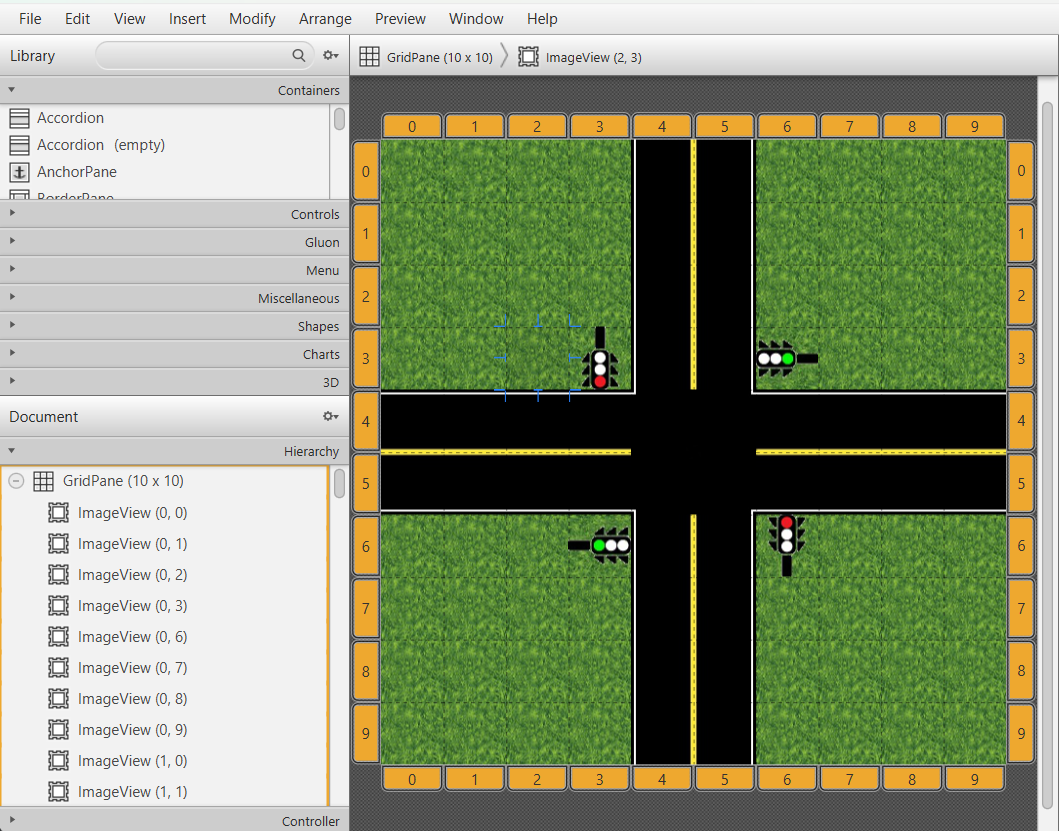
Rapport projet temps réel

# Git

Les différentes versions du projet sont disponibles sur les différentes branches du git : <https://github.com/MichelGreg/ProjetTempsReel>

# Interface graphique.

Pour l’interface graphique, j’ai décider d’utiliser la librairie javaFx que j’avais déjà utilisé auparavant. Cela permit de créer facilement l’interface grâce à l’outil sceneBuilder :



Au vu de l’énoncé du projet, je suis parti sur une gridView afin de pouvoir facilement déplacer les voitures de case en case.

# Version 1

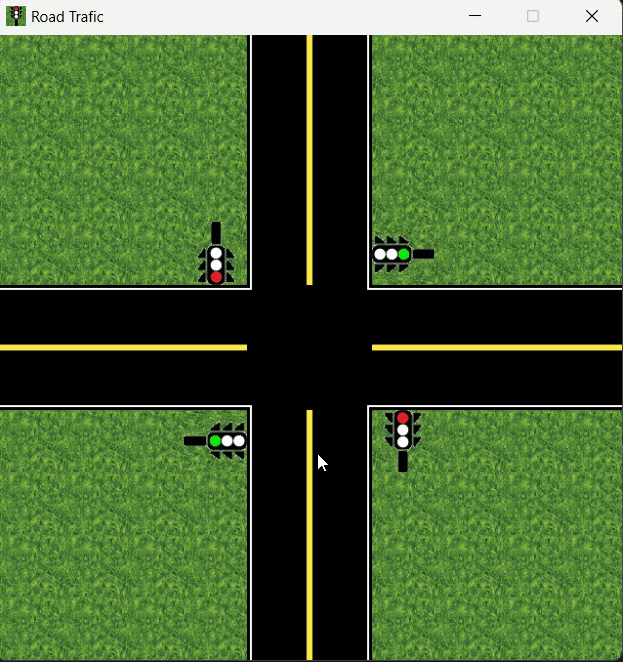
La première version fonctionnelle du projet utilisait 2 sémaphores *hsem* et *vsem* définis dans le main et partagés aux feux et aux voitures. Cette solution permettait de facilement implémenter une exclusion mutuelle de l’accès au carrefour.

Le thread *feux* s’occupant des feux bloquait l’un des deux sémaphores en fonction de la voie avec les feux verts.  
Chaque thread *voiture* avançait si elle pouvait et demandait à la vue de déplacer une image aux nouvelles coordonnées.  
Les voitures arrivant au feu rouge attendaient donc que le thread *feux* libère la possibilité d’aller dans le carrefour, puis bloquaient le carrefour jusqu’à en sortir pour assurer le fait qu’il n’y ait qu’une voiture qui passe à la fois.

