ISEN
ALL IS DIGITAL!
LILLE

yncréa

# Projet JTrafic

CIR3

DEHOOGHE FLORENTIN
LANGUE LOUIS-CLEMENT
VANDEWALLE REMI

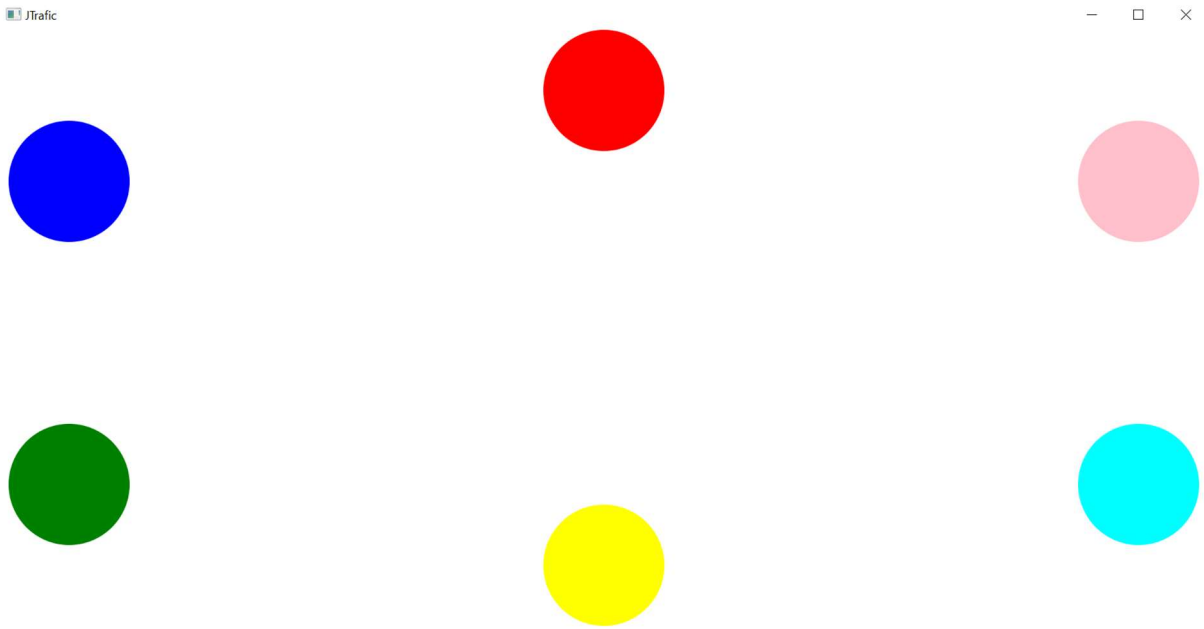
***On souhaite développer un simulateur de trafic routier. Le principe est de pouvoir simuler, sur un réseau routier donné, un certain nombre de véhicules dotés de vitesses variées. En fonction de la longueur des trajets, des feux, carrefours et intersections rencontrés. La finalité d'un tel projet est de pouvoir étudier la formation de bouchons ou de nœuds de trafic en fonction du trafic généré.***

Pour cela, nous avons tout d'abord réfléchi aux différents éléments dont nous aurions besoin. Il fallait pouvoir afficher des villes, des voitures, des routes et gérer des chemins et des intersections entre ces éléments. Nous avons trouvé, dès le départ, plus épuré de représenter les villes par des cercles, les voitures par des rectangles, et bien entendu les routes par des lignes. Nous n'avons pas voulu insérer d'images à proprement parlé dans notre simulateur.

Pour afficher tout cela, il nous fallait bien entendu un outil d'affichage. Nous avons dû nous familiariser avec la bibliothèque **JavaFx** pour appréhender l'interface graphique de notre projet.

