668
Name : Player88, score : 13.333333333333334
Name : Player96, score : 15.0
Name : Player23, score : 13.0
Name : Player47, score : 14.333333333333332
Name : Player57, score : 14.0
Name : Player74, score : 15.333333333333332
Name : Player87, score : 9.0
Name : Player12, score : 10.666666666666666

Name = dakovu, Rank = Rank2, Status = In progress

Name = dakovu, Rank = Rank2, Status = In progress

List of Players :

Name : Player72, score : 16.666666666666668
Name : Player94, score : 17.333333333333336
Name : Player33, score : 15.333333333333332
Name : Player85, score : 15.666666666666666
Name : Player11, score : 14.0
Name : Player13, score : 14.666666666666668
Name : Player31, score : 11.0
Name : Player37, score : 10.666666666666668
Name : Player38, score : 11.333333333333334
Name : Player39, score : 12.0

Name = hekoco, Rank = Rank1, Status = In progress

List of Players :

Name : Player53, score : 15.333333333333332
Name : Player7, score : 14.666666666666668
Name : Player21, score : 16.0
Name : Player62, score : 17.333333333333332
Name : Player6, score : 14.333333333333334
Name : Player90, score : 18.666666666666668
Name : Player93, score : 11.0
Name : Player10, score : 17.333333333333336
Name : Player15, score : 18.0
Name : Player63, score : 12.0


In [98]:
```

By now, we have methods to play a round no matters which one it is, we just have to define the last function: the Delete_Last10 function. This method takes all the players, sort them by score, drop the last 10 ones and create a new tree with the remaining players.

Just by making this 3 functions working together, we can play until 10 players. First round -> PlayFirstRound, Other rounds -> PlayRankingRound. Of course at the end of each round we delete the last 10 and we continue.

Output for PlayUntil10 :

```
In [99]: runfile('/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced Data/Project/Part1.py', wdir='/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced
Data/Project')
Reloaded modules: Name, Module1, Database
Here is the 10 last Players :

------------------------------------------------------

------------------------------------------------------

name : Player11,score : 62.666666666666664
name : Player14,score : 65.0
name : Player17,score : 65.33333333333333
name : Player26,score : 63.0
name : Player27,score : 64.33333333333333
name : Player3,score : 64.33333333333334
name : Player42,score : 66.33333333333333
name : Player43,score : 63.333333333333336
name : Player48,score : 72.66666666666666
name : Player91,score : 62.66666666666667

In [100]:
```

The Final Round: who's gonna be the winner ?

The battle had been relentless and there's only 10 survivors. Scores are reset to 0 and now it's time to play the last round composed of 5 games. The method PlayLastRound take the database as parameters and create a last ranking game. Then, with the method RandomizeLastRound of the class Game, we can randomise the scores. Finally, by using Display_Podium we display the winner.

Final output of this part:

```
In [108]: runfile('/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced Data/Project/Part1.py', wdir='/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced Data/Project')
Reloaded modules: Name, Module1, Database
Here are the 10 last Players :

------------------------------------------------------------

------------------------------------------------------------

name : Player20,score : 67.0
name : Player22,score : 71.66666666666667
name : Player3,score : 63.33333333333333
name : Player37,score : 66.33333333333333
name : Player41,score : 63.666666666666664
name : Player54,score : 65.33333333333334
name : Player77,score : 61.666666666666664
name : Player81,score : 65.66666666666666
name : Player9,score : 62.33333333333334
name : Player94,score : 63.0

------------------------------------------------------------


------------------------------------------------------------


Do you want to end the Game hutyra ?(yes/no)
yes
Game ended

average results of the final games :

Name = hutyra, Rank = Rank1, Status = Ended

List of Players :

Name : Player20, score : 4.4
Name : Player22, score : 6.6
Name : Player3, score : 4.6
Name : Player37, score : 3.8
Name : Player41, score : 5.4
Name : Player54, score : 7.6
Name : Player77, score : 6.2
Name : Player81, score : 4.0
Name : Player9, score : 6.6
Name : Player94, score : 6.8

Here is the podium of the tournament :


