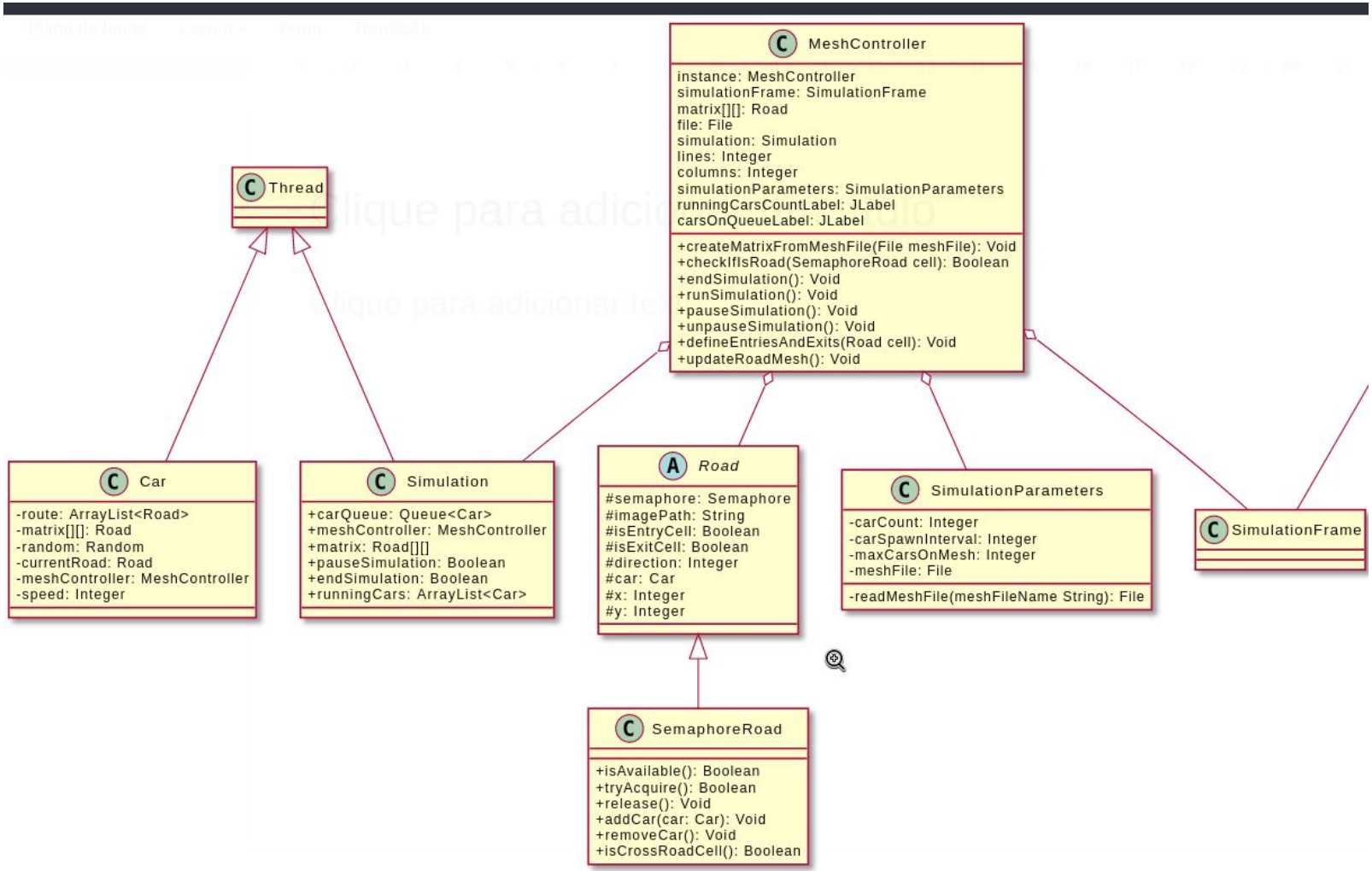
Simulador de Tráfego em Malha Viária

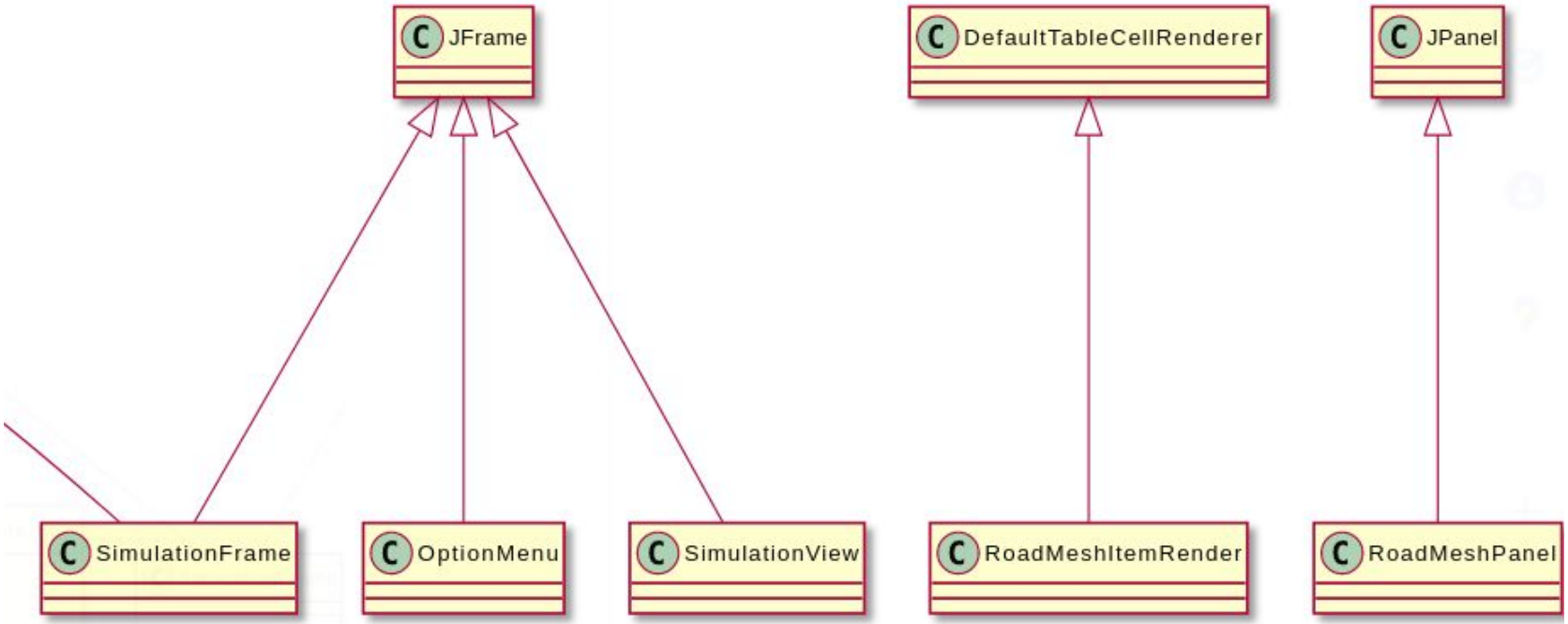
65DSD

Lucas Dolsan
Fernando dos Santos

Tópicos

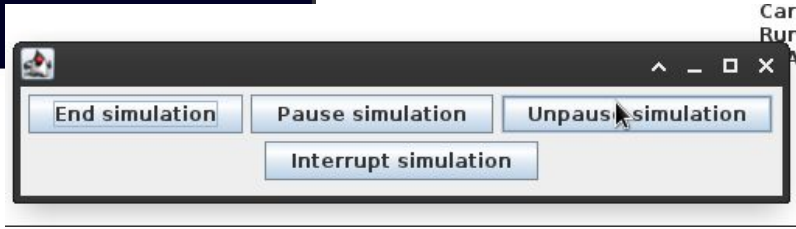
1. Leitura do arquivo da malha
2. Iniciar simulação
3. Gerar carros
4. Definir rotas
5. Iniciar threads dos carros (mover)


