Projet DAC :

réalisé par :

* Hamida Oussama Islem / groupe : 03
* Ait Mbarek Idir / groupe : 03

Description:

Ce projet consiste à réaliser un système simulant la gestion du trafic de manière multithread.

Pour réaliser ce projet, nous avons utilisé la bibliothèque de création d'applications de bureau : "JavaFX". L'idée était d'utiliser ses caractéristiques graphiques intégrées pour donner à la logique de concurence une implémentation visible.

L'architecture de l'application :

Tous les fichiers importants du projet se trouvent dans le dossier "src"/source.

Le dossier source contient les images utilisées dans la partie graphique du projet :

* cross.png
* light_pole.png
* traffic-light.png
* the_blue_car.png
* the_green_car.png
* the_red_car.png

Et le paquet "sample" contenant :

* les classes java :
    * Car.java
    * CarCreator.java
    * CarThread.java
    * Controler.java
    * FireThread.java
    * Main.java

* un fichier css : sample.css pour styliser certains composants de l'interface graphique.

* un fichier fxml : sample.fxml pour définir certains composants graphiques.

L'es threads et sémaphores:

I / Les threads :

1) CarThread : Une classe java qui extends Thread, une instance de cette classe est instanciée pour chaque voiture individuelle dans le trafic (1 voiture = 1 instance). Ce thread gère les mouvements de la voiture tout en restant dans l'ordre et la disponibilité du feu.

* Son constructeur prend comme arguments : La voiture qu'il doit gérer , le Sémaphore de la direction dans laquelle la voiture se déplace , le Sémaphore de la liste des voitures en attente dans cette direction , le Sémaphore du feu , la liste des voitures en attente , la hauteur de la voiture , et la Pane (composant graphique).

La method run :

```
@Override
public void run() {
      // réserver la direction
       reserveDirection();
      // réserver le feu
       reserveFire();
      // réserver  la liste
       reserveList();
      // réserver  la voiture
       reserveCar(The_Car);
      int index =  CarList.indexOf(The_Car);
```

```
    if(index >= 0){
    // supprimer la voiture actuelle de la liste
        CarList.remove(index);
    }



    // circuler
    this.The_Car.move(MoveDuration,600);
    The_Car.getCarSemaphore().release();



    // repositionner toutes les voitures encore en attente
dans la liste

    repositionAllCars();
    ListSemaphore.release();

    // mettre en place une minuterie pour libérer les
sémaphores afin de laisser entrer la prochaine voiture.
    new java.util.Timer().schedule(
            new java.util.TimerTask() {
                @Override
                public void run() {

                    FireSemaphore.release();
                    DirectionSemaphore.release();
                }
            },
            MoveDuration/3 + 100
    );
    // mettre en place une minuterie pour supprimer le
modèle de voiture (nous n'en avons plus l'utilité).
    new java.util.Timer().schedule(
            new java.util.TimerTask() {
```

```
                @Override
                public void run() {
                    Platform.runLater(()->{

Pane.getChildren().remove(The_Car.getCarModel());
                    });
                }
            },
            MoveDuration
    );


}
```

2) CarCreator : Une classe java qui extends Thread, est responsable de la création de nouvelles voitures et de leur dépose pour attendre dans la file ou passer le carrefour.

Dans cette application, 2 instances de cette classe sont créées, chacune gère la création de voitures dans une direction (verticale ou horizontale).

* Son constructeur prend comme arguments : un String qui détermine la direction de création le Sémaphore du feu, Sémaphore de creation ,la hauteur de la voiture , et la Pane (composant graphique).

Le constructeur va créer :

* une liste de voitures dans le sens de la création.
* un sémaphore pour la liste de voitures
* un sémaphore pour la direction de la création.