Winner , Name : Player54, score : 7.6


2nd , Name : Player94, score : 6.8


3rd , Name : Player22, score : 6.6


In [109]:
```

So now as all the Part1 is working well and the outputs seems to be up correct, we can move on Part2 : Find who are the imposters.

PART2 : Who are the bad guys ?

Files required for this part : Part2.py, Module2.py, graph.txt

Context:

So even if we have to organise the tournament, what we've done before, Zerator told us we can also be a player of this event. So here, we are trying to be the winner and therefore we must develop an efficient strategy to grab points and climb the ladder. Our strategie, as a crewmate, is to find out who are the imposters in the group of 10 players. A key information to do so is that imposters never walk together.

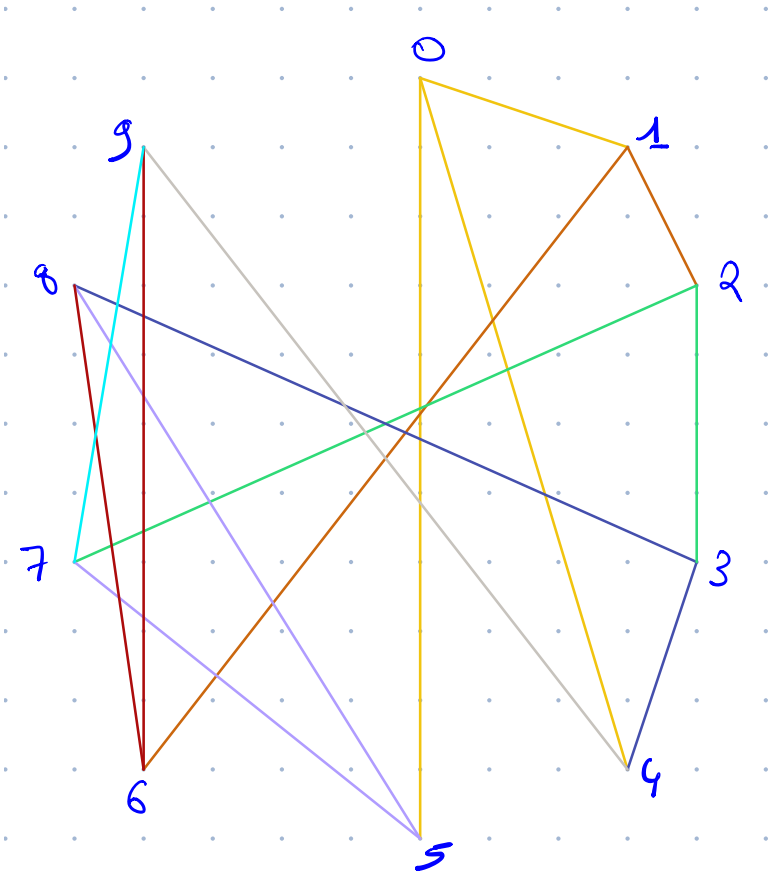
We have also informations about players who crossed theirs paths:

- 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

The situation takes places after player 0 has been found dead. We have to investigate about guilt of every player.

How to represent the relations between players ?:

First step is to report manually the datas to use them. So I decided to write a text file named graph.txt which represents the relations between 2 players on each line. Thank's to that, we will be able to load the datas into the code and to work on them. I've also draw a graph to better understand and visualise the "have seen relations " between players. Here it is :



There I choose to represent only relations in one way in order to keep this graph clean (for example 9-6 and 6-9 will be only one path, not two).

Then, as everything has been well explained and well defined, we can discuss about the algorithms and processing this datas.

I've decided to represent the relations graph in this part with a dictionary of this shape : {

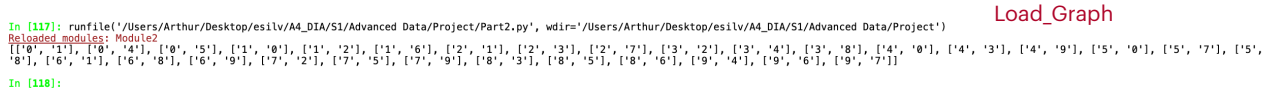
vertice : [edges_connected] }

vertice are keys and values are list of connected edges.

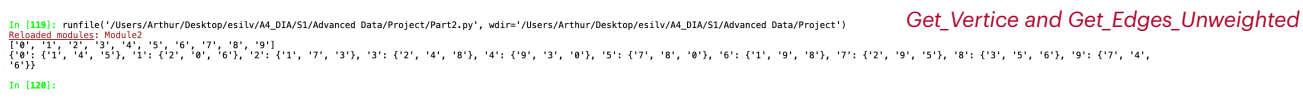
To do so, I've defined a first method to load the text file defined before and to convert it into a list of list. Now with the method Get_Vertice, a set of vertices is created to build a graph model. Last method, with Get_Edges_Unweighted, we can build our graph by assigning the edges on the vertices we get before. We need also ton convert the object of the graph into int type, it'll be easier to manipulate.

Outputs of this 3 functions:

Load_Graph



Get_Vertice and Get_Edges_Unweighted



Thank's to this methods, we have our data structure which will be used in this part. Now let's focus on finding a set of probable impostors.

Set of probables impostors :

We are going to work on the graph we've defined previously in this subpart to find the imposters. So 1,4 or 5 could be imposters and we assume

that on this set we could have two imposters. Using the graph, I've been able to implement the method Get_SetImposters to see what are the set of imposters.

output:

```
In [124]: Get_SetImposters(g)
Out[124]:
([{1, 4},
  {1, 5},
  {4, 5},
  {1, 3},
  {1, 7},
  {1, 8},
  {1, 9},
  {2, 4},
  {4, 6},
  {4, 7},
  {4, 8},
  {2, 5},
  {3, 5},
  {5, 6},
  {5, 9}],
 15)

In [125]:
```

This problem can also we solved by colouring nodes of our graph. The way to do it is this one: colour vertex 0 in the beginning and try to colour the others by checking the colours of the ones connected to it. Backtrack is gonna be the approach il will use to find solutions.

Graph colouring algorithm:

We use two functions working together to find solutions of this problem. Resolve_Coloring is the main method which sets the starting vertex and give it a colour. Then we use ColorGraph to find by backtracking the colours stratifying the conditions of colouring b returning the colours that could be used by the GetColors method.

A Solution:

