Simulador de Tráfego em Malha Viária

65DSD • Desenvolvimento de Sistemas Paralelos e Distribuídos

Guilherme Furlan Nunes
Ramon José Pinto

Fernando dos Santos

65DSD • Desenvolvimento de Sistemas Paralelos e Distribuídos

Visão geral do trabalho

Simulador de Tráfego

Sistema distribuído e paralelo para simular um tráfego de veículos em uma malha viária.

Funcionalidades

- Seleção de malhas para simulação
- Configuração de parâmetros da simulação
- Iniciar, encerrar e finalizar inserção de veículos

Abordagem adotada

Aplicações de Threads e sistemas de exclusão mútua

Modelo da aplicação

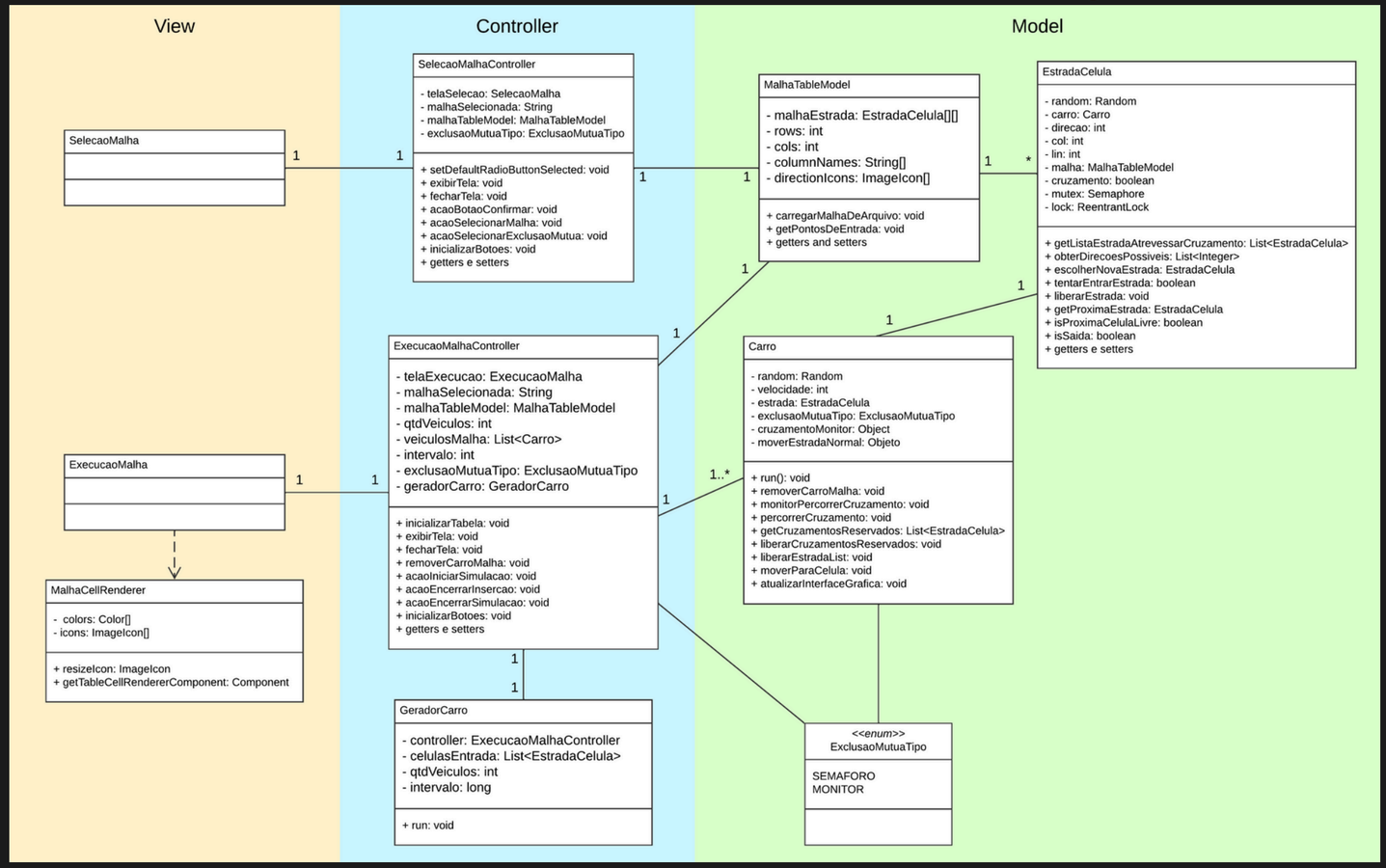
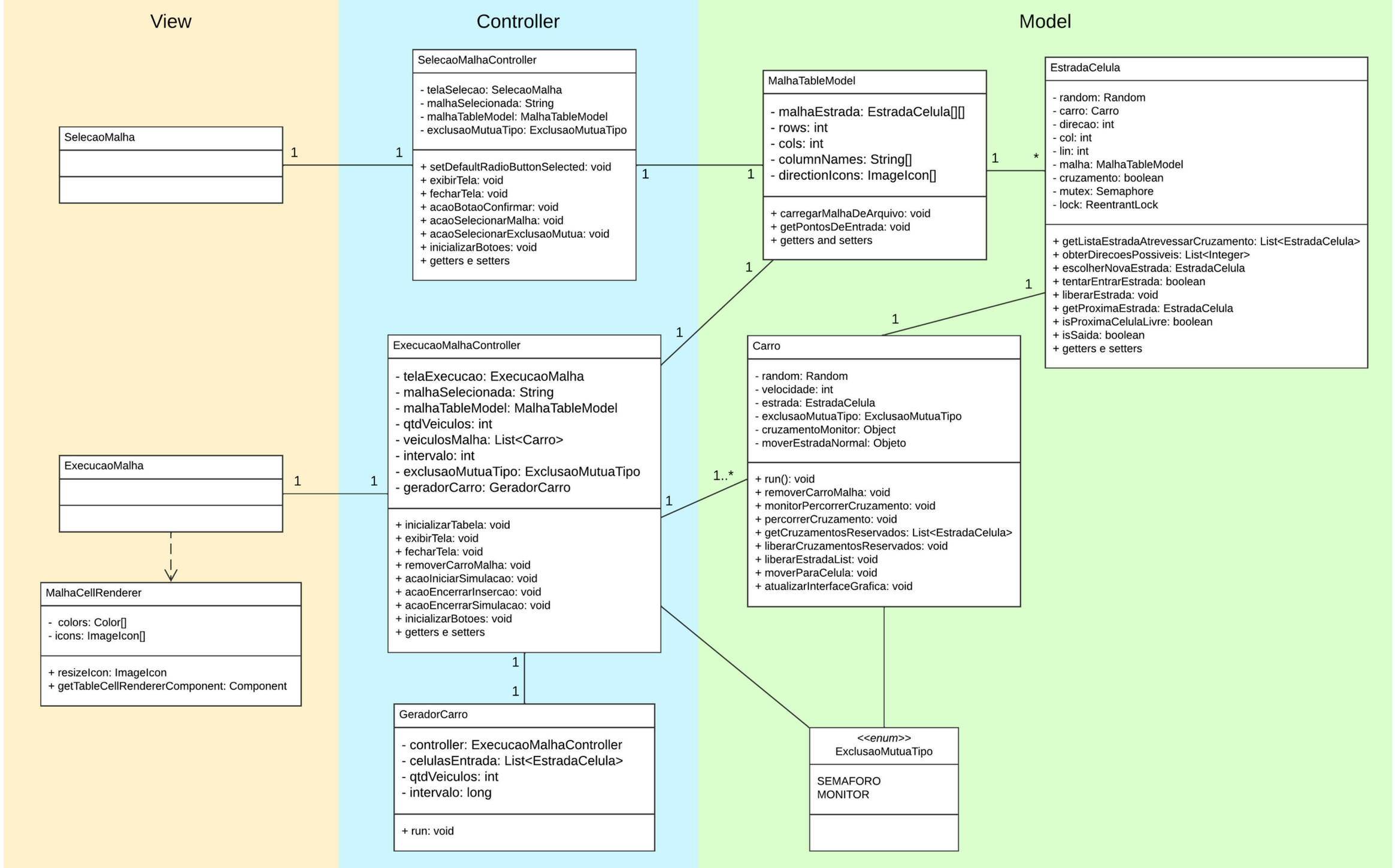
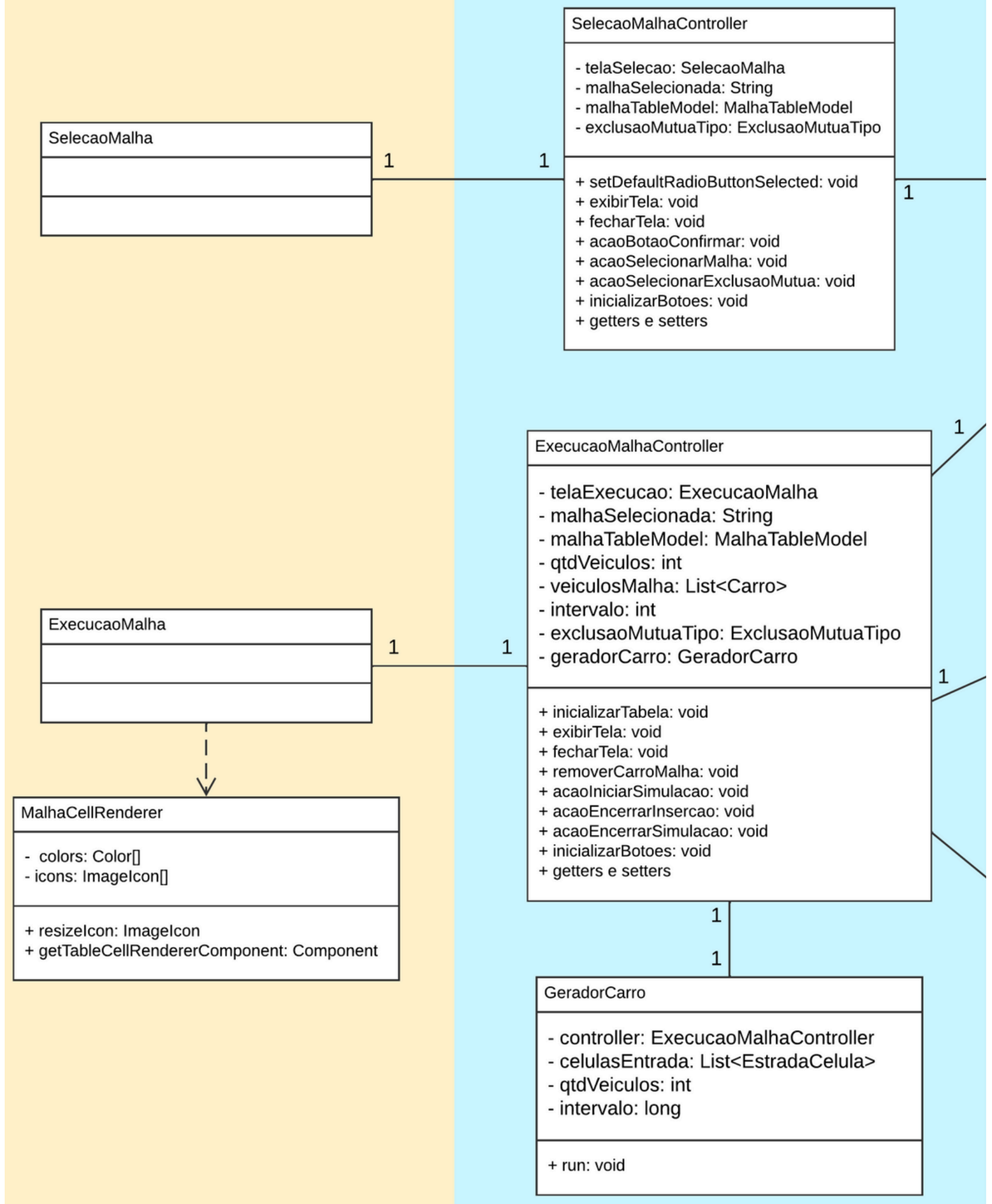
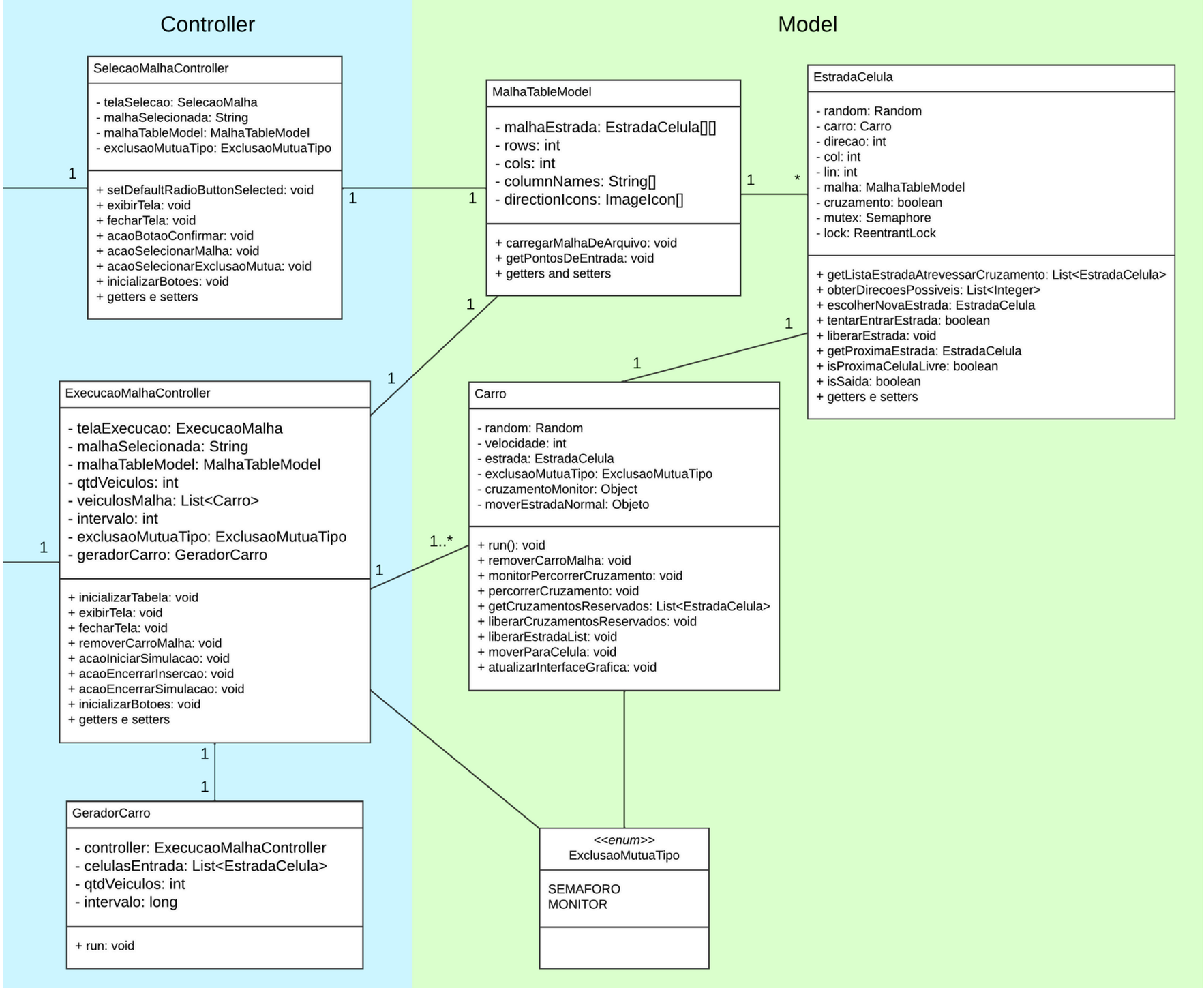


