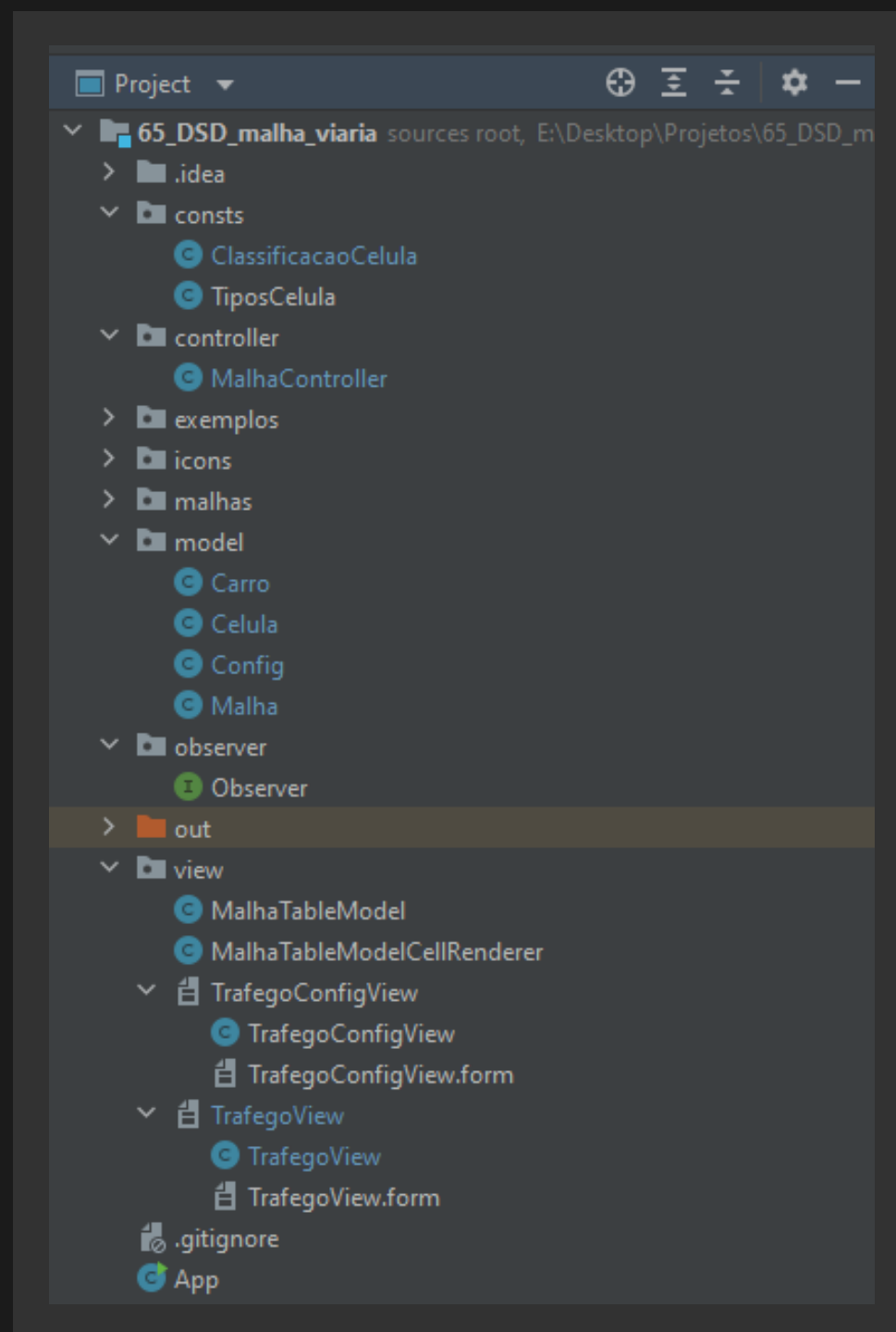




Repositório



Model



Controller



View



Observer



Consts