```
{'0': 'green', '1': 'yellow', '2': 'green', '3': 'yellow', '4': 'red', '5': 'blue', '6': 'red', '7': 'yellow', '8': 'green', '9': 'green'}
```

PART3:
I don’t see him, but I can give proofs he vents !

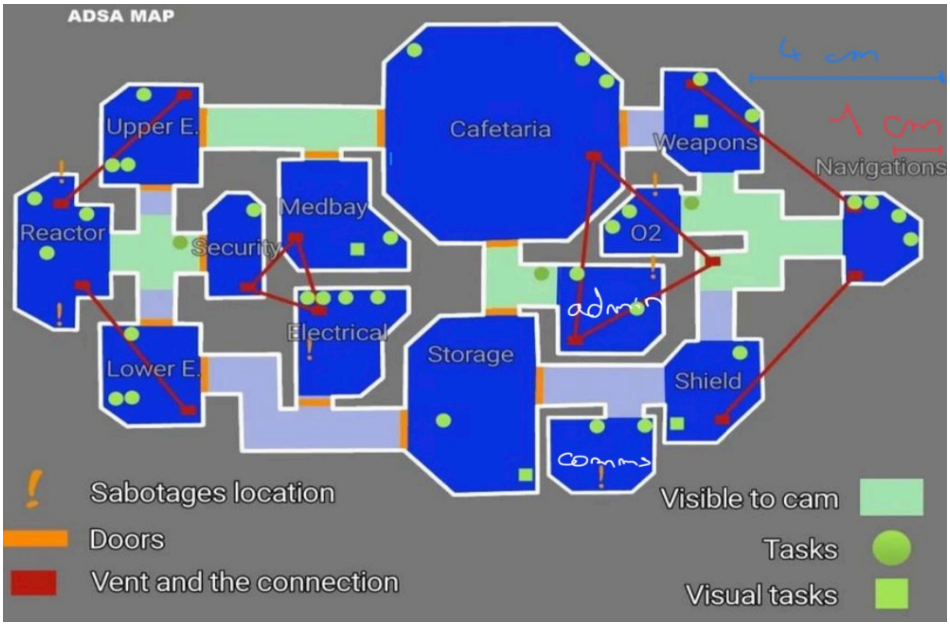
Files involved in this part: Part3.py, Module2.py, map_crewmate.txt, map_imposter.txt.

Introduction:

To continue on the idea to unmask imposters, like we tried to do in part2, we have an other idea to do so. In fact on the map there's shortcuts called "vents" that allows to travel faster through the map. That vents could only be taken by imposters and linked two rooms without travelling time. This fact leads to the idea to compare travelling time between players because imposters can walk faster from a room to an other if there is a vent on the way.

Imposter and crewmate : not the same map model

Firstly, we need to give a bit of explanations about the choices I've made to represent the map and how I prepare datas. Here is the map given but with some changes to better exploit it :



Here you can see I've precise the name of the rooms which were not given : Comms(communication) and Admin. This rooms needed to be named because we've to visit them if there are tasks to do in there. I've also defined my own measuring scale : a cm corresponds in fact of the distance between Weapons and Cafeteria. With this scale we can obtain the time to travel between two linked rooms (output provided by Load_Graph function):

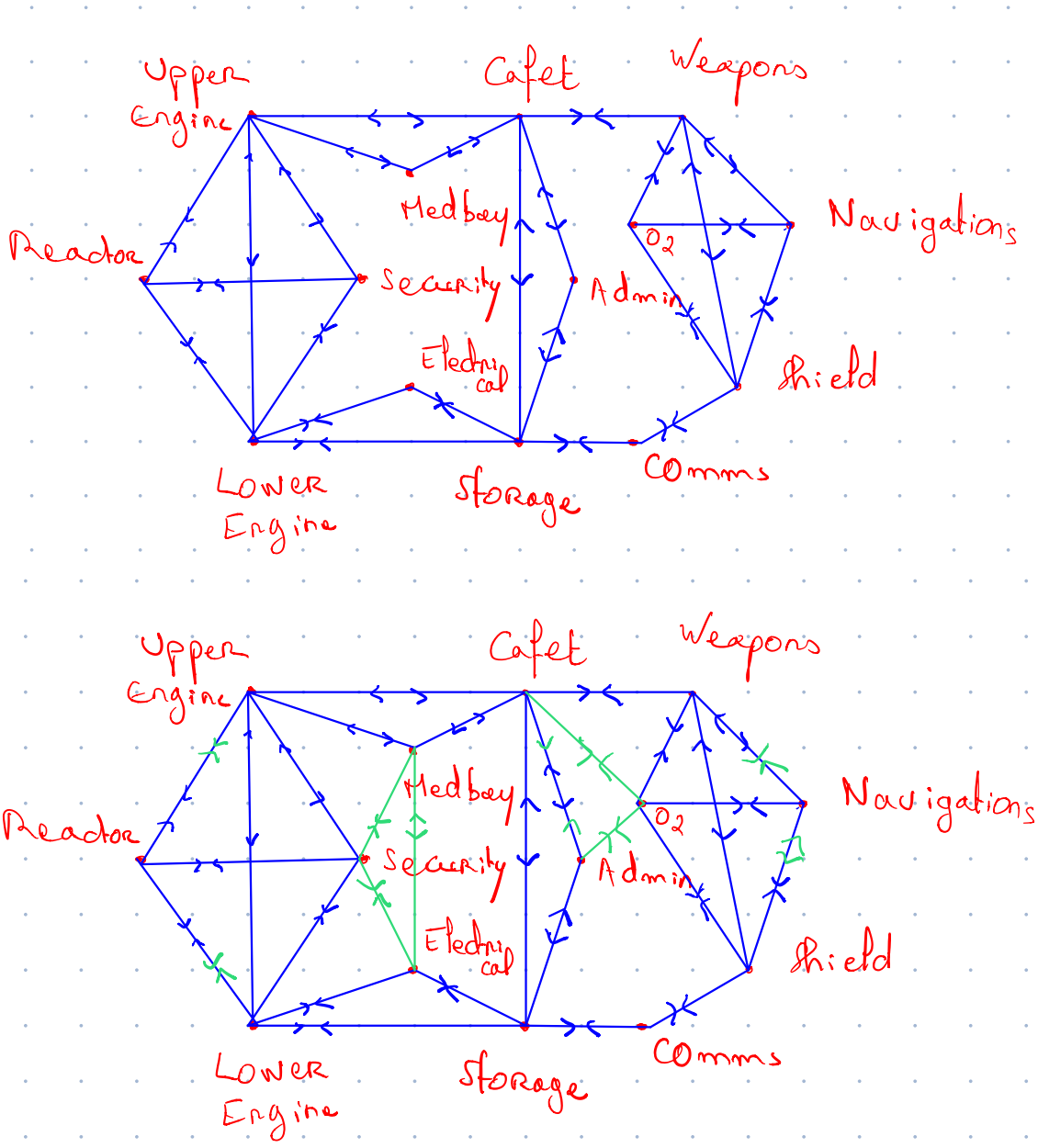
Crewmate map