Figura 1 - Estrutura de MVC da aplicação


View
Controller
Model
SelecaoMalha
SelecaoMalhaController
- telaSelecao: SelecaoMalha
- malhaSelecionada: String
- malhaTableModel: MalhaTableModel
- exclusaoMutuaTipo: ExclusaoMutuaTipo
+ setDefaultRadioButtonSelected: void
+ exibirTela: void
+ fecharTela: void
+ acaoBotaoConfirmar: void
+ acaoSelecionarMalha: void
+ acaoSelecionarExclusaoMutua: void
+ inicializarBotoes: void
+ getters e setters
MalhaTableModel
- malhaEstrada: EstradaCelula[][]
- rows: int
- cols: int
- columnNames: String[]
- directionIcons: ImageIcon[]
+ carregarMalhaDeArquivo: void
+ getPontosDeEntrada: void
+ getters and setters
EstradaCelula
- random: Random
- carro: Carro
- direcao: int
- col: int
- lin: int
- malha: MalhaTableModel
- cruzamento: boolean
- mutex: Semaphore
- lock: ReentrantLock
+ getListaEstradaAtrevessarCruzamento: List<EstradaCelula>
+ obterDirecoesPossiveis: List<Integer>
+ escolherNovaEstrada: EstradaCelula
+ tentarEntrarEstrada: boolean
+ liberarEstrada: void
+ getProximaEstrada: EstradaCelula
+ isProximaCelulaLivre: boolean
+ isSaida: boolean
+ getters e setters
ExecucaoMalhaController
- telaExecucao: ExecucaoMalha
- malhaSelecionada: String
- malhaTableModel: MalhaTableModel
- qtdVeiculos: int
- veiculosMalha: List<Carro>
- intervalo: int
- exclusaoMutuaTipo: ExclusaoMutuaTipo
- geradorCarro: GeradorCarro
+ inicializarTabela: void
+ exibirTela: void
+ fecharTela: void
+ removerCarroMalha: void
+ acaoIniciarSimulacao: void
+ acaoEncerrarInsercao: void
+ acaoEncerrarSimulacao: void
+ inicializarBotoes: void
+ getters e setters
Carro
- random: Random
- velocidade: int
- estrada: EstradaCelula
- exclusaoMutuaTipo: ExclusaoMutuaTipo
- cruzamentoMonitor: Object
- moverEstradaNormal: Objeto
+ run(): void
+ removerCarroMalha: void
+ monitorPercorrerCruzamento: void
+ percorrerCruzamento: void
+ getCruzamentosReservados: List<EstradaCelula>
+ liberarCruzamentosReservados: void
+ liberarEstradaList: void
+ moverParaCelula: void
+ atualizarInterfaceGrafica: void
ExecucaoMalha
MalhaCellRenderer
- colors: Color[]
- icons: ImageIcon[]
+ resizeIcon: ImageIcon
+ getTableCellRendererComponent: Component
GeradorCarro
- controller: ExecucaoMalhaController
- celulasEntrada: List<EstradaCelula>
- qtdVeiculos: int
- intervalo: long
+ run: void
<<enum>>
ExclusaoMutuaTipo
SEMAFORO
MONITOR
1
1
1
1
1
*
1
1
1
1
1
1
1
1
1..*
1
1
1


SelecaoMalhaController
- telaSelecao: SelecaoMalha
- malhaSelecionada: String
- malhaTableModel: MalhaTableModel
- exclusaoMutuaTipo: ExclusaoMutuaTipo
+ setDefaultRadioButtonSelected: void
+ exibirTela: void
+ fecharTela: void
+ acaoBotaoConfirmar: void
+ acaoSelecionarMalha: void
+ acaoSelecionarExclusaoMutua: void
+ inicializarBotoes: void
+ getters e setters
SelecaoMalha
1
1
1
ExecucaoMalhaController
- telaExecucao: ExecucaoMalha
- malhaSelecionada: String
- malhaTableModel: MalhaTableModel
- qtdVeiculos: int
- veiculosMalha: List<Carro>
- intervalo: int
- exclusaoMutuaTipo: ExclusaoMutuaTipo
- geradorCarro: GeradorCarro
+ inicializarTabela: void
+ exibirTela: void
+ fecharTela: void
+ removerCarroMalha: void
+ acaoIniciarSimulacao: void
+ acaoEncerrarInsercao: void
+ acaoEncerrarSimulacao: void
+ inicializarBotoes: void
+ getters e setters
1
1
1
1
ExecucaoMalha
1
1
MalhaCellRenderer
- colors: Color[]
- icons: ImageIcon[]
+ resizeIcon: ImageIcon
+ getTableCellRendererComponent: Component
1
1
GeradorCarro
- controller: ExecucaoMalhaController
- celulasEntrada: List<EstradaCelula>
- qtdVeiculos: int
- intervalo: long
+ run: void