MeshController
instance: MeshController
simulationFrame: SimulationFrame
matrix[][]: Road
file: File
simulation: Simulation
lines: Integer
columns: Integer
simulationParameters: SimulationParameters
runningCarsCountLabel: JLabel
carsOnQueueLabel: JLabel
+createMatrixFromMeshFile(File meshFile): Void
+checkIfIsRoad(SemaphoreRoad cell): Boolean
+endSimulation(): Void
+runSimulation(): Void
+pauseSimulation(): Void
+unpauseSimulation(): Void
+defineEntriesAndExits(Road cell): Void
+updateRoadMesh(): Void
Thread
Car
-route: ArrayList<Road>
-matrix[][]: Road
-random: Random
-currentRoad: Road
-meshController: MeshController
-speed: Integer
Simulation
+carQueue: Queue<Car>
+meshController: MeshController
+matrix: Road[][]
+pauseSimulation: Boolean
+endSimulation: Boolean
+runningCars: ArrayList<Car>
Road
#semaphore: Semaphore
#imagePath: String
#isEntryCell: Boolean
#isExitCell: Boolean
#direction: Integer
#car: Car
#x: Integer
#y: Integer
SimulationParameters
-carCount: Integer
-carSpawnInterval: Integer
-maxCarsOnMesh: Integer
-meshFile: File
-readMeshFile(meshFileName String): File
SimulationFrame
SemaphoreRoad
+isAvailable(): Boolean
+tryAcquire(): Boolean
+release(): Void
+addCar(car: Car): Void
+removeCar(): Void
+isCrossRoadCell(): Boolean


JFrame
DefaultTableCellRenderer
JPanel
SimulationFrame
OptionMenu
SimulationView
RoadMeshItemRender
RoadMeshPanel


OptionMenu
Car count
100
Spawn rate (seconds)
1
Maximum cars on mesh at the same time
5
mesh 1
mesh 2
mesh 3
Start simulation


SimulationFrame
Cars on queue: 57
Running cars: 5
PLAYING...


End simulation
Pause simulation
Unpause simulation
Interrupt simulation

```java
1 usage  Lucas Dolsan
public void createMatrixFromMeshFile(File meshFile) {
    Scanner meshScanner = null;
    try {

        meshScanner = new Scanner(meshFile);

        while (meshScanner.hasNextInt()) {
            this.lines = meshScanner.nextInt();
            this.columns = meshScanner.nextInt();

            this.matrix = new Road[this.columns][this.lines];

            for (int y = 0; y < this.lines; y++) {
                for (int x = 0; x < this.columns; x++) {
                    int direction = meshScanner.nextInt();
                    SemaphoreRoad cell = new SemaphoreRoad(x, y, direction);
                    if (this.checkIfIsRoad(cell)) {
                        this.defineEntriesAndExits(cell);
                    }
                    this.matrix[x][y] = cell;
                }
            }
        }

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    } finally {
        meshScanner.close();
    }
}
```

MeshController.java

```
10
16
0   0   0   0   0   0   0   3   1   0   0   0   0   0   0   0
0   0   0   0   0   0   0   3   1   0   0   0   0   0   0   0
0   0   0   0   0   0   0   3   1   0   0   0   0   0   0   0
0   0   0   0   0   0   0   3   1   0   0   0   0   0   0   0
4   4   4   4   4   4   4   12  10  4   4   4   4   4   4   4
2   2   2   2   2   2   2   11  9   2   2   2   2   2   2   2
0   0   0   0   0   0   0   3   1   0   0   0   0   0   0   0
0   0   0   0   0   0   0   3   1   0   0   0   0   0   0   0
0   0   0   0   0   0   0   3   1   0   0   0   0   0   0   0
0   0   0   0   0   0   0   3   1   0   0   0   0   0   0   0
```