```
In [140]: Load_Graph(path1)
Out[140]:
[['Reactor', 'Upper', '2'],
 ['Reactor', 'Security', '2'],
 ['Reactor', 'Lower', '2'],
 ['Upper', 'Reactor', '2'],
 ['Upper', 'Security', '2'],
 ['Upper', 'Lower', '4'],
 ['Upper', 'Medbay', '3'],
 ['Upper', 'Cafet', '4'],
 ['Security', 'Upper', '2'],
 ['Security', 'Reactor', '2'],
 ['Security', 'Lower', '2'],
 ['Lower', 'Security', '2'],
 ['Lower', 'Reactor', '2'],
 ['Lower', 'Upper', '4'],
 ['Lower', 'Electrical', '3'],
 ['Lower', 'Storage', '5'],
 ['Electrical', 'Lower', '3'],
 ['Electrical', 'Storage', '2'],
 ['Storage', 'Electrical', '2'],
 ['Storage', 'Lower', '5'],
 ['Storage', 'Admin', '1'],
 ['Storage', 'Cafet', '2'],
 ['Storage', 'Comms', '2'],
 ['Storage', 'Shield', '3'],
 ['Admin', 'Cafet', '1'],
 ['Admin', 'Storage', '1'],
 ['Comms', 'Storage', '2'],
 ['Comms', 'Shield', '1'],
 ['Shield', 'Storage', '3'],
 ['Shield', 'Comms', '1'],
 ['Shield', 'O2', '5'],
 ['Shield', 'Navigations', '4'],
 ['Shield', 'Weapons', '5'],
 ['Navigations', 'Shield', '4'],
 ['Navigations', 'O2', '2'],
 ['Navigations', 'Weapons', '2'],
 ['O2', 'Navigations', '2'],
 ['O2', 'Shield', '5'],
 ['O2', 'Weapons', '1'],
 ['Weapons', 'Shield', '5'],
 ['Weapons', 'Navigations', '2'],
 ['Weapons', 'O2', '1'],
 ['Weapons', 'Cafet', '1'],
 ['Cafet', 'Weapons', '1'],
 ['Cafet', 'Medbay', '1'],
 ['Cafet', 'Upper', '4'],
 ['Cafet', 'Admin', '2'],
 ['Cafet', 'Storage', '2'],
 ['Medbay', 'Cafet', '1'],
 ['Medbay', 'Upper', '3']]

In [141]:
```

Imposter map

```
In [142]: Load_Graph(path2)
Out[142]:
[['Reactor', 'Upper', '2'],
 ['Reactor', 'Security', '2'],
 ['Reactor', 'Lower', '2'],
 ['Upper', 'Reactor', '2'],
 ['Upper', 'Security', '2'],
 ['Upper', 'Lower', '4'],
 ['Upper', 'Medbay', '3'],
 ['Upper', 'Cafet', '4'],
 ['Security', 'Upper', '2'],
 ['Security', 'Reactor', '2'],
 ['Security', 'Lower', '2'],
 ['Lower', 'Security', '2'],
 ['Lower', 'Reactor', '2'],
 ['Lower', 'Upper', '4'],
 ['Lower', 'Electrical', '3'],
 ['Lower', 'Storage', '5'],
 ['Electrical', 'Lower', '3'],
 ['Electrical', 'Storage', '2'],
 ['Storage', 'Electrical', '2'],
 ['Storage', 'Lower', '5'],
 ['Storage', 'Admin', '1'],
 ['Storage', 'Cafet', '2'],
 ['Storage', 'Comms', '2'],
 ['Storage', 'Shield', '3'],
 ['Admin', 'Cafet', '1'],
 ['Admin', 'Storage', '1'],
 ['Comms', 'Storage', '2'],
 ['Comms', 'Shield', '1'],
 ['Shield', 'Storage', '3'],
 ['Shield', 'Comms', '1'],
 ['Shield', 'O2', '5'],
 ['Shield', 'Navigations', '4'],
 ['Shield', 'Weapons', '5'],
 ['Navigations', 'Shield', '4'],
 ['Navigations', 'O2', '2'],
 ['Navigations', 'Weapons', '2'],
 ['O2', 'Navigations', '2'],
 ['O2', 'Shield', '5'],
 ['O2', 'Weapons', '1'],
 ['Weapons', 'Shield', '5'],
 ['Weapons', 'Navigations', '2'],
 ['Weapons', 'O2', '1'],
 ['Weapons', 'Cafet', '1'],
 ['Cafet', 'Weapons', '1'],
 ['Cafet', 'Medbay', '1'],
 ['Cafet', 'Upper', '4'],
 ['Cafet', 'Admin', '2'],
 ['Cafet', 'Storage', '2'],
 ['Medbay', 'Cafet', '1'],
 ['Medbay', 'Upper', '3'],
 ['Upper', 'Reactor', '0'],
 ['Reactor', 'Upper', '0'],
 ['Reactor', 'Lower', '0'],
 ['Lower', 'Reactor', '0'],
 ['Security', 'Medbay', '0'],
 ['Security', 'Electrical', '0'],
 ['Electrical', 'Security', '0'],
 ['Electrical', 'Medbay', '0'],
 ['Medbay', 'Electrical', '0'],
 ['Medbay', 'Security', '0'],
 ['Shield', 'Navigations', '0'],
 ['Navigations', 'Shield', '0'],
 ['Navigations', 'Weapons', '0'],
 ['Weapons', 'Navigations', '0'],
 ['O2', 'Cafet', '0'],
 ['Cafet', 'O2', '0'],
 ['Admin', 'O2', '0'],
 ['O2', 'Admin', '0'],
 ['Cafet', 'Admin', '0'],
 ['Admin', 'Cafet', '0']]