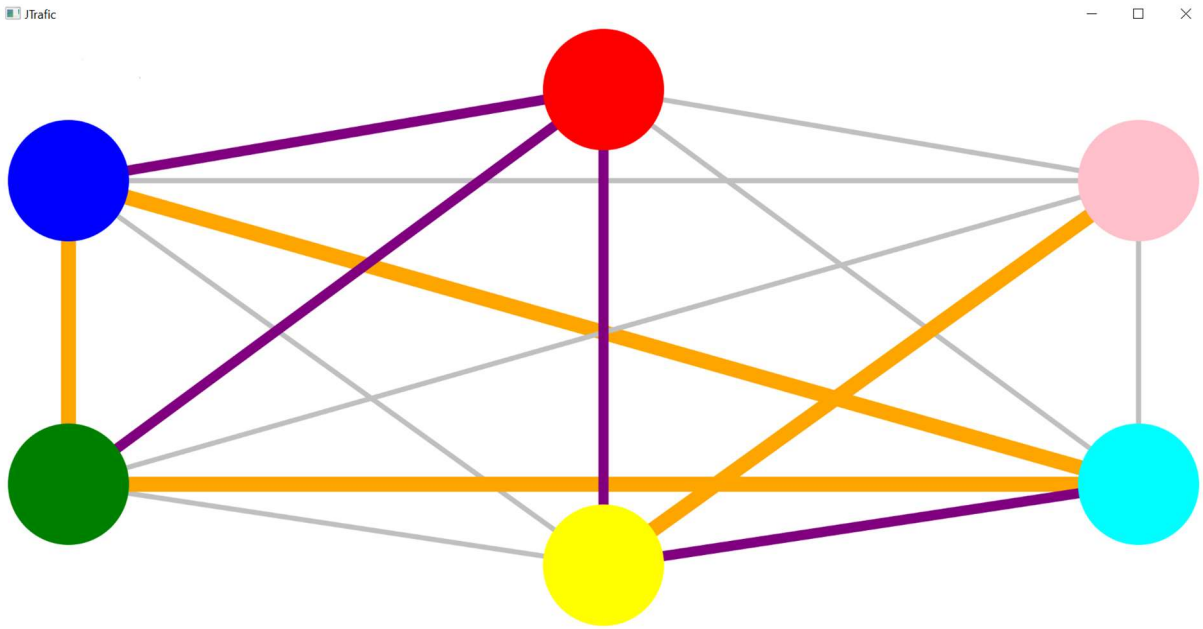

Après cette prise en main, nous avons pu créer notre **scène**, qui n'est ni plus ni moins notre fenêtre d'affichage, pour pouvoir y ajouter nos différents éléments. Nous avons également, en parallèle, ouvert un GIT afin de faciliter l'échange de fichiers et une conversation Messenger pour pouvoir discuter du projet.

Pour commencer, nous avons décidé de créer nos villes grâce à la classe **Circle** déjà existante en JavaFx.



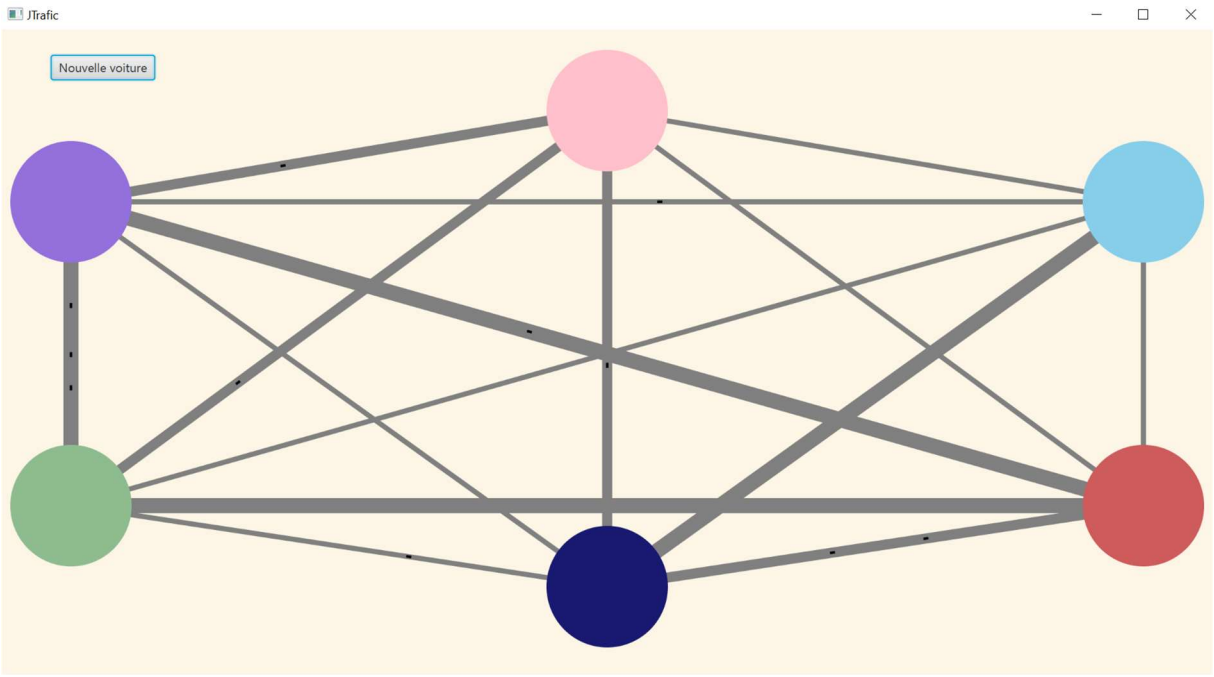
Dès lors que nos villes furent positionnées, nous avons pu gérer les déplacements entre elles. Pour cela, nous avons créé une fonction **Path (generatePathAB)** créant un chemin entre 2 villes. Nous avons ajouté à ça une animation **PathTransition (generatePathTransition)** qui nous permet de gérer les paramètres de ce chemin comme le temps de parcours, l'orientation, la périodicité…