Malhas



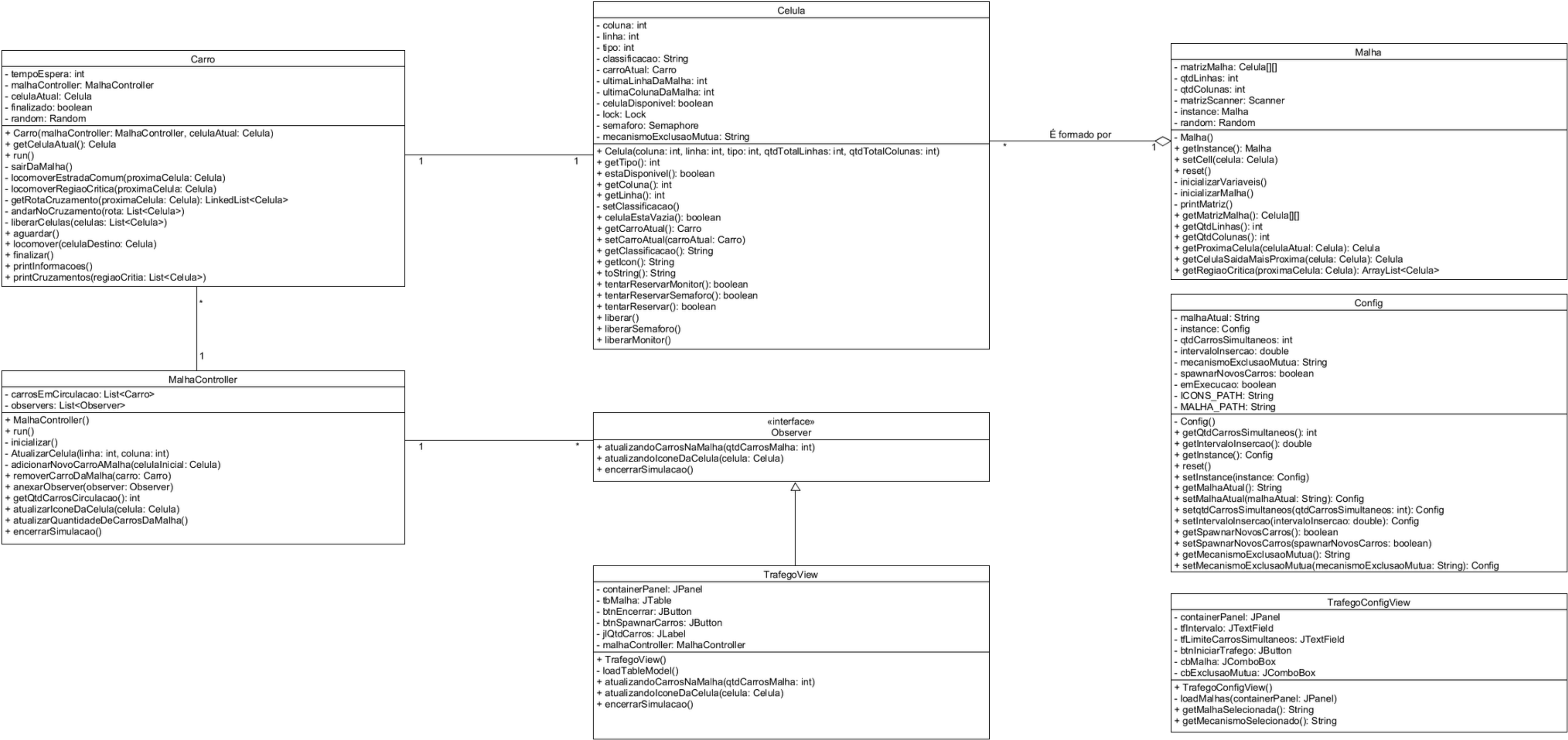
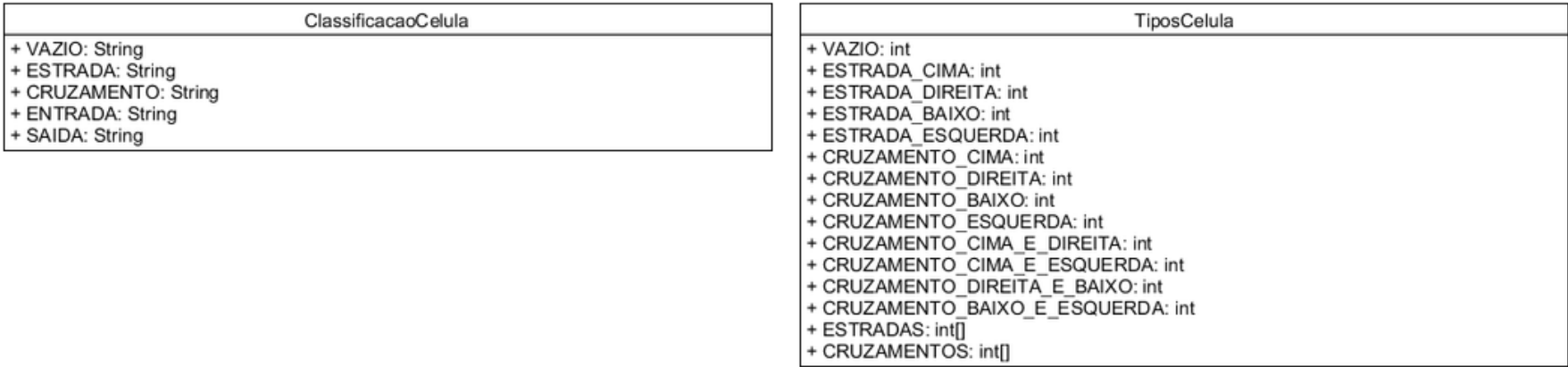
Exemplos