In [143]:
```

This is our two models for the maps (first one for crewmates and second one for imposters), vent travel options are in green.



Then, since the model has been explained and all the datas have been well prepared, we can load it into Python with the same process as we use in the second Part. The dictionaries will have this shapes :

map = {room : {connect_room : time_totravel}

So here we have our 2 maps :

Crewmate map :

```
In [145]: map1
Out[145]:
{'Admin': {'Cafet': 1, 'Storage': 1},
 'Cafet': {'Weapons': 1, 'Medbay': 1, 'Upper': 4, 'Admin': 2, 'Storage': 2},
 'Comms': {'Storage': 2, 'Shield': 1},
 'Electrical': {'Lower': 3, 'Storage': 2},
 'Lower': {'Security': 2,
  'Reactor': 2,
  'Upper': 4,
  'Electrical': 3,
  'Storage': 5},
 'Medbay': {'Cafet': 1, 'Upper': 3},
 'Navigations': {'Shield': 4, 'O2': 2, 'Weapons': 2},
 'O2': {'Navigations': 2, 'Shield': 5, 'Weapons': 1},
 'Reactor': {'Upper': 2, 'Security': 2, 'Lower': 2},
 'Security': {'Upper': 2, 'Reactor': 2, 'Lower': 2},
 'Shield': {'Storage': 3, 'Comms': 1, 'O2': 5, 'Navigations': 4, 'Weapons': 5},
 'Storage': {'Electrical': 2,
  'Lower': 5,
  'Admin': 1,
  'Cafet': 2,
  'Comms': 2,
  'Shield': 3},
 'Upper': {'Reactor': 2, 'Security': 2, 'Lower': 4, 'Medbay': 3, 'Cafet': 4},
 'Weapons': {'Shield': 5, 'Navigations': 2, 'O2': 1, 'Cafet': 1}}

In [146]:
```

Imposter map :

```
In [147]: map2
Out[147]:
{'Admin': {'Cafet': 0, 'Storage': 1, 'O2': 0},
 'Cafet': {'Weapons': 1,
  'Medbay': 1,
  'Upper': 4,
  'Admin': 0,
  'Storage': 2,
  'O2': 0},
 'Comms': {'Storage': 2, 'Shield': 1},
 'Electrical': {'Lower': 3, 'Storage': 2, 'Security': 0, 'Medbay': 0},
 'Lower': {'Security': 2,
  'Reactor': 0,
  'Upper': 4,
  'Electrical': 3,
  'Storage': 5},
 'Medbay': {'Cafet': 1, 'Upper': 3, 'Electrical': 0, 'Security': 0},
 'Navigations': {'Shield': 0, 'O2': 2, 'Weapons': 0},
 'O2': {'Navigations': 2, 'Shield': 5, 'Weapons': 1, 'Cafet': 0, 'Admin': 0},
 'Reactor': {'Upper': 0, 'Security': 2, 'Lower': 0},
 'Security': {'Upper': 2,
  'Reactor': 2,
  'Lower': 2,
  'Medbay': 0,
  'Electrical': 0},
 'Shield': {'Storage': 3, 'Comms': 1, 'O2': 5, 'Navigations': 0, 'Weapons': 5},
 'Storage': {'Electrical': 2,
  'Lower': 5,
  'Admin': 1,
  'Cafet': 2,
  'Comms': 2,
  'Shield': 3},
 'Upper': {'Reactor': 0, 'Security': 2, 'Lower': 4, 'Medbay': 3, 'Cafet': 4},
 'Weapons': {'Shield': 5, 'Navigations': 0, 'O2': 1, 'Cafet': 1}}

In [148]:
```

The pathfinding algorithm to solve our problem : Dijkstra

Here we want to find to most efficient path (in terms of time to travel) to travel between two rooms and lately between any pair of rooms. Dijkstra seems to be the wright algorithm to choose for solving this problem in our context. So by applying this method for the two maps, we will be able to see each time of travel and also calculate the time difference for an imposter and a crewmate.

To display the shortest path between every pair of rooms I've created two algorithms: Dijkstra algorithm which find the shortest path for two rooms and TimeToTravel using the previous one for all the combinations pairs possible. So we have the output here:

```
In [149]: TimeToTravel(map1)
Out[149]:
[('Admin', 'Cafet', 1),
 ('Admin', 'Comms', 3),
 ('Admin', 'Electrical', 3),
 ('Admin', 'Lower', 6),
 ('Admin', 'Medbay', 2),
 ('Admin', 'Navigations', 4),
 ('Admin', 'O2', 3),
 ('Admin', 'Reactor', 7),
 ('Admin', 'Security', 7),
 ('Admin', 'Shield', 4),
 ('Admin', 'Storage', 1),
 ('Admin', 'Upper', 5),
 ('Admin', 'Weapons', 2),
 ('Cafet', 'Comms', 4),
 ('Cafet', 'Electrical', 4),
 ('Cafet', 'Lower', 7),
 ('Cafet', 'Medbay', 1),
 ('Cafet', 'Navigations', 3),
 ('Cafet', 'O2', 2),
 ('Cafet', 'Reactor', 6),
 ('Cafet', 'Security', 6),
 ('Cafet', 'Shield', 5),
 ('Cafet', 'Storage', 2),
 ('Cafet', 'Upper', 4),
 ('Cafet', 'Weapons', 1),
 ('Comms', 'Electrical', 4),
 ('Comms', 'Lower', 7),
 ('Comms', 'Medbay', 5),
 ('Comms', 'Navigations', 5),
 ('Comms', 'O2', 6),
 ('Comms', 'Reactor', 9),
 ('Comms', 'Security', 9),
 ('Comms', 'Shield', 1),
 ('Comms', 'Storage', 2),
 ('Comms', 'Upper', 8),
 ('Comms', 'Weapons', 5),
 ('Electrical', 'Lower', 3),
 ('Electrical', 'Medbay', 5),
 ('Electrical', 'Navigations', 7),
 ('Electrical', 'O2', 6),
 ('Electrical', 'Reactor', 5),
 ('Electrical', 'Security', 5),
 ('Electrical', 'Shield', 5),
 ('Electrical', 'Storage', 2),
 ('Electrical', 'Upper', 7),
```

Map_Crewmate