Simulation.java

```
Lucas Dolsan
public void run() {
    this.carQueue = this.loadCars();

    this.meshController.getCarsOnQueueLabel().setText("Cars on queue: " + this.carQueue.size());

    this.matrix = this.meshController.getMatrix();

    this.meshController.updateRoadMesh();

    SimulationParameters params = this.meshController.getSimulationParameters();

    while(!carQueue.isEmpty()) {
        for (int y = 0; y < this.meshController.getLines(); y++) {
            for (int x = 0; x < this.meshController.getColumns(); x++) {

                Road startingRoad = matrix[x][y];

                Boolean canAddCarsToSimulation = (
                    startingRoad.isEntryCell() &&
                    startingRoad.isAvailable() &&
                    !carQueue.isEmpty() &&
                    !this.pauseSimulation &&
                    !this.endSimulation &&
                    this.runningCars.size() < params.getMaxCarsOnMesh()
                );

                if (canAddCarsToSimulation) {...}
            }
        }
    }
}
```

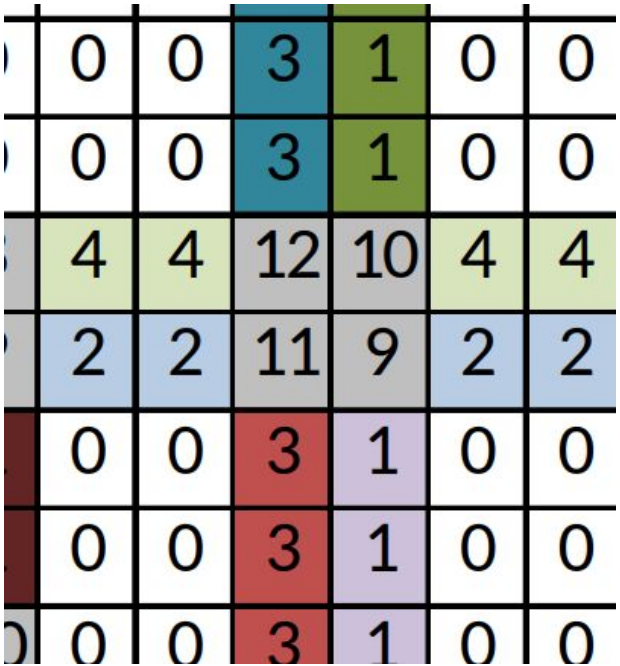
Simulation.java

```
if (canAddCarsToSimulation) {
    try {
        sleep( millis: params.getCarSpawnInterval() * 1000);

        Car car = carQueue.remove();

        this.meshController.getCarsOnQueueLabel().setText("Cars on queue: " + this.carQueue.size());

        car.defineRoute(startingRoad);
        car.start();

        car.setSimulation(this);

        this.addRunningCar(car);
        this.meshController.getRunningCarsCountLabel().setText("Running cars: " + this.runningCars.size());

    } catch (Exception e) {
        throw new RuntimeException(e);
    }
}
```

Car.java defineRoute()

```
1 usage  Lucas Dolsan
public void defineRoute(Road startingRoad) throws Exception {
    // defineRoute will append road cells to the route list until the last one is an exit
    boolean foundExit = false;

    Road nextRoadOnRoute = startingRoad;

    route.add(nextRoadOnRoute);

    // this variable counts the amount of crossroad cells immediately on route, to prevent driving in circles
    // inside the crossroad
    int amountOfCrossroadsCellsOnCurrentCrossroad = 0;

    while (!foundExit) {...}
}
```

Car.java




```java
while (!foundExit) {
    int direction = nextRoadOnRoute.getDirection();
    int currentRouteX = nextRoadOnRoute.getX();
    int currentRouteY = nextRoadOnRoute.getY();

    boolean isRoad = direction <= 4;

    if (isRoad) {
        nextRoadOnRoute = this.chooseRoad(direction, currentRouteX, currentRouteY);
    } else {
        nextRoadOnRoute = this.chooseCrossroad(
                direction,
                currentRouteX,
                currentRouteY,
                amountOfCrossroadsCellsOnCurrentCrossroad
        );
        if (nextRoadOnRoute.isCrossRoadCell()) {
            amountOfCrossroadsCellsOnCurrentCrossroad++;
        } else {
            amountOfCrossroadsCellsOnCurrentCrossroad = 0;
        }
    }

    route.add(nextRoadOnRoute);

    foundExit = nextRoadOnRoute.isExitCell();
}
}
```