Controller
Model
SelecaoMalhaController
- telaSelecao: SelecaoMalha
- malhaSelecionada: String
- malhaTableModel: MalhaTableModel
- exclusaoMutuaTipo: ExclusaoMutuaTipo
+ setDefaultRadioButtonSelected: void
+ exibirTela: void
+ fecharTela: void
+ acaoBotaoConfirmar: void
+ acaoSelecionarMalha: void
+ acaoSelecionarExclusaoMutua: void
+ inicializarBotoes: void
+ getters e setters
MalhaTableModel
- malhaEstrada: EstradaCelula[][]
- rows: int
- cols: int
- columnNames: String[]
- directionIcons: ImageIcon[]
+ carregarMalhaDeArquivo: void
+ getPontosDeEntrada: void
+ getters and setters
EstradaCelula
- random: Random
- carro: Carro
- direcao: int
- col: int
- lin: int
- malha: MalhaTableModel
- cruzamento: boolean
- mutex: Semaphore
- lock: ReentrantLock
+ getListaEstradaAtrevessarCruzamento: List<EstradaCelula>
+ obterDirecoesPossiveis: List<Integer>
+ escolherNovaEstrada: EstradaCelula
+ tentarEntrarEstrada: boolean
+ liberarEstrada: void
+ getProximaEstrada: EstradaCelula
+ isProximaCelulaLivre: boolean
+ isSaida: boolean
+ getters e setters
ExecucaoMalhaController
- telaExecucao: ExecucaoMalha
- malhaSelecionada: String
- malhaTableModel: MalhaTableModel
- qtdVeiculos: int
- veiculosMalha: List<Carro>
- intervalo: int
- exclusaoMutuaTipo: ExclusaoMutuaTipo
- geradorCarro: GeradorCarro
+ inicializarTabela: void
+ exibirTela: void
+ fecharTela: void
+ removerCarroMalha: void
+ acaoIniciarSimulacao: void
+ acaoEncerrarInsercao: void
+ acaoEncerrarSimulacao: void
+ inicializarBotoes: void
+ getters e setters
Carro
- random: Random
- velocidade: int
- estrada: EstradaCelula
- exclusaoMutuaTipo: ExclusaoMutuaTipo
- cruzamentoMonitor: Object
- moverEstradaNormal: Objeto
+ run(): void
+ removerCarroMalha: void
+ monitorPercorrerCruzamento: void
+ percorrerCruzamento: void
+ getCruzamentosReservados: List<EstradaCelula>
+ liberarCruzamentosReservados: void
+ liberarEstradaList: void
+ moverParaCelula: void
+ atualizarInterfaceGrafica: void
GeradorCarro
- controller: ExecucaoMalhaController
- celulasEntrada: List<EstradaCelula>
- qtdVeiculos: int
- intervalo: long
+ run: void
<<enum>>
ExclusaoMutuaTipo
SEMAFORO
MONITOR
1
1
1
1
1
*
1
1
1
1
1
1..*
1
1
1
1

Demonstração de funcionamento do projeto...

Análise do código-fonte

Thread Spawner de Carros

```
@Override
public void run() {
    while(!this.isInterrupted()) {

        if (!(controller.getVeiculosMalha().size() == qtdVeiculos)) {

            for (int i = 0; i < qtdVeiculos; i++) {
                EstradaCelula estradaEntrada = celulasEntrada.get(index:new Random().nextInt(bound:celulasEntrada.size()));

                Carro carro = new Carro(estrada: estradaEntrada, exclusaoMutuaTipo:controller.getExclusaoMutuaTipo(), controller);
                estradaEntrada.tentarEntrarEstrada();
                estradaEntrada.setCarro(carro);
                controller.getVeiculosMalha().add(e: carro);

                estradaEntrada.getMalha().fireTableCellUpdated(row: estradaEntrada.getLin(), column:estradaEntrada.getCol());

                carro.start();
                carro.atualizarInterfaceGrafica();

                try {
                    Thread.sleep(millis: intervalo);
                } catch (Exception e) {
                    throw new RuntimeException(cause:e);
                }
            }

            this.interrupt();
        }

    }
}
```

```java
@Override
public void run() {
    try {
        Thread.sleep(velocidade);
        while (!estrada.isSaida() && !this.isInterrupted()) {

            if (estrada.getProximaEstrada().isCruzamento()) {
                try {
                    percorrerCruzamento();
                } catch (InterruptedException e) {
                    throw new RuntimeException(e);
                }
            } else if (estrada.isProximaCelulaLivre()) {
                moverParaCelula(estrada.getProximaEstrada(), true);
            }

            atualizarInterfaceGrafica();

            Thread.sleep(velocidade);
        }
    } catch (InterruptedException e) {
        throw new RuntimeException(e);
    }

    if (estrada.isSaida()) {
        if (!this.isInterrupted()) this.interrupt();
        removerCarroMalha();
    }
}
```

Thread Carro

```java
private void percorrerCruzamento() throws InterruptedException {
    EstradaCelula primeiraEstradaCruzamento = estrada.getProximaEstrada();

    if (primeiraEstradaCruzamento.isCruzamento()) {
        List<EstradaCelula> estradasAtravessarCruzamento = primeiraEstradaCruzamento.getListaEstradaAtrevessarCruzamento();

        List<EstradaCelula> estradasCruzamentoReservados = null;

        if(exclusaoMutuaTipo == ExclusaoMutuaTipo.MONITOR){
            estradasCruzamentoReservados = monitorPercorrerCruzamento(estradasAtravessarCruzamento);
        } else {
            estradasCruzamentoReservados = getCruzamentosReservados(estradasAtravessarCruzamento);
        }

        if (estradasAtravessarCruzamento.size() == estradasCruzamentoReservados.size