SIMULADOR DE TRÁFEGO

Angelina Siqueira

Gabriel Perini

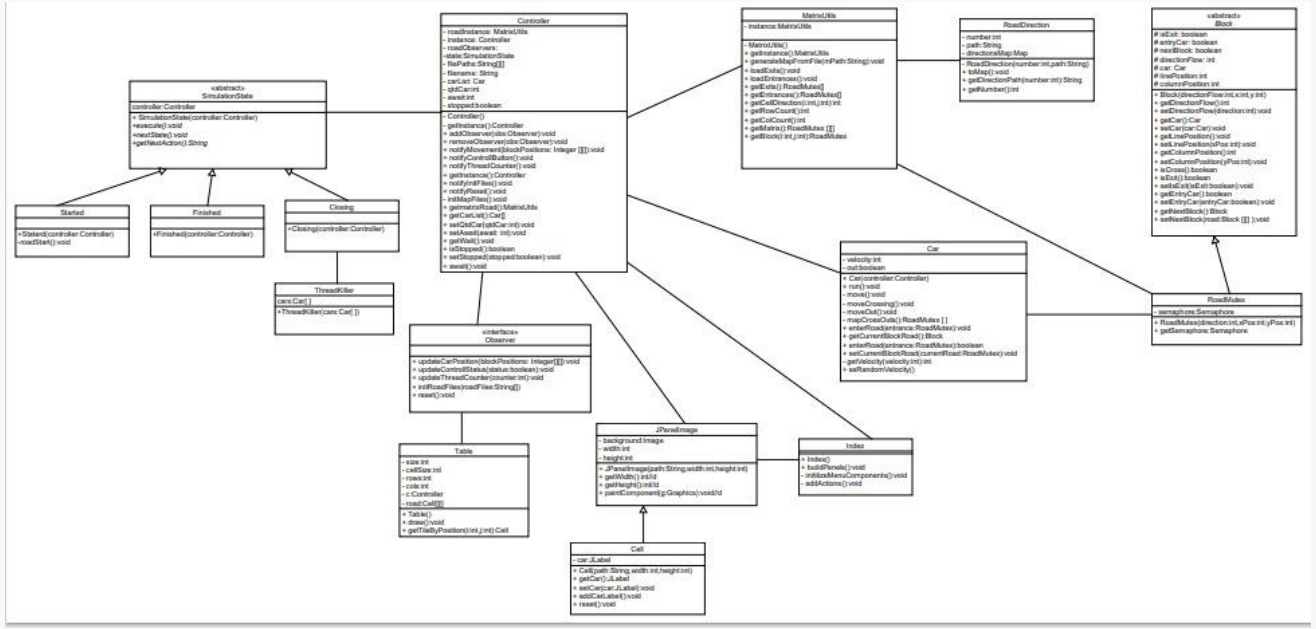