La method run :

```
@Override
public void run() {
    // un sommeil différent pour le thread horizontal
(pour des raisons inhabituelles seulement)
    if(!Type.equals("vertical")) sleepOff(1000);

// mettre en place une minuterie pour libérer le sémaphore
afin de laisser entrer la prochaine voiture.
    while (true){
        int sleep_time = 4000;
        // créer un nouvel objet voiture
        Car newCar =  createCar();
```

```
            // réserver  la voiture
            reserveCar(newCar);

           // réserver  la list
            reserveList();


            int carNumb = CarList.size();

          // ajouter la nouvelle voiture a la list
            CarList.add(newCar);

// changer le temps de sommeil en fonction du nombre de voitures
            if(carNumb < 6 ){
                sleep_time = 4000;
            }
             if(carNumb < 3 ){
                sleep_time = 2000;
            }
            //


// cette méthode fait apparaître la nouvelle voiture et
prend sa place dans la file d'attente, elle utilise le
nombre de voitures en attente pour déterminer où elle doit
s'arrêter (en fait, elle connaît la distance qu'elle aura
à parcourir mais l'endroit où elle est rendue dépend du
nombre de voitures en attente, plus le nombre est grand,
plus elle sera loin)
            dropNewCar(newCar,carNumb);
//


            ListSemaphore.release();
            new java.util.Timer().schedule(
                    new java.util.TimerTask() {
                        @Override
                        public void run() {

newCar.getCarSemaphore().release();
                            createCarThread(newCar);
```

```
                    }
                },
                Drop_Duration
            );
        // endormir
            sleepOff(sleep_time);

    }

}
```

3) FireThread :

Une classe java qui extends Thread, une instance de is responsable for changing the fire color and managing the acess of each direction.

* Son constructeur prend comme arguments : deux nœuds graphiques (cercles) utilisés pour représenter l'état du feu (vert ou rouge).

Le constructeur crée deux sémaphores F1,F2 et un boolean Fire qui représentent l'état logique du feu .

La method run :

```
Override
    public void run() {
        Green_Light.setFill(Color.valueOf("#1fff5a"));
        Red_Light.setFill(Color.GRAY);



        while(true){

            // endormir
             try {
                 sleep(8000);
             } catch (InterruptedException e) {
                 e.printStackTrace();
             }
```

```
                // inverse la valeur de "Fire".
                 reverse();
                 if(this.Fire){
        // dans ce cas le feu est vert donc toutes les voitures horizontales doivent attendre
                        reserve(F2);
                        F1.release();
        // le cercle vert est affiché et le cercle rouge est caché.
Green_Light.setFill(Color.valueOf("#1fff5a"));
Red_Light.setFill(Color.GRAY);



                 }else{


// dans ce cas le feu est rouge donc toutes les voitures verticales doivent attendre
                        reserve(F1);
                        F2.release();
// le cercle rouge est affiché et le cercle vert est caché.
                        Green_Light.setFill(Color.GRAY);
                        Red_Light.setFill(Color.valueOf("#ff1f1f"));



                 }


            }
```

I I / Les Sémaphores :

1) Sémaphore de la liste des voitures :

ce sémaphore est instancié avec une valeur de 1 :

```
Semaphore ListSemaphore =  new Semaphore(1) ;
```

Et est utilisé pour contrôler l'accès à la liste des voitures en attente,

Chaque "CarCreator" crée un tel sémaphore et le passe à toutes les voitures qu'il crée.

2) Sémaphore de la direction :

ce sémaphore est instancié avec une valeur de 1 :

```
Semaphore DirectionSemaphore =  new Semaphore(1) ;
```

Il est utilisé pour contrôler l'accès à une certaine direction afin que seulement une voiture de cette direction puisse traverser à la fois,

Chaque "CarCreator" crée un tel sémaphore et le passe à toutes les voitures qu'il crée.

3) Sémaphore de la voiture :

ce sémaphore est instancié avec une valeur de 1 :

```
Semaphore CarSemaphore =  new Semaphore(1) ;
```

Il est utilisé pour contrôler l'accès à une certaine voiture afin que seul 1 Thread puisse la déplacer en même temps,

Lorsqu'une voiture est créée, dans son constructeur, ce sémaphore est créé.

4) Sémaphore de creation :

ce sémaphore est instancié avec une valeur de 1 :

```
Semaphore create =  new Semaphore(1) ;
```

Il est utilisé pour éviter certaines erreurs de threads "JavaFX", chaque fois qu'un CarCreator crée et prépare une voiture, il utilise d'abord ce sémaphore avant de rendre la voiture dans l'interface graphique.