Seulement, cela ne nous permet que de créer un chemin, mais pas de l'afficher. Pour cette partie nous avons donc dû utiliser notre **generateRouteAB** créé grâce à la classe **Line** dédié dans le JavaFx pour pouvoir cette fois-ci afficher au sens propre la route correspondant au chemin. Nous avons également créé une boucle qui permet de créer une route entre chaque ville et de choisir, plus ou moins aléatoirement, le type de route (autoroute, nationale, départementale) et de les afficher de différentes couleurs pour notre compréhension.

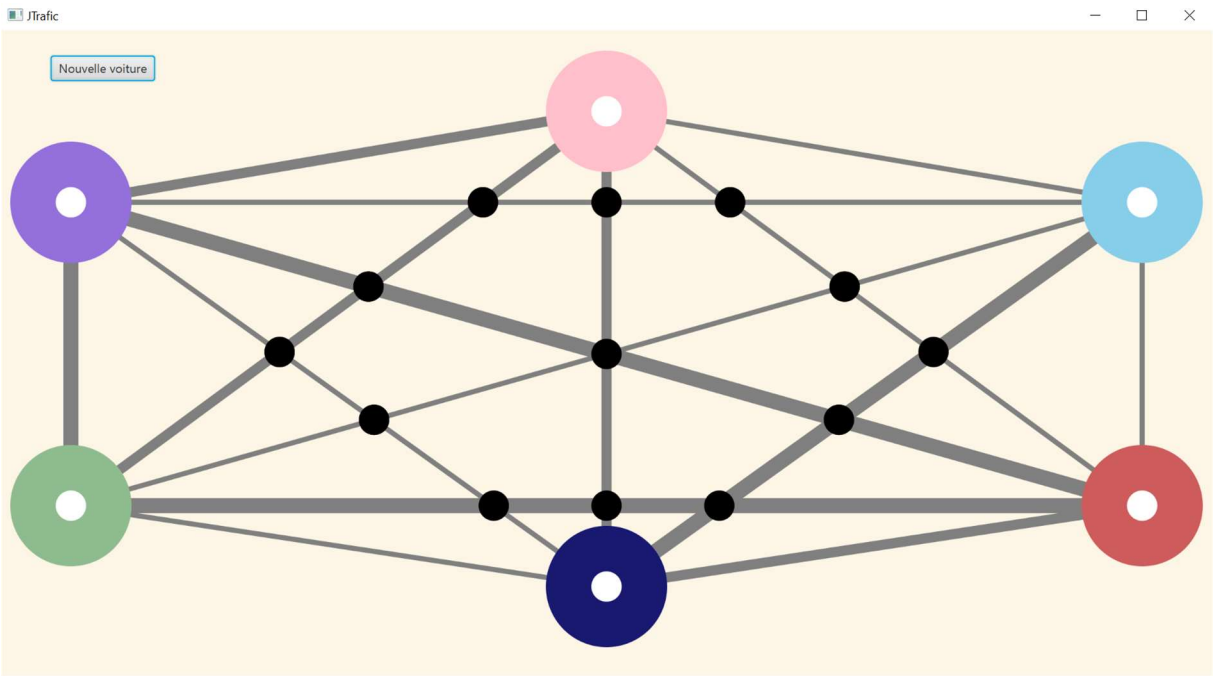



Cette base posée, il nous fallait maintenant traiter les voitures. Pour cela nous avons créé une classe **Voiture**, qui permet de les afficher grâce à la classe **Rectangle** déjà existante en JavaFx. Nous avons donc pu, dès lors, créer notre **generateCar** de type **Voiture** qui permet, en prenant en compte notre **PathTransition** de créer la voiture et de lui attribuer un chemin.