Problèmes :

Cette implémentation avait 2 problèmes :

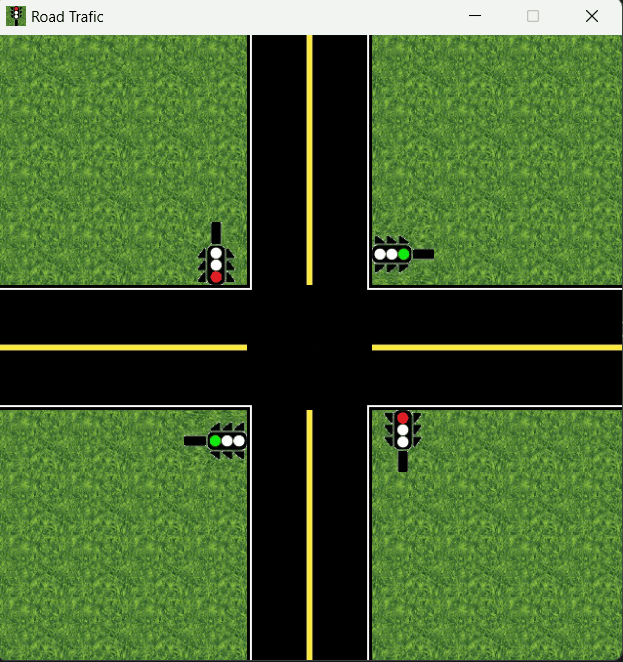
* Les voitures n’avaient pas conscience des autres, donc pas de collision et elles se superposaient en attendant.
* La synchronisation les feux étaient bloqués en attendant que toutes les voitures qui attendaient passent le carrefour



# Version 2

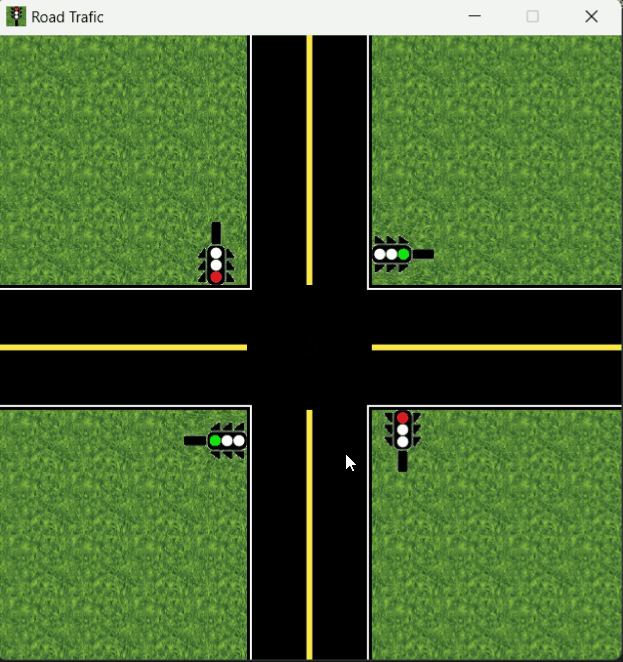
Pour corriger le premier problème, j’ai utilisé une matrice 10x10 représentant les cases du carrefour. Les cases vides ont la valeur -1, et les cases avec une voiture ont une valeur entre 0 et 3 représentant le sens dans lequel elles vont.   
Grâce à cela, elles peuvent savoir si la case devant elle est occupée avant d’y aller.  
Pour régler certains problèmes graphiques, ce ne sont plus les processus qui demandent à la vue de changer, mais c’est un listner sur la matrice qui adapte la vue en fonction des données de la matrice.

Ensuite, pour corriger le problème des feux, j’ai ajouté un mutex *carrefour* utilisé par les voitures afin d’attendre qu’elles aient la possibilité d’entrer dans le carrefour avant de bloquer un sémaphore. De cette manière, une voiture ne bloque plus de sémaphore lorsqu’elle traverse le carrefour, et les feux peuvent changer de couleur au milieu de la traversée. Cela empêche aussi les voiture de glisser le feu car elles étaient dans la liste d’attente.



# Version 3

Cette version rajoute une ambulance plus prioritaire que les voitures, la classe du processus ambulance hérite de la classe voiture avec les adaptations lui permettant de changer de voie. Avec l’implémentation actuelle du carrefour, le problème est que l’ambulance attend la libération de ce dernier avant de partir.   
En effet, pour être sûre de ne croiser personne, elle bloque le mutex carrefour comme les autres voitures, mais dès son apparition, ce qui peut prendre du temps si plusieurs voitures attendent.



Solution 1 : Rajouter un mutex qui gère priorise le passage de l’ambulance par rapport aux voitures.   
Cette solution a l’avantage d’être facile à mettre en place. De cette manière lorsque l’ambulance apparait, les autres voitures ne peuvent plus accéder au mutex du carrefour, donc l’ambulance peut passer en priorité.

Solution 2 : Créer un sémaphore personnalisé ayant une PriorityBlockingQueue afin de réveiller le thread avec la plus grande priorité en premier. Ainsi, si l’ambulance apparait et que 2 autres voitures attendent de passer au feu, cela permettrait qu’elle soit réveillée en première et ainsi coder une véritable priorité.   
Cette solution est cependant plus complexe à gérer et je n’ai pas réussi à l’implémenter à cause du notify qui levait une « java.lang.IllegalMonitorStateException: current thread is not owner »

# Version 4

Les voitures ont la possibilité de tourner au feu.