```
 ('Electrical', 'Upper', 7),
 ('Electrical', 'Weapons', 5),
 ('Lower', 'Medbay', 7),
 ('Lower', 'Navigations', 10),
 ('Lower', 'O2', 9),
 ('Lower', 'Reactor', 2),
 ('Lower', 'Security', 2),
 ('Lower', 'Shield', 8),
 ('Lower', 'Storage', 5),
 ('Lower', 'Upper', 4),
 ('Lower', 'Weapons', 8),
 ('Medbay', 'Navigations', 4),
 ('Medbay', 'O2', 3),
 ('Medbay', 'Reactor', 5),
 ('Medbay', 'Security', 5),
 ('Medbay', 'Shield', 6),
 ('Medbay', 'Storage', 3),
 ('Medbay', 'Upper', 3),
 ('Medbay', 'Weapons', 2),
 ('Navigations', 'O2', 2),
 ('Navigations', 'Reactor', 9),
 ('Navigations', 'Security', 9),
 ('Navigations', 'Shield', 4),
 ('Navigations', 'Storage', 5),
 ('Navigations', 'Upper', 7),
 ('Navigations', 'Weapons', 2),
 ('O2', 'Reactor', 8),
 ('O2', 'Security', 8),
 ('O2', 'Shield', 5),
 ('O2', 'Storage', 4),
 ('O2', 'Upper', 6),
 ('O2', 'Weapons', 1),
 ('Reactor', 'Security', 2),
 ('Reactor', 'Shield', 10),
 ('Reactor', 'Storage', 7),
 ('Reactor', 'Upper', 2),
 ('Reactor', 'Weapons', 7),
 ('Security', 'Shield', 10),
 ('Security', 'Storage', 7),
 ('Security', 'Upper', 2),
 ('Security', 'Weapons', 7),
 ('Shield', 'Storage', 3),
 ('Shield', 'Upper', 9),
 ('Shield', 'Weapons', 5),
 ('Storage', 'Upper', 6),
 ('Storage', 'Weapons', 3),
 ('Upper', 'Weapons', 5)]

In [150]:
```

```
In [151]: TimeToTravel(map2)
Out[151]:
[('Admin', 'Cafet', 0),
 ('Admin', 'Comms', 2),
 ('Admin', 'Electrical', 1),
 ('Admin', 'Lower', 3),
 ('Admin', 'Medbay', 1),
 ('Admin', 'Navigations', 1),
 ('Admin', 'O2', 0),
 ('Admin', 'Reactor', 3),
 ('Admin', 'Security', 1),
 ('Admin', 'Shield', 1),
 ('Admin', 'Storage', 1),
 ('Admin', 'Upper', 3),
 ('Admin', 'Weapons', 1),
 ('Cafet', 'Comms', 2),
 ('Cafet', 'Electrical', 1),
 ('Cafet', 'Lower', 3),
 ('Cafet', 'Medbay', 1),
 ('Cafet', 'Navigations', 1),
 ('Cafet', 'O2', 0),
 ('Cafet', 'Reactor', 3),
 ('Cafet', 'Security', 1),
 ('Cafet', 'Shield', 1),
 ('Cafet', 'Storage', 1),
 ('Cafet', 'Upper', 3),
 ('Cafet', 'Weapons', 1),
 ('Comms', 'Electrical', 3),
 ('Comms', 'Lower', 5),
 ('Comms', 'Medbay', 3),
 ('Comms', 'Navigations', 1),
 ('Comms', 'O2', 2),
 ('Comms', 'Reactor', 5),
 ('Comms', 'Security', 3),
 ('Comms', 'Shield', 1),
 ('Comms', 'Storage', 2),
 ('Comms', 'Upper', 5),
 ('Comms', 'Weapons', 1),
 ('Electrical', 'Lower', 2),
 ('Electrical', 'Medbay', 0),
 ('Electrical', 'Navigations', 2),
 ('Electrical', 'O2', 1),
 ('Electrical', 'Reactor', 2),
 ('Electrical', 'Security', 0),
 ('Electrical', 'Shield', 2),
 ('Electrical', 'Storage', 2),
 ('Electrical', 'Upper', 2),
 ('Electrical', 'Weapons', 2),
```

Map_Imposter