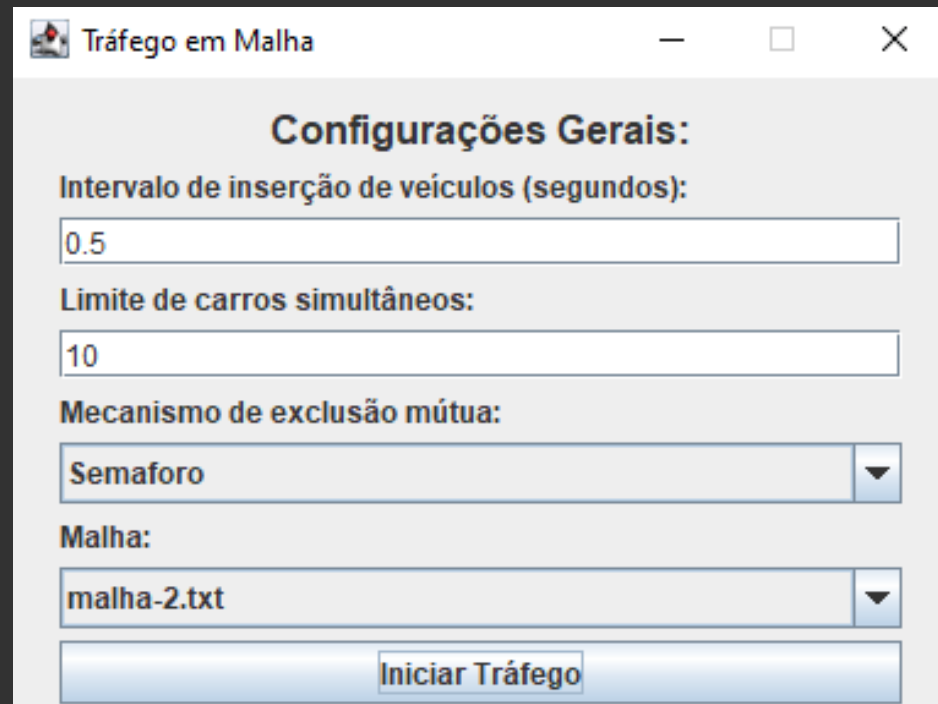
Icons



App



Config



Tráfego em Malha

Configurações Gerais:

Intervalo de inserção de veículos (segundos):
0.5

Limite de carros simultâneos:
10

Mecanismo de exclusão mútua:
Semaforo

Malha:
malha-2.txt

Iniciar Tráfego

```
btnIniciarTrafego.addActionListener((ActionEvent e) -> {  
    Config.reset();  
    Config.getInstance()  
        .setMalhaAtual(getMalhaSelecionada())  
        .setMecanismoExclusaoMutua(getMecanismoSelecionado())  
        .setqtdCarrosSimultaneos(Integer.parseInt(tfLimiteCarrosSimultaneos.getText()))  
        .setIntervaloInsercao(Double.parseDouble(tfIntervalo.getText()));  
    Malha.reset();  
    new TrafegoView();  
    super.dispose();  
});
```

```
public class Config {  
  
    private static Config instance;  
    private String malhaAtual;  
    private int qtdCarrosSimultaneos;  
    private double intervaloInsercao;  
    private String mecanismoExclusaoMutua;  
  
    private boolean spawnarNovosCarros = true;  
  
    public static final String ICONS_PATH = "icons/";  
    public static final String MALHA_PATH = "malhas/";  
  
    public boolean emExecucao = false;  
  
    public static synchronized Config getInstance(){  
        if (instance == null)  
            reset();  
        return instance;  
    }  
  
    public static synchronized void reset(){  
        instance = new Config();  
    }  
}
```