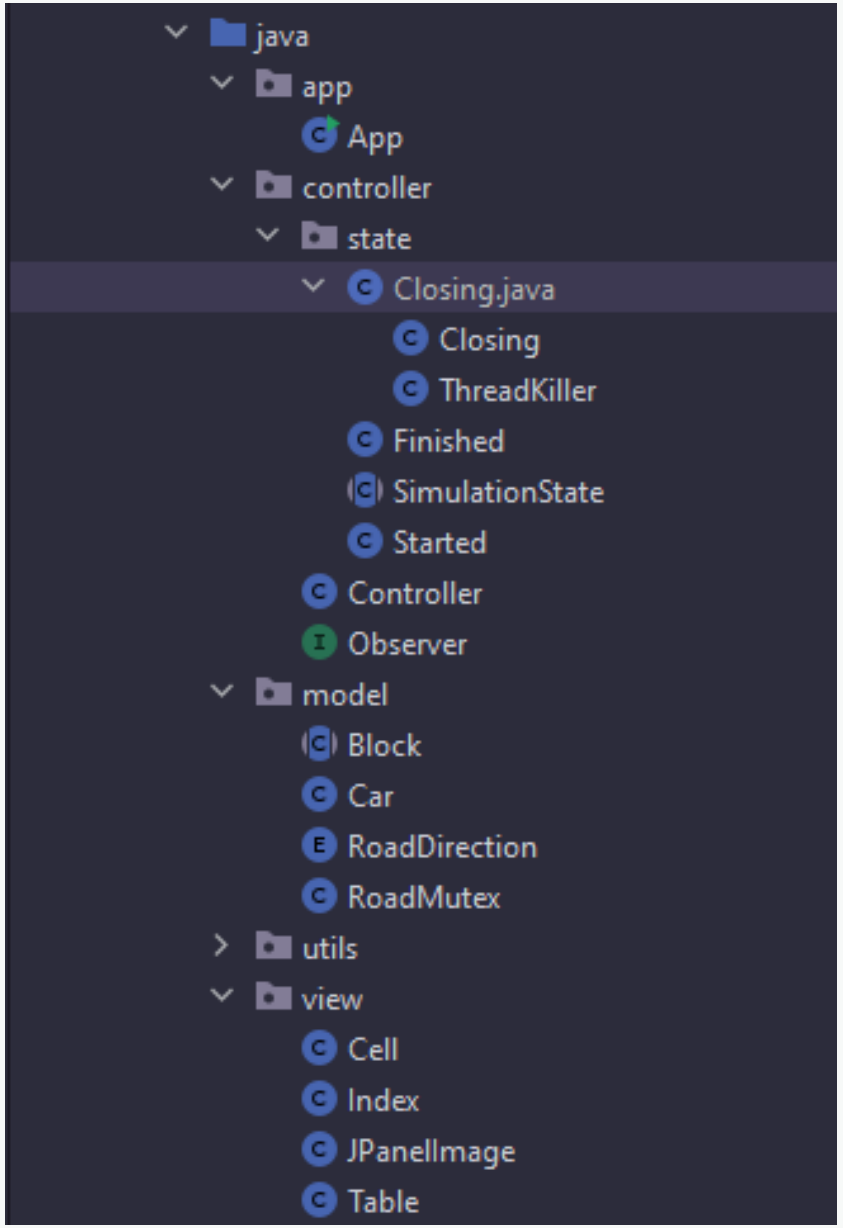
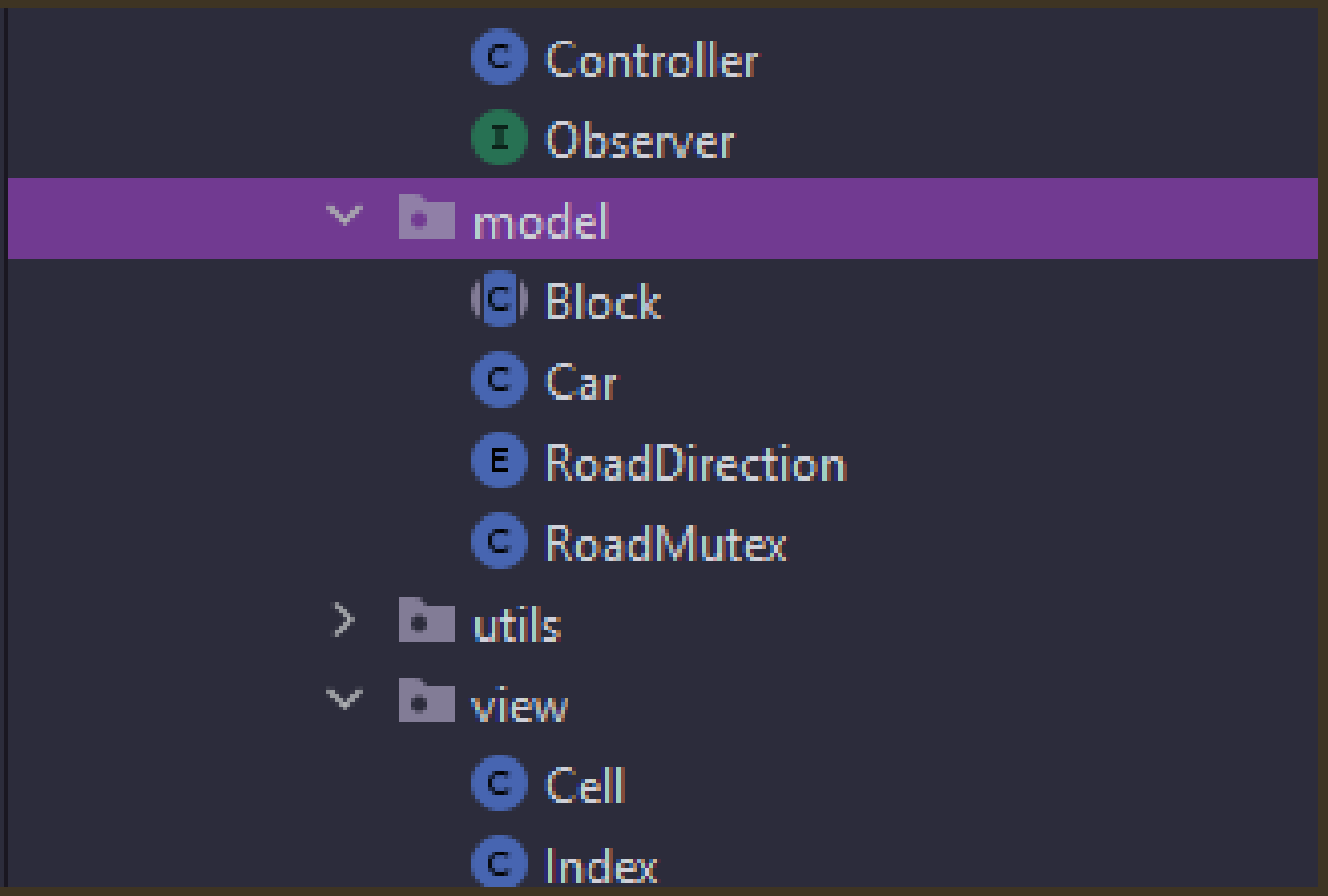