5) Sémaphore de feu “F1” :

ce sémaphore est instancié avec une valeur de 1 :

```
Semaphore F1 =  new Semaphore(1) ;
```

Et est utilisé pour contrôler l'accès à une direction en fonction du feu, de sorte que seulement lorsque le feu est vert, les voitures peuvent traverser.

5) Sémaphore de feu “F2” :

ce sémaphore est instancié avec une valeur de 0 :

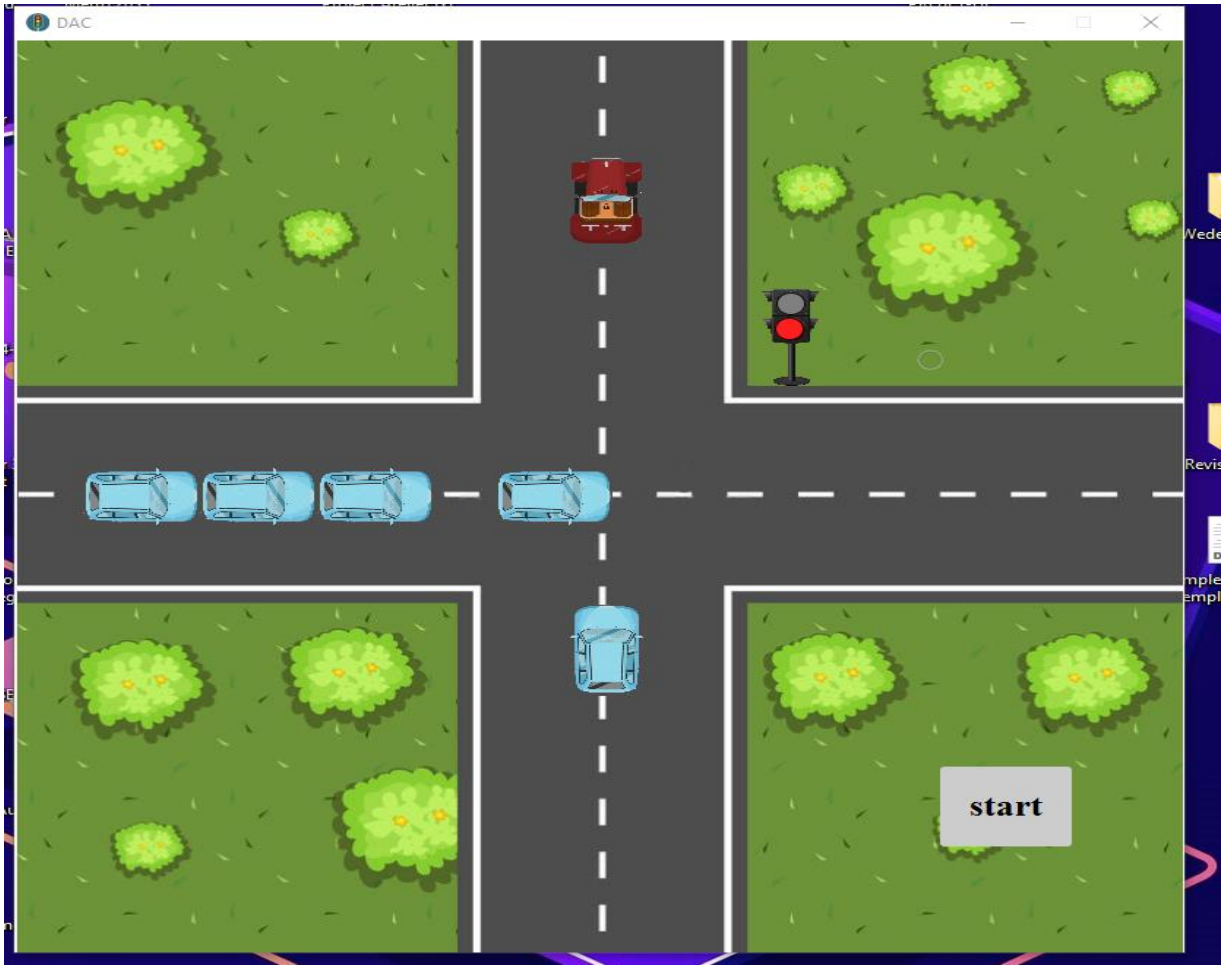
```
Semaphore F2 =  new Semaphore(0) ;
```

Et est utilisé pour contrôler l'accès à une direction en fonction du feu, de sorte que seulement lorsque le feu est rouge, les voitures peuvent traverser.