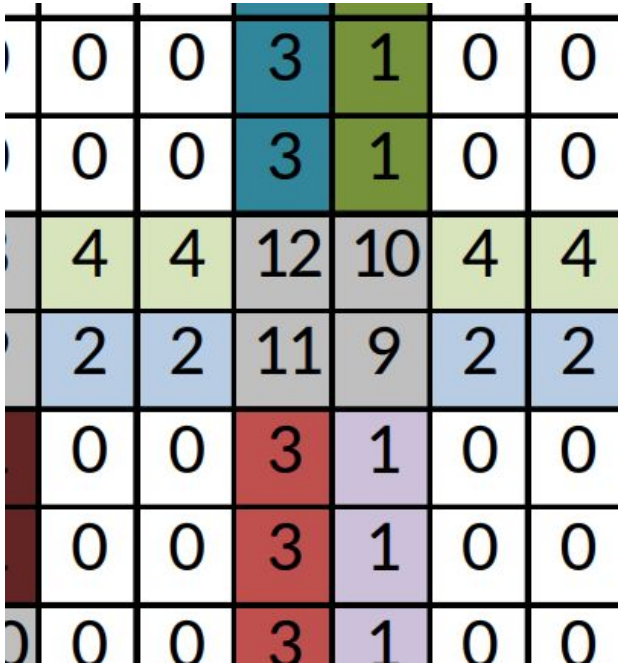
Car.java run()

```
Lucas Dolsan
@Override
public void run() {
    while (!route.isEmpty()) {
        int nextRoadIndex = 0;

        if (route.get(nextRoadIndex).isCrossRoadCell()) {
            handleCrossroads();
        } else {
            Road road = this.route.remove(nextRoadIndex);

            this.move(road, needsAcquire: true);
        }
    }
    this.getCurrentRoad().removeCar();
    this.getCurrentRoad().release();
    this.simulation.removeRunningCar(this);
    this.meshController.updateRoadMesh();
}
```

Car.java

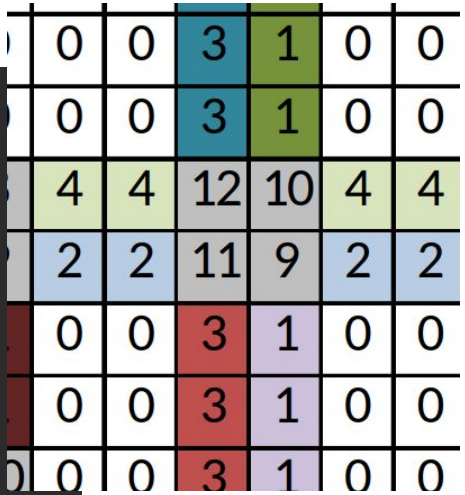
handleCrossroads()

```
1 usage   Lucas Dolsan
private void handleCrossroads() {
    try {
        sleep(this.speed);
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    }

    ArrayList<Road> crossroadsToAcquire = new ArrayList<>();
    ArrayList<Road> acquiredCrossroads = new ArrayList<>();


    for (int i = 0; i < this.route.size(); i++) {
        Road road = this.route.get(i);

        // take into consideration the next road cell right outside the crossroads
        crossroadsToAcquire.add(road);

        if (!road.isCrossRoadCell()) {
            break;
        }
    }
```



Car.java

handleCrossroads()




```
    for (Road crossroadToAcquire : crossroadsToAcquire) {
        if (crossroadToAcquire.tryAcquire()) {
            acquiredCrossroads.add(crossroadToAcquire);
        } else {
            // failed to acquire all the crossroads + the road right outside the crossroad
            for (Road acquiredRoad : acquiredCrossroads) {
                acquiredRoad.release();
            }
            break;
        }
    }

    boolean acquiredAllNecessaryCrossroads = acquiredCrossroads.size() == crossroadsToAcquire.size();

    if (acquiredAllNecessaryCrossroads) {
        for (Road acquiredCrossroad : acquiredCrossroads) {
            this.route.remove(acquiredCrossroad);
            this.move(acquiredCrossroad, needsAcquire: false);
        }
    }
}
```