Diagrama de classe

Estrutura do projeto




Controller
Observer
model
Block
Car
RoadDirection
RoadMutex
utils
view
Cell
Index

MVC

```java
public interface Observer {

    public void updateCarPosition(Integer [][]blockPositions);

    public void updateControllStatus(boolean status);

    public void updateThreadCounter(int counter);

    public void initRoadFiles(String[] roadFiles);

    public void reset();

}

```

Observer

```
10  public class Table extends JPanel implements Observer {
11
12      private final int size = 575;
13      private int cellSize;
14      private int rows;
15      private int cols;
16      private Controller c;
17      private Cell[][] road;
```

```
11  public class Index extends JFrame implements Observer {
12
13      Table road;
14      Controller controller;
15      JPanelImage settingsPanel;
16      GridBagLayout gbl = new GridBagLayout();
17      GridBagConstraints layoutConstraint = new GridBagConstraints();
18
```

Observer

```
11
12  public class MatrixUtils {
13
14      private static MatrixUtils instance;
15
16      private MatrixUtils() {
17      }
18
19      public static MatrixUtils getInstance() {
20          if (instance == null) {
21              instance = new MatrixUtils();
22          }
23          return instance;
24      }
25
```

Singleton

```java
public class Controller {

    private final MatrixUtils roadInstance = MatrixUtils.getInst
    private static Controller instance;
    private List<Observer> roadObservers = new ArrayList<>();
    private String filename = "src/main/resources/casefiles/malh

    //Singleton
    private Controller() {
        try {
            RoadDirection.toMap();
            this.roadInstance.generateMapFromFile(filename);
        } catch (IOException e) {
            e.printStackTrace();
        }

    }
```

Singleton

```java
*/
public abstract class SimulationState {

    Controller controller;

    public SimulationState(Controller controller){
        this.controller = controller;
    }

    public abstract void execute();

    public abstract void nextState();

    public abstract String getNextAction();

```

```java
public class Started extends SimulationState {

    public Started(Controller controller) {
        super(controller);
    }


    @Override
    public void execute() {
        if(!controller.isStopped()){
            roadStart();
        }
    }

```

```java
public class Closing extends SimulationState {

    public Closing(Controller controller) {
        super(controller);
    }

    @Override
    public void execute() {
        controller.await();
        if (controller.getCarList().isEmpty()) {
            nextState();
        }
    }

```

```java
public class Finished extends SimulationState {

    public Finished(Controller controller) {
        super(controller);
    }


    @Override
    public void execute() {
        controller.await();
    }

    @Override
```