Celula

```
public class Celula {

    private int coluna;
    private int linha;
    private int tipo; // Representa a direção da estrada, é correspondente ao número na matriz
    private String classificacao; // Representa se a estrada é uma ENTRADA/SAIDA/CRUZAMENTO/VAZIO
    private Carro carroAtual = null;
    private int ultimaLinhaDaMalha;
    private int ultimaColunaDaMalha;
    private boolean celulaDisponivel = true;
    private Lock lock;
    private Semaphore semaforo;

    private String mecanismoExclusaoMutua;

    public Celula(int coluna, int linha, int tipo, int qtdTotalLinhas, int qtdTotalColunas) {
        this.mecanismoExclusaoMutua = Config.getInstance().getMecanismoExclusaoMutua();
        this.lock = new ReentrantLock();
        this.semaforo = new Semaphore(permits: 1);
        this.coluna = coluna;
        this.linha = linha;
        this.tipo = tipo;
        this.ultimaLinhaDaMalha = qtdTotalLinhas - 1;
        this.ultimaColunaDaMalha = qtdTotalColunas - 1;
        this.setClassificacao();
    }
}
```

Consts






Classificação



Tipo

Consts


▼  consts















-  ClassificacaoCelula
-  TiposCelula

```
public class ClassificacaoCelula {  
  
    public static final String VAZIO = "VAZIO";  
    public static final String ESTRADA = "ESTRADA";  
    public static final String CRUZAMENTO = "CRUZAMENTO";  
    public static final String ENTRADA = "ENTRADA";  
    public static final String SAIDA = "SAIDA";  
  
}
```

```
Celula celulaAtual = Malha.getInstance().getMatrizMalha()[linha][coluna];  
if (!celulaAtual.getClassificacao().equals(ClassificacaoCelula.ENTRADA))  
    return;
```

```
public class TiposCelula {  
  
    public static final int VAZIO = 0;  
    public static final int ESTRADA_CIMA = 1;  
    public static final int ESTRADA_DIREITA = 2;  
    public static final int ESTRADA_BAIXO = 3;  
    public static final int ESTRADA_ESQUERDA = 4;  
    public static final int CRUZAMENTO_CIMA = 5;  
    public static final int CRUZAMENTO_DIREITA = 6;  
    public static final int CRUZAMENTO_BAIXO = 7;  
    public static final int CRUZAMENTO_ESQUERDA = 8;  
    public static final int CRUZAMENTO_CIMA_E_DIREITA = 9;  
    public static final int CRUZAMENTO_CIMA_E_ESQUERDA = 10;  
    public static final int CRUZAMENTO_DIREITA_E_BAIXO = 11;  
    public static final int CRUZAMENTO_BAIXO_E_ESQUERDA = 12;
```

▼  icons

-  icon0.png
-  icon1.png
-  icon2.png
-  icon3.png
-  icon4.png
-  icon5.png
-  icon6.png
-  icon7.png
-  icon8.png
-  icon9.png
-  icon10.png
-  icon11.png
-  icon12.png
-  icon-carro.png

```
case TiposCelula.ESTRADA_CIMA:  
    return getCelulaADireita(celula);
```