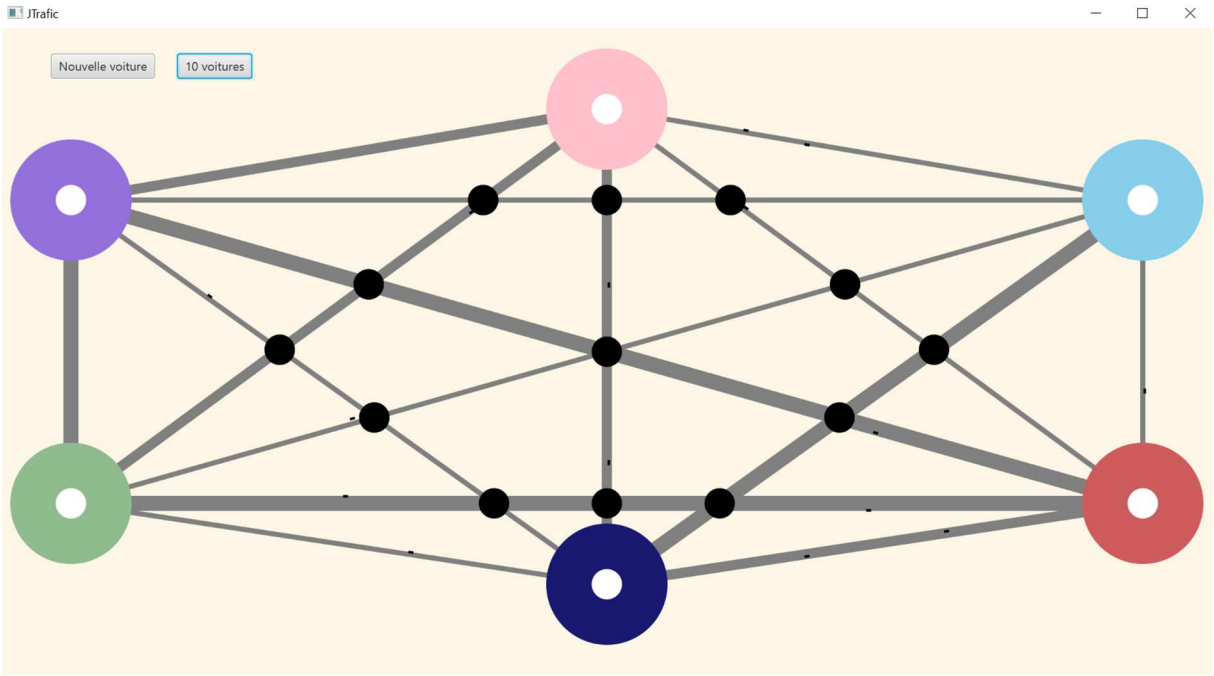
Pour plus de clarté, nous avons décidé de changer les couleurs des différents éléments et de créer un bouton **CarCreator** auquel nous avons ajouté une animation afin qu'à chaque fois que l'on clique dessus, une voiture soit automatiquement générée.




Le principe du simulateur étant posé, il fallait maintenant gérer les croisements de routes. Pour cela, nous avons créé 2 méthodes. Une méthode **ajoutIntersection** qui permet, lorsque 2 routes se croisent de définir ça comme une intersection, et une autre, **traceIntersection**, qui permet justement d'afficher ces intersections.



Nous avons également décidé de créer un autre bouton qui permet d'ajouter directement 10 voitures pour plus de simplicité.




Malgré le repérage des intersections, nous n'avons malheureusement pas réussi à faire en sorte que nos voitures s'arrêtent à celles-ci et par conséquent pas réussi à gérer les priorités et ralentissements. Nous avons pensé à plusieurs méthodes. Tout d'abord, nous avons réussi à récupérer la position de la voiture à chaque instant pour repérer lorsqu'elle se situe près de l'intersection mais nous n'arrivions pas à arrêter la voiture. De ce fait, nous avons pensé à fragmenter nos routes en les considérant comme une addition de segments. Un segment commence d'une ville de départ et s'arrête à une intersection, puis un nouveau segment commence jusqu'à la prochaine intersection et ainsi de suite jusque la vile d'arrivée. Mais cette méthode s'est avérée beaucoup trop complexe pour gérer la continuité des chemins. Nous n'arrivons toujours pas à gérer l'arrêt de l'animation de déplacement de la voiture malgré ces idées.

Tout au long du projet, nous nous sommes organisés et avons créé différentes classes, fonctions ou variables. Nous pouvons donc vous présenter, en plus du projet, un Javadoc et un diagramme de classe correspondant à celui-ci.

Aperçu du diagramme de classes :