State

Requisitos Obrigatórios

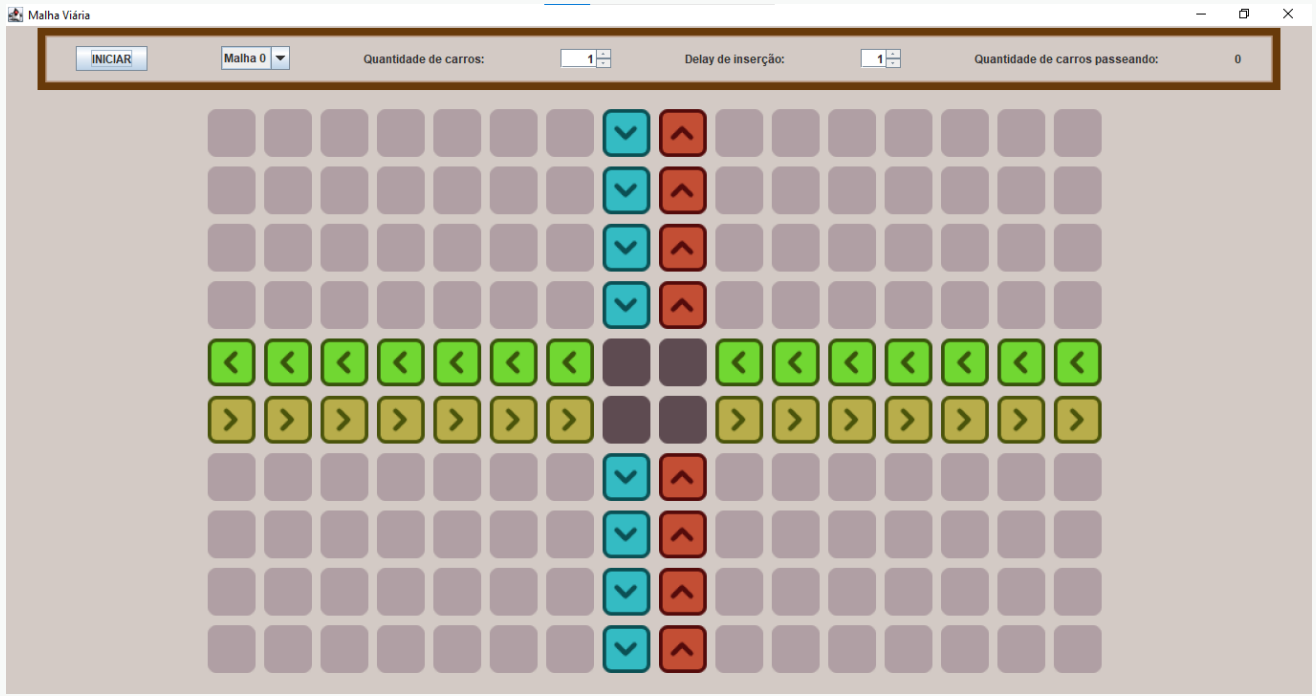
- limite para quantidade de veículos;

- Iniciar Simulação com a restrição anterior;