```
 ('Electrical', 'Weapons', 2),
 ('Lower', 'Medbay', 2),
 ('Lower', 'Navigations', 4),
 ('Lower', 'O2', 3),
 ('Lower', 'Reactor', 0),
 ('Lower', 'Security', 2),
 ('Lower', 'Shield', 4),
 ('Lower', 'Storage', 4),
 ('Lower', 'Upper', 0),
 ('Lower', 'Weapons', 4),
 ('Medbay', 'Navigations', 2),
 ('Medbay', 'O2', 1),
 ('Medbay', 'Reactor', 2),
 ('Medbay', 'Security', 0),
 ('Medbay', 'Shield', 2),
 ('Medbay', 'Storage', 2),
 ('Medbay', 'Upper', 2),
 ('Medbay', 'Weapons', 2),
 ('Navigations', 'O2', 1),
 ('Navigations', 'Reactor', 4),
 ('Navigations', 'Security', 2),
 ('Navigations', 'Shield', 0),
 ('Navigations', 'Storage', 2),
 ('Navigations', 'Upper', 4),
 ('Navigations', 'Weapons', 0),
 ('O2', 'Reactor', 3),
 ('O2', 'Security', 1),
 ('O2', 'Shield', 1),
 ('O2', 'Storage', 1),
 ('O2', 'Upper', 3),
 ('O2', 'Weapons', 1),
 ('Reactor', 'Security', 2),
 ('Reactor', 'Shield', 4),
 ('Reactor', 'Storage', 4),
 ('Reactor', 'Upper', 0),
 ('Reactor', 'Weapons', 4),
 ('Security', 'Shield', 2),
 ('Security', 'Storage', 2),
 ('Security', 'Upper', 2),
 ('Security', 'Weapons', 2),
 ('Shield', 'Storage', 2),
 ('Shield', 'Upper', 4),
 ('Shield', 'Weapons', 0),
 ('Storage', 'Upper', 4),
 ('Storage', 'Weapons', 2),
 ('Upper', 'Weapons', 4)]

In [152]:
```

Now last step is to display the time differences between the previous ones. The method TimeDifference is used to do this. It uses the previous method applying on both maps and display the difference of travelling time.

Output of TimeDifference:

```
In [155]: TimeDifference(map1,map2)
 difference of time to travel between Admin and Cafet : 1 secondes
 difference of time to travel between Admin and Comms : 1 secondes
 difference of time to travel between Admin and Electrical : 2 secondes
 difference of time to travel between Admin and Lower : 3 secondes
 difference of time to travel between Admin and Medbay : 1 secondes
 difference of time to travel between Admin and Navigations : 3 secondes
 difference of time to travel between Admin and O2 : 3 secondes
 difference of time to travel between Admin and Reactor : 4 secondes
 difference of time to travel between Admin and Security : 6 secondes
 difference of time to travel between Admin and Shield : 3 secondes
 difference of time to travel between Admin and Storage : 0 secondes
 difference of time to travel between Admin and Upper : 2 secondes
 difference of time to travel between Admin and Weapons : 1 secondes
 difference of time to travel between Cafet and Comms : 2 secondes
 difference of time to travel between Cafet and Electrical : 3 secondes
 difference of time to travel between Cafet and Lower : 4 secondes
 difference of time to travel between Cafet and Medbay : 0 secondes
 difference of time to travel between Cafet and Navigations : 2 secondes
 difference of time to travel between Cafet and O2 : 2 secondes
 difference of time to travel between Cafet and Reactor : 3 secondes
 difference of time to travel between Cafet and Security : 5 secondes
 difference of time to travel between Cafet and Shield : 4 secondes
 difference of time to travel between Cafet and Storage : 1 secondes
 difference of time to travel between Cafet and Upper : 1 secondes
 difference of time to travel between Cafet and Weapons : 0 secondes
 difference of time to travel between Comms and Electrical : 1 secondes
 difference of time to travel between Comms and Lower : 2 secondes
 difference of time to travel between Comms and Medbay : 2 secondes
 difference of time to travel between Comms and Navigations : 4 secondes
 difference of time to travel between Comms and O2 : 4 secondes
 difference of time to travel between Comms and Reactor : 4 secondes
 difference of time to travel between Comms and Security : 6 secondes
 difference of time to travel between Comms and Shield : 0 secondes
 difference of time to travel between Comms and Storage : 0 secondes
 difference of time to travel between Comms and Upper : 3 secondes
 difference of time to travel between Comms and Weapons : 4 secondes
 difference of time to travel between Electrical and Lower : 1 secondes
 difference of time to travel between Electrical and Medbay : 5 secondes
 difference of time to travel between Electrical and Navigations : 5 secondes
 difference of time to travel between Electrical and O2 : 5 secondes
 difference of time to travel between Electrical and Reactor : 3 secondes
 difference of time to travel between Electrical and Security : 5 secondes
 difference of time to travel between Electrical and Shield : 3 secondes
 difference of time to travel between Electrical and Storage : 0 secondes
 difference of time to travel between Electrical and Upper : 5 secondes
 difference of time to travel between Electrical and Weapons : 3 secondes
 difference of time to travel between Lower and Medbay : 5 secondes
 difference of time to travel between Lower and Navigations : 6 secondes
 difference of time to travel between Lower and O2 : 6 secondes
 difference of time to travel between Lower and Reactor : 2 secondes
 difference of time to travel between Lower and Security : 0 secondes
 difference of time to travel between Lower and Shield : 4 secondes
 difference of time to travel between Lower and Storage : 1 secondes
 difference of time to travel between Lower and Upper : 4 secondes
 difference of time to travel between Lower and Weapons : 4 secondes
 difference of time to travel between Medbay and Navigations : 2 secondes
 difference of time to travel between Medbay and O2 : 2 secondes
 difference of time to travel between Medbay and Reactor : 3 secondes
 difference of time to travel between Medbay and Security : 5 secondes
 difference of time to travel between Medbay and Shield : 4 secondes
 difference of time to travel between Medbay and Storage : 1 secondes
 difference of time to travel between Medbay and Upper : 1 secondes
 difference of time to travel between Medbay and Weapons : 0 secondes
 difference of time to travel between Navigations and O2 : 1 secondes
 difference of time to travel between Navigations and Reactor : 5 secondes
 difference of time to travel between Navigations and Security : 7 secondes
 difference of time to travel between Navigations and Shield : 4 secondes
 difference of time to travel between Navigations and Storage : 3 secondes
 difference of time to travel between Navigations and Upper : 3 secondes
 difference of time to travel between Navigations and Weapons : 2 secondes
 difference of time to travel between O2 and Reactor : 5 secondes
 difference of time to travel between O2 and Security : 7 secondes
 difference of time to travel between O2 and Shield : 4 secondes
 difference of time to travel between O2 and Storage : 3 secondes
 difference of time to travel between O2 and Upper : 3 secondes
 difference of time to travel between O2 and Weapons : 0 secondes
 difference of time to travel between Reactor and Security : 0 secondes
 difference of time to travel between Reactor and Shield : 6 secondes
 difference of time to travel between Reactor and Storage : 3 secondes
 difference of time to travel between Reactor and Upper : 2 secondes
 difference of time to travel between Reactor and Weapons : 3 secondes
 difference of time to travel between Security and Shield : 8 secondes
 difference of time to travel between Security and Storage : 5 secondes
 difference of time to travel between Security and Upper : 0 secondes
 difference of time to travel between Security and Weapons : 5 secondes
 difference of time to travel between Shield and Storage : 1 secondes
 difference of time to travel between Shield and Upper : 5 secondes
 difference of time to travel between Shield and Weapons : 5 secondes
 difference of time to travel between Storage and Upper : 2 secondes
 difference of time to travel between Storage and Weapons : 1 secondes
 difference of time to travel between Upper and Weapons : 1 secondes