Malha

```
public class Malha {

    private Celula matrizMalha[][];
    private int qtdLinhas;
    private int qtdColunas;
    private Scanner matrizScanner;
    private static Malha instance;

    private Random random = new Random();

    private Malha() {
        this.inicializarVariaveis();
        this.inicializarMalha();
        this.printMatriz();
    }

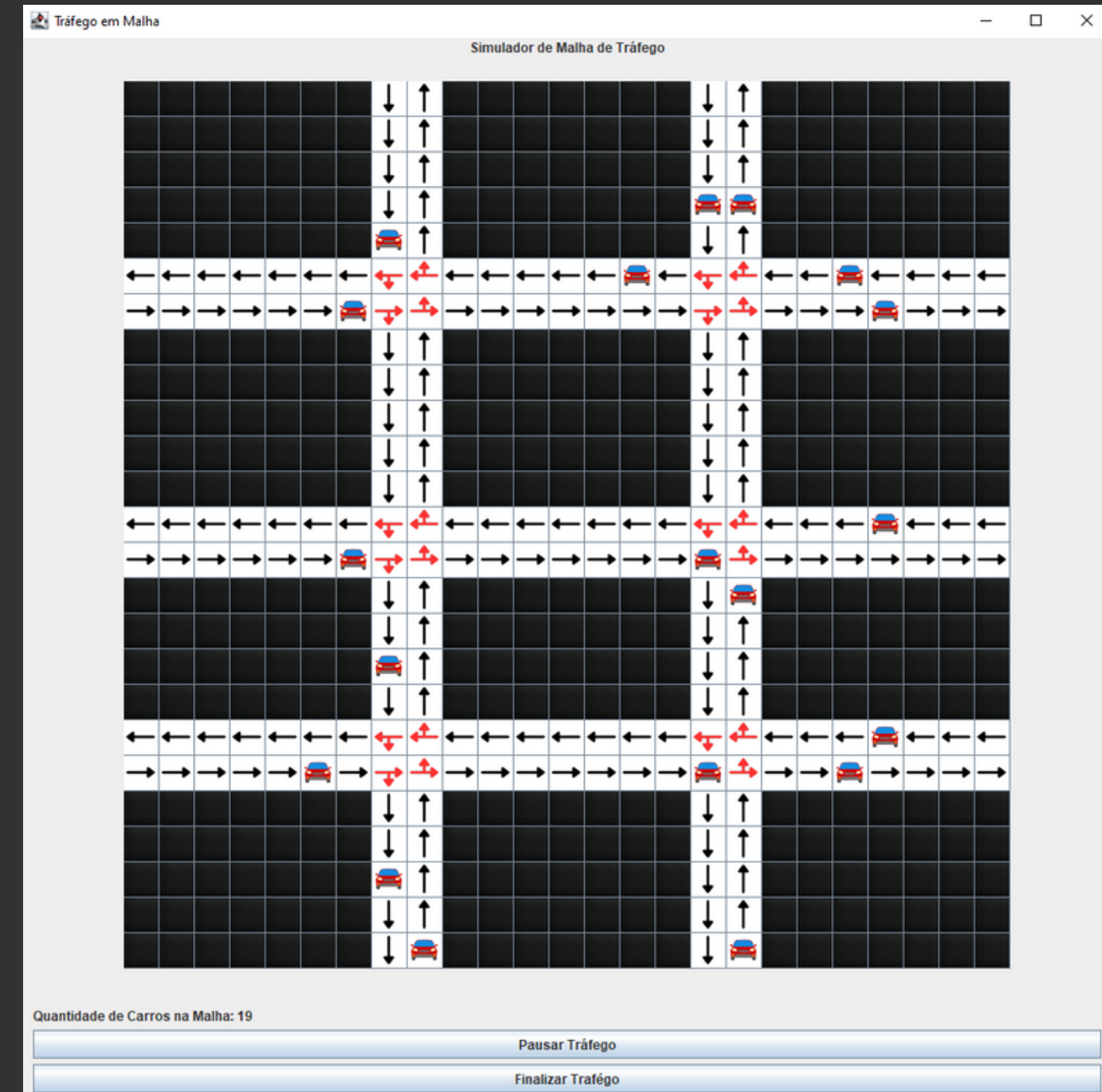
    public synchronized static Malha getInstance() {
        if (instance == null){
            reset();
        }
        return instance;
    }
}
```

```
private void inicializarVariaveis(){
    File arquivoMalha = new File(Config.getInstance().getMalhaAtual());
    try{
        matrizScanner = new Scanner(arquivoMalha);
        this.qtdLinhas = matrizScanner.nextInt();
        this.qtdColunas = matrizScanner.nextInt();
        this.matrizMalha = new Celula[qtdLinhas][qtdColunas];
    }catch (Exception e){
        System.out.println(e.getMessage()+" - "+ Arrays.toString(e.getStackTrace()));
    }
}

private void inicializarMalha(){
    while (matrizScanner.hasNextInt()){
        for (int linha = 0; linha < this.qtdLinhas; linha++) {
            for (int coluna = 0; coluna < this.qtdColunas; coluna++) {
                int tipo = matrizScanner.nextInt();
                Celula celulaAtual = new Celula(coluna, linha, tipo, qtdLinhas, qtdColunas);
                this.matrizMalha[linha][coluna] = celulaAtual;
            }
        }
    }
    matrizScanner.close();
}
```