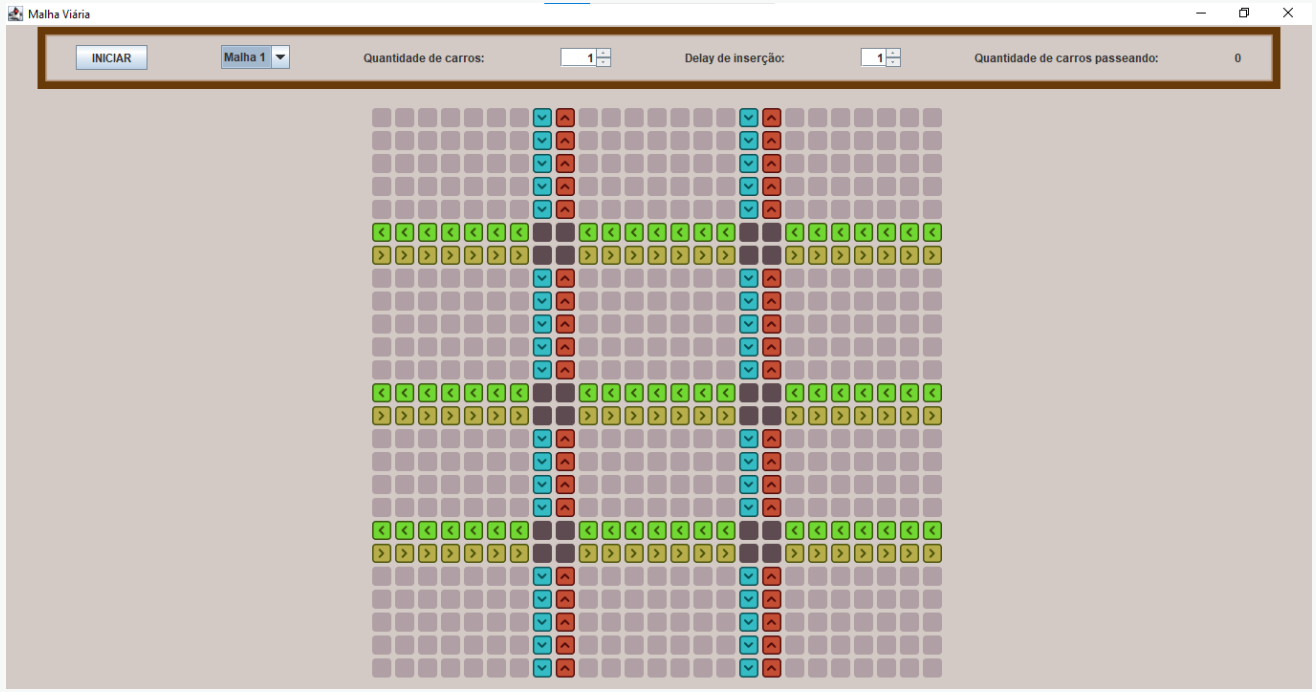
Simulação encerra - não adicionar e encerra imediatamente todos os veículos;

```java

    public void generateMapFromFile(String mPath) throws IOException {
        Path path = Paths.get(mPath);
        List<String> lines = Files.readAllLines(path);
        System.out.println(lines.get(0));
        System.out.println(lines.get(1));
        matriz = new RoadMutex[Integer.parseInt(lines.get(0))][Integer.parseInt(lines.get(1))];
        StringBuilder strRoad = new StringBuilder();

        //Criação da Matriz
        for (int i = 2; i < lines.size(); i++) {
            String[] line = lines.get(i).split("\t");
            for (int j = 0; j < line.length; j++) {
                matriz[i - 2][j] = new RoadMutex(Integer.parseInt(line[j]), i - 2, j);

                strRoad.append(line[j] + " ");
            }
            strRoad.append("\n");
        }

        //Setando nextBlock de todas as células
        for (int i = 0; i < matriz.length; i++) {
            for (int j = 0; j < matriz[i].length; j++) {
                matriz[i][j].setNextBlock(matriz);
            }
        }

        loadEntrances();
        loadExits();

        System.out.println(strRoad);
    }
```