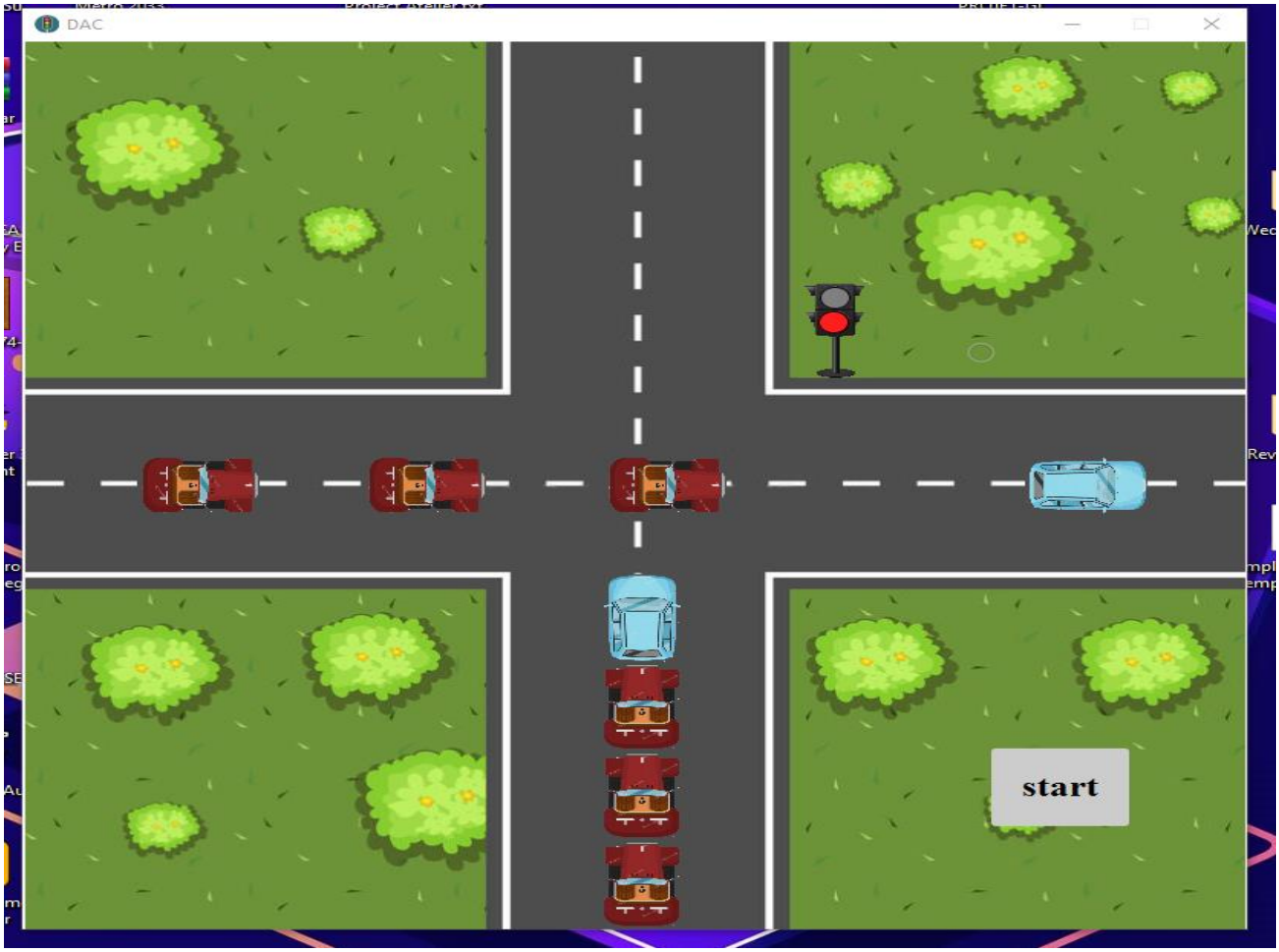
I I I / les Captures d'écran :

// Avant d'appuyer sur “start”

// Après d'appuyer sur “start”


DAC
start


DAC
start


DAC
start