Rapport

1) Visualisation du Graphe

La méthode d'instance VisualiserGraphe de la classe GraphMetro et Graphe Utilisateur n'a pas été grandement modifiée depuis le rendu 1. Elle utilise les bibliothèques System.Drawing et Microsoft.Forms. Les prompts pour la réaliser était :

"Voici ma classe graphe en utilisant la bibliothèque Microsoft.Forms, aide moi à visualiser ce graphe, on doit pouvoir ouvrir une fenêtre afficher l'ensemble des noeuds avec leurs identifiants et leurs liens"

"Peux tu améliorer ce code pour que les noeuds ne se touchent pas et soit espacé d'une distance minimale"

2) Partie Interface

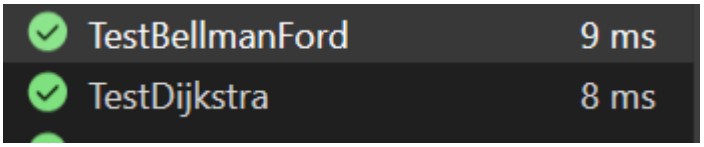
La partie interface à été réalisée avec la bibliothèque Windows Forms, pour avoir accès au code on peut cliquer droit sur un fichier et faire **"Afficher le code".** Chaque fichier comporte une classe comportant la page.

3) Choix de l'algorithme de plus court chemin

Après plusieurs Tests nous avons décidé de prendre l'algorithme de plus court chemin "**Dijkstra**". Nous avons fait le choix car nous avons pas réussi à intégrer les temps de changements dans celui de Floyd Warshall de plus son temps d'exécution est considérablement plus élevé.

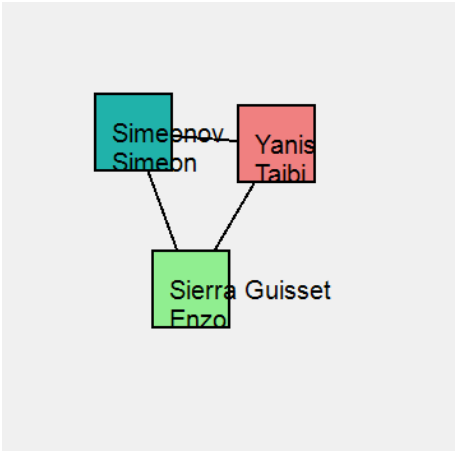
Entre *BellmanFord* et *Djisktra* le temps d'exécution est très similaire, de plus les résultats obtenus aussi. Le chemin le plus court trouvé est identique dans les stations de départ et d'arrivée que nous avons testé.

Nous avons choisi Djisktra au final car notre graphe ne possède pas de poids négatifs donc c'était plus adapté.

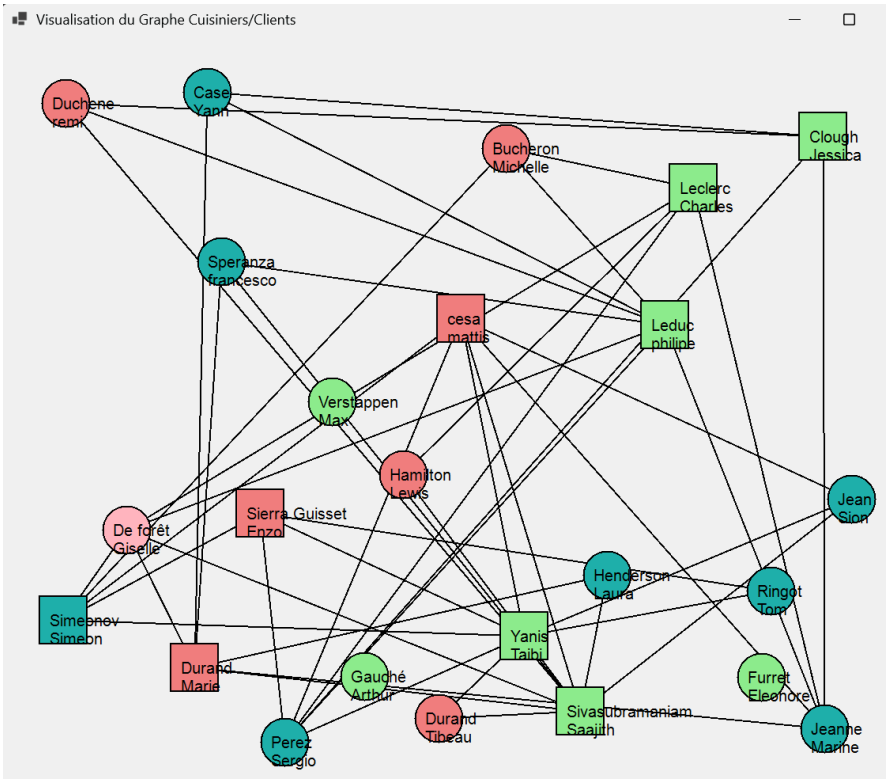


4) Coloration du graphe

Après avoir appliqué **l'algorithme de Welsh-Powell**, nous obtenons un nombre **minimal de couleurs de 4** pour les utilisateurs qu'on possède**.** Ce résultat peut évoluer en fonction des nouvelles commandes.
Notre graphe ayant un nombre minimal de quatres couleurs **il n'est donc pas biparti**.
En revanche, la coloration ne permet pas de savoir s'il est planaire ou pas. Mais en analysant graphiquement et en sachant qu'un client peut avoir 1,2 à n cuisiniers, et chaque cuisinier peut commander en tant que client à 1, 2, …, n-1 autres cuisiniers **très rapidement le graphe ne sera plus planaire.**



A trois utilisateurs interconnectés on obtient 3 couleurs.



Graphe actuel.

On remarque qu'il n'y pas de groupes indépendants tels que plusieurs clients partagent un seul cuisinier, **chaque client possède plusieurs cuisiniers**, en revanche on remarque que le client qui à commandé auprès du plus grand nombre de cuisiniers est **Giselle De forêt** du par sa couleur. On remarque aussi que **Furrer Eléonore** n'a pas encore fait de commandes.

5) Choix de faire deux graphes au lieu d'utiliser le type générique

En essayant d'adapter la classe *GrapheMetro* pour qu'elle puisse être générique, on a rencontré une erreur qu'on n'a pas su résoudre et qui a motivé notre choix de faire deux classes distinctes.

```
public class GrapheMetro<T>
{
    List<Noeud<T>> List_Noeuds;
```

```
public GrapheMetro(String chemin_fichier_stations, String chemin_fichier_arcs)
{
    // 1) Initialisation
    this.List_Noeuds = new List<Noeud<T>>();
```

```
public station_metro TrouverStationAvecNom (string nom_station)
{
    for (int i = 0; i < List_Noeuds.Count; i++)
    {
        if (List_Noeuds[i].Valeur.Nom == nom_station)
        {
            return List_Noeuds[i].Valeur;
        }
    }
    return null;
}
```

En effet, il était impossible d'accéder à des propriétés de la liste générique sans rencontrer de multiple erreurs qui obligèrent à modifier toute la classe. Pour éviter ça, on a pris la décision de faire deux classes distinctes.