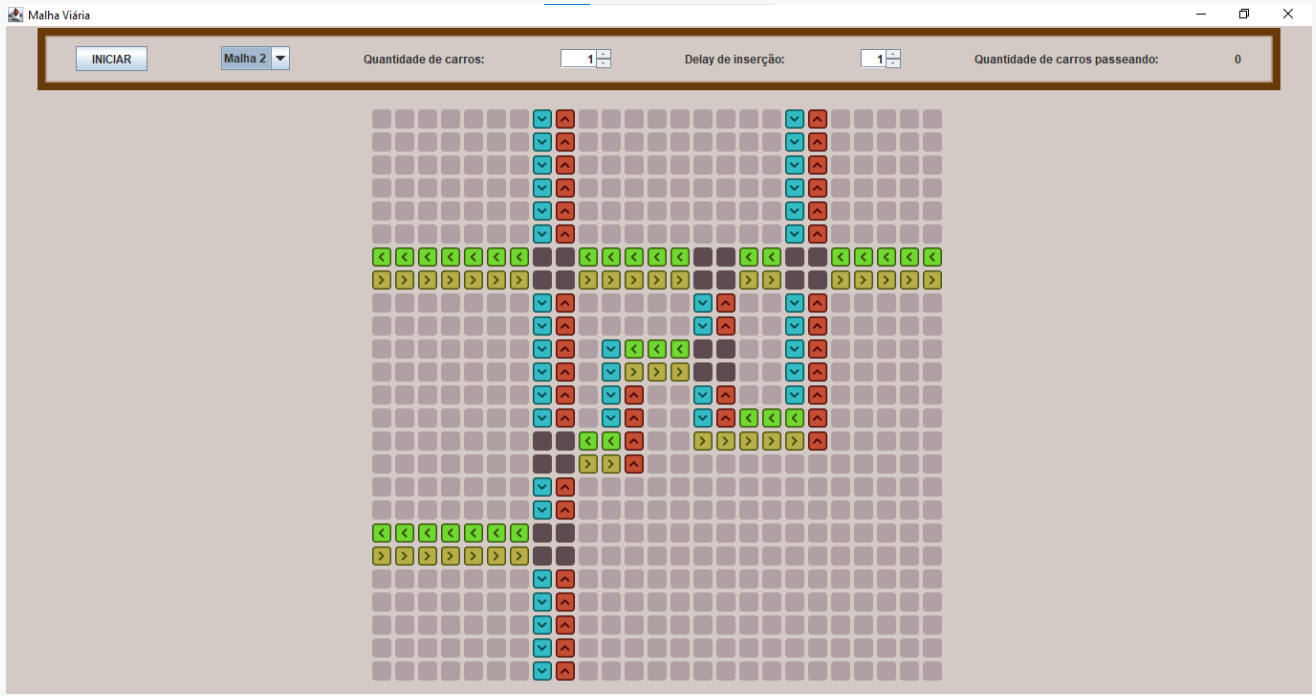
Matrix


Malha Viária
INICIAR
Malha 0
Quantidade de carros:
1
Delay de inserção:
1
Quantidade de carros passeando:
0

Malha – UI1


Malha Viária
INICIAR
Malha 1
Quantidade de carros:
1
Delay de inserção:
1
Quantidade de carros passeando:
0

Malha - UI2


Malha Viária
INICIAR
Malha 2
Quantidade de carros:
1
Delay de inserção:
1
Quantidade de carros passeando:
0

Malha - UI3

```java
public void start() {
    int i = 0;
    this.stopped = false;
    notifyControllButton();
    while (true) {
        if (carList.size() < qtdCar) {
            Car newCar = new Car(this);
            RoadMutex entrance = roadInstance.getEntrances().get(i);
            Integer[][] positions = {{null, null}, {entrance.getLinePosition(), entrance.getColumn

            newCar.enterRoad(entrance);
            carList.add(newCar);
            notifyThreadCounter();
            notifyMovement(positions);
            newCar.start();

            i++;
            if (i == roadInstance.getEntrances().size()) {
                i = 0;
            }

            try {
                Thread.currentThread().sleep(this.await);
            } catch (InterruptedException ex) {
                Logger.getLogger(Controller.class.getName()).log(Level.SEVERE, null, ex);
            }
        }

    }
}
```

Quantidade de veículos & Início da Simulação

```java
public class RoadMutex extends Block {

    private final Semaphore semaphore = new Semaphore(1);

    public RoadMutex(int direction, int xPos, int yPos) {
        super(direction, xPos, yPos);
    }

    public Semaphore getSemaphore() {
        return semaphore;
    }
}
```

Mecanismo de exclusão mútua

Requisitos Opcionais

- Intervalo na inserção de veículos;

- Encerrar a simulação aguardando os veículos se retirarem;

```
69      public MatrixUtils getMatrixRoad() {
70          return roadInstance;
71      }
72
73      private List<Car> carList = new ArrayList<>();
74      private int qtdCar;
75      private int await;
76      private boolean stopped = true;
77
78  >   public void setQtdCar(int qtdCar) {···
80      }
81
82      public void setAwait(int await) {
83          this.await = await * 1000;
84      }
85
86      public void getAwait(int await) {
87          this.await = await;
88      }
89
```

Opção de intervalo de inserção

Hands on!

- muitos carros

- uma malha

- carros rápidos

- seguir semáforo