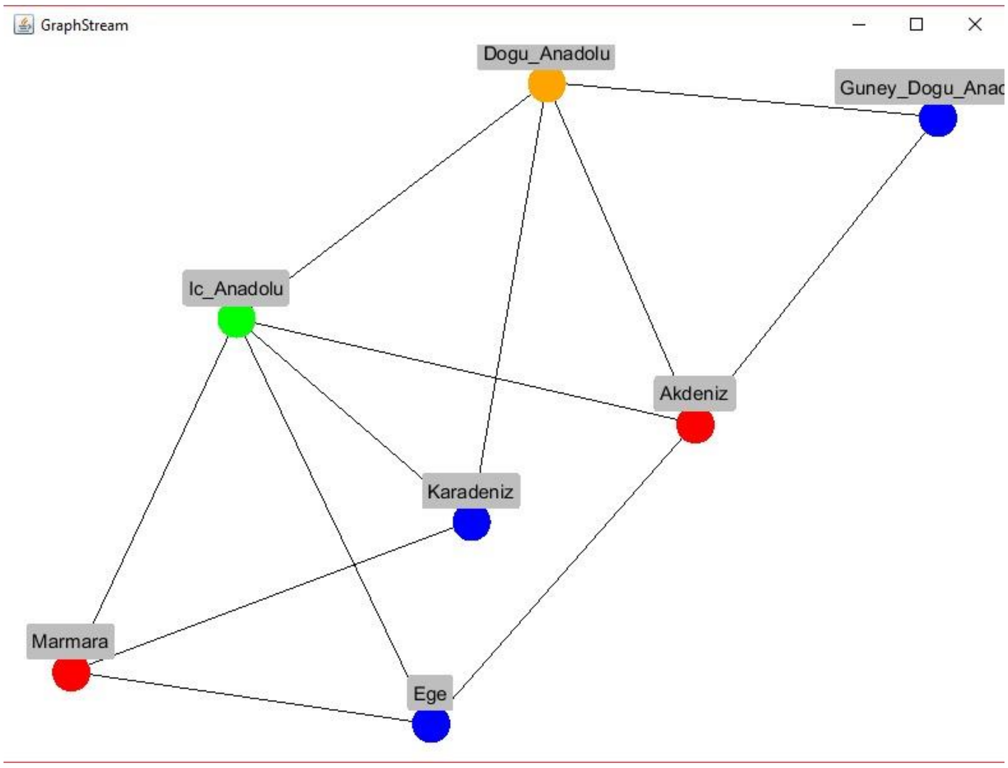
Constraint Logic Programming Report
Map colouring problem

We decided to create a program able to solve a map-colouring problem and to generate a visual representation of the solution.
A map colouring problem consist of having a map of cities where each city is link to one or few others. We want to allocate a colour to each city so that each city has a different colour from its neighbour, but by using the minimum number of colours.

To do so, we used the JacoP library to model and solve the problem and we used the GraphStream library for the graphical part.



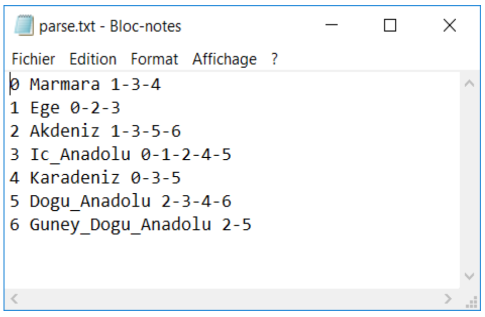
Classes and Functions

City

It was useful to create a class for making our own 'datatype'
Class includes elements such as: id(int), name(string), neighbours(arraylist) and a constructor.

File Adapter

We decided to store our data in a .txt file so that it would be dynamic depending of user and easy to adapt. Two example are included in the project: *parse.txt* and *parseCities.txt*.


parse.txt - Bloc-notes

```
0 Marmara 1-3-4
1 Ege 0-2-3
2 Akdeniz 1-3-5-6
3 Ic_Anadolu 0-1-2-4-5
4 Karadeniz 0-3-5
5 Dogu_Anadolu 2-3-4-6
6 Guney_Dogu_Anadolu 2-5
```

Each line represents a city. The first element is its *ID*, the second one is the *Name*, and the last elements (numbers separated by “-“) are the IDs of the neighbours of the city.
To import your own file, you can basically name it *parse.txt* or you can give your filename as an argument.

This file is read with the help of the *FileAdapter* object. This object opens the *txt* file, read it line by line and isolate the different data in an array with the help of ‘Parse’ function.
The function returns the *City* array.

Relations

Class designed for create relations between the two libraries that we used ‘Jacop’ and ‘GraphStream’. It has a property of ‘City’ array which stores cities that we want to create relations between cities with the help of ‘initCities’ function.

We created ‘XneqY’ constraints dynamically from our user file input depending of the cities and its neighbours that we previously defined with the city class. From the result that we obtain from jacop (as integer values) we matched them into colors and define every node color dynamically via css (feature of the ‘GraphStream’ library).

We visualized every node as city and ‘neighbour’ relation between them as edge. We are gathering result from jacop and creating nodes and edges as ‘GraphStream’ wants in order to clearly visualize our problem.

Conclusions

This project allowed us to work on a concrete example of Constraint Logic Programming with the use of the JacoP library.
We gave a particular attention to the representation of our problem, we wanted it to be very clear and to be easily modifiable for our user. In that purpose, we decided to use the GraphStream library which was adequate with our goal.
This is also why we made the modelization of our problem dynamic to make it more user-friendly.