Trafego View

```
public class TrafegoView extends JFrame implements Observer {  
  
    private MalhaController malhaController;  
  
    public TrafegoView() {  
        super( title: "Tráfego em Malha");  
        super.setSize(new Dimension( width: 1000, height: 1000));  
        super.setContentPane(this.containerPanel);  
        super.setExtendedState(JFrame.MAXIMIZED_BOTH);  
        super.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);  
  
        this.loadTableModel();  
        malhaController = new MalhaController();  
        malhaController.anexarObserver(this);  
        malhaController.start();  
    }  
}
```



Malha Controller

```
public class MalhaController extends Thread {

    private List<Carro> carrosEmCirculacao;
    private List<Observer> observers;

    public MalhaController() {
        this.carrosEmCirculacao = new ArrayList<>();
        this.observers = new ArrayList<>();
    }

    @Override
    public void run() {
        inicializar();
    }

    private void inicializar() {
        Config.getInstance().emExecucao = true;
        while (Config.getInstance().emExecucao){
            while (Config.getInstance().getSpawnerNovosCarros() && Config.getInstance().emExecucao){
                for (int linha = 0; linha < Malha.getInstance().getQtdLinhas(); linha++) {
                    for (int coluna = 0; coluna < Malha.getInstance().getQtdColunas(); coluna++) {
                        this.AtualizarCelula(linha,coluna);
                    }
                }
            }
            if (this.getQtdCarrosCirculacao() == 0)
                Config.getInstance().emExecucao = false;
        }
        encerrarSimulacao();
    }
}
```

```
private void AtualizarCelula(int linha, int coluna){
    Celula celulaAtual = Malha.getInstance().getMatrizMalha()[linha][coluna];
    if (!celulaAtual.getClassificacao().equals(ClassificacaoCelula.ENTRADA))
        return;
    if (!celulaAtual.celulaEstaVazia())
        return;
    if (this.getQtdCarrosCirculacao() == Config.getInstance().getQtdCarrosSimultaneos())
        return;
    try{
        Thread.sleep((long) (Config.getInstance().getIntervaloInsercao()* 1000));
        adicionarNovoCarroAMalha(celulaAtual);
    } catch (Exception e){
        System.out.println(e.getMessage()+" - "+ Arrays.toString(e.getStackTrace()));
    }
}

private void adicionarNovoCarroAMalha(Celula celulaInicial){
    Carro carro = new Carro( malhaController: this, celulaInicial);

    carrosEmCirculacao.add(carro);
    this.atualizarQuantidadeDeCarrosDaMalha();
    this.atualizarIconeDaCelula(celulaInicial);
    carro.printInformacoes();
    carro.start();
}

public void removerCarroDaMalha(Carro carro){
    this.carrosEmCirculacao.remove(carro);
    Celula celula = carro.getCelulaAtual();
    celula.setCarroAtual(null);
    this.atualizarIconeDaCelula(celula);
    this.atualizarQuantidadeDeCarrosDaMalha();
}
}
```

Carro

```
public class Carro extends Thread {

    private int tempoEspera; // em milissegundos
    private MalhaController malhaController;
    private Celula celulaAtual;
    private boolean finalizado = false;
    private final Random random = new Random();

    public Carro(MalhaController malhaController, Celula celulaAtual){
        this.malhaController = malhaController;
        this.celulaAtual = celulaAtual;
        this.tempoEspera = (random.nextInt( bound: 10)*100) + 200; // sorteio de 0,2 segundos a 1,2 segundos de espera
        celulaAtual.setCarroAtual(this);
    }

    @Override
    public void run() {
        while (Config.getInstance().emExecucao && !this.finalizado){
            Celula proximaCelula = Malha.getInstance().getProximaCelula(celulaAtual);

            // null = não há proxima celula
            if (proximaCelula == null)
                sairDaMalha();

            // se for cruzamento se lome na região critica
            else if (proximaCelula.getClassificacao().equals(ClassificacaoCelula.CRUZAMENTO))
                this.locomoverRegiaoCritica(proximaCelula);

            // se for estrada comum se lome diretamente para próxima célula
            else
                locomoverEstradaComum(proximaCelula);
        }
        this.finalizar();
    }
}
```