In [156]:
```

PART4:
Secure the last tasks

Files involved in this part: Part4.py, Module2.py, map_crewmate.txt.

In this last part, we will discuss about crewmates' tasks they need to finish to win the game. The safest way to finish tasks is to group and walk across the map so that imposters won't kill anyone (too risky). In this matter of fact, crewmates decided to walk through the map and to visit only a room once. The model of the map I used is the same as the one in the previous part for crewmates.

The algorithm:

To find a route passing through each room only once, I though about using Hamilton Paths. Because of our map is a graph, we can try to find Hamilton paths to solve our problem.

By trying to do it on paper, I've figured out that some rooms in the map couldn't be starting point for finding Hamilton paths : Cafet, Lower Engine, Reactor, Security, Shield, Storage, Upper.

Firstly I've created the Graph class for this algorithm. Its role is simple: create an adjacency matrix in the shape of a data frame like the following one below.

```
In [192]: test.adj_matr
Out[192]:
             Admin  Cafet  Comms  Electrical  ...  Shield  Storage  Upper  Weapons
Admin            0      1      0           0  ...       0        1      0        0
Cafet            1      0      0           0  ...       0        1      1        1
Comms            0      0      0           0  ...       1        1      0        0
Electrical       0      0      0           0  ...       0        1      0        0
Lower            0      0      0           1  ...       0        1      1        0
Medbay           0      1      0           0  ...       0        0      1        0
Navigations      0      0      0           0  ...       1        0      0        1
O2               0      0      0           0  ...       1        0      0        1
Reactor          0      0      0           0  ...       0        0      1        0
Security         0      0      0           0  ...       0        0      1        0
Shield           0      0      1           0  ...       0        1      0        1
Storage          1      1      1           1  ...       1        0      0        0
Upper            0      1      0           0  ...       0        0      0        0
Weapons          0      1      0           0  ...       1        0      0        0

[14 rows x 14 columns]

In [193]:
```

After that I used 2 algorithms to find solutions: HamiltonianPaths and Hamilton_path_source. The first one solves the Hamilton path problem by using recursion and backtracking. The second one asks user to input a room, initialises variables for HamiltonianPaths method and display all the Hamilton paths with the previous input.

Here is a possible output calling Hamilton_path_source and picking Navigations as a starting point:

```
In [204]: runfile('/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced Data/Project/Part4.py', wdir='/Users/Arthur/Desktop/esilv/A4_DIA/S1/Advanced
Data/Project')
Reloaded modules: Module2
Choose your sources among the ones below


['Admin', 'Cafet', 'Comms', 'Electrical', 'Lower', 'Medbay', 'Navigations', 'O2', 'Reactor', 'Security', 'Shield', 'Storage', 'Upper',
'Weapons']


Choice ?

Navigations
['Navigations', 'O2', 'Weapons', 'Shield', 'Comms', 'Storage', 'Admin', 'Cafet', 'Medbay', 'Upper', 'Reactor', 'Security', 'Lower',
'Electrical']
['Navigations', 'O2', 'Weapons', 'Shield', 'Comms', 'Storage', 'Admin', 'Cafet', 'Medbay', 'Upper', 'Security', 'Reactor', 'Lower',
'Electrical']
['Navigations', 'O2', 'Weapons', 'Shield', 'Comms', 'Storage', 'Electrical', 'Lower', 'Reactor', 'Security', 'Upper', 'Medbay', 'Cafet',
'Admin']
['Navigations', 'O2', 'Weapons', 'Shield', 'Comms', 'Storage', 'Electrical', 'Lower', 'Security', 'Reactor', 'Upper', 'Medbay', 'Cafet',
'Admin']
['Navigations', 'Weapons', 'O2', 'Shield', 'Comms', 'Storage', 'Admin', 'Cafet', 'Medbay', 'Upper', 'Reactor', 'Security', 'Lower',
'Electrical']
['Navigations', 'Weapons', 'O2', 'Shield', 'Comms', 'Storage', 'Admin', 'Cafet', 'Medbay', 'Upper', 'Security', 'Reactor', 'Lower',
'Electrical']
['Navigations', 'Weapons', 'O2', 'Shield', 'Comms', 'Storage', 'Electrical', 'Lower', 'Reactor', 'Security', 'Upper', 'Medbay', 'Cafet',
'Admin']
['Navigations', 'Weapons', 'O2', 'Shield', 'Comms', 'Storage', 'Electrical', 'Lower', 'Security', 'Reactor', 'Upper', 'Medbay', 'Cafet',
'Admin']

In [205]:
```