Locomoções



Estrada Comum



Cruzamentos

→ *Regiões Críticas*

Exclusão Mútua

```
public class Celula {  
    private Lock lock;  
    private Semaphore semaforo;
```

```
public boolean tentarReservar(){  
    if (this.carroAtual != null)  
        return false;  
    if (this.mecanismoExclusaoMutua.equals("Semaforo"))  
        return this.tentarReservarSemaforo();  
    else  
        return tentarReservarMonitor();  
}
```

```
public void liberar(){  
    if (this.mecanismoExclusaoMutua.equals("Semaforo"))  
        this.liberarSemaforo();  
    else  
        this.liberarMonitor();  
}
```

```
public boolean tentarReservarSemaforo(){  
    try{  
        return this.semaforo.tryAcquire( timeout: 100, TimeUnit.MILLISECONDS);  
    } catch (InterruptedException e) {  
        System.out.println(e.getStackTrace());  
        return false;  
    }  
}
```

```
public void liberarSemaforo(){  
    try{  
        this.semaforo.release();  
    } catch (Exception e){}
```

```
public boolean tentarReservarMonitor(){  
    try{  
        return this.lock.tryLock( time: 100, TimeUnit.MILLISECONDS);  
    } catch (InterruptedException e) {  
        System.out.println(e.getStackTrace());  
        return false;  
    }  
}
```

```
public void liberarMonitor(){  
    try{  
        this.lock.unlock();  
    } catch (Exception e){}
```

Estrada Comum

```
private void locomoverEstradaComum(Celula proximaCelula){
    boolean locomoveu = false;
    while (!locomoveu){
        if (proximaCelula.tentarReservar()){
            locomover(proximaCelula);
            locomoveu = true;
            proximaCelula.liberar();
        }else
            try {
                sleep( millis: 100 + random.nextInt( bound: 400));
            }
            catch (Exception e){
                System.out.println(e);
                System.out.println(e.getMessage());
            }
    }
}
```

```
public void locomover(Celula celulaDestino){
    this.aguardar();

    this.celulaAtual.setCarroAtual(null);
    this.malhaController.atualizarIconeDaCelula(celulaAtual);

    celulaDestino.setCarroAtual(this);
    this.celulaAtual = celulaDestino;
    this.malhaController.atualizarIconeDaCelula(celulaDestino);
}
```

Cruzamento

```
private void locomoverRegiaoCritica(Celula proximaCelula){

    LinkedList<Celula> rotaCruzamento = this.getRotaCruzamento(proximaCelula);
    boolean reservou = false;

    while (!reservou){
        LinkedList<Celula> celulasReservadas = new LinkedList<>();
        for (Celula celula : rotaCruzamento){
            if (!celula.tentarReservar()) {
                liberarCelulas(celulasReservadas);
                try {
                    sleep( millis: 100 + random.nextInt( bound: 1000));
                }catch (Exception e){
                    System.out.println(e);
                    System.out.println(e.getMessage());
                }
                break;
            }
            celulasReservadas.add(celula);
            reservou = celulasReservadas.size() == rotaCruzamento.size();
        }
    }
    andarNoCruzamento(rotaCruzamento);
}
```

```
private LinkedList<Celula> getRotaCruzamento(Celula proximaCelula){
    LinkedList<Celula> rota = new LinkedList<>();
    rota.add(proximaCelula);
    while (rota.getLast().getClassificacao().equals(ClassificacaoCelula.CRUZAMENTO)) {
        if (rota.size() >=3)
            proximaCelula = Malha.getInstance().getCelulaSaidaMaisProxima(proximaCelula);
        else
            proximaCelula = Malha.getInstance().getProximaCelula(proximaCelula);
        rota.add(proximaCelula);
    }
    return rota;
}
```

```
private void andarNoCruzamento(List<Celula> rota) {
    for (Celula celula : rota) {
        this.locomover(celula);
        celula.liberar();
    }
}

private synchronized void liberarCelulas(List<Celula> celulas){
    for (Celula celula : celulas){
        celula.liberar();
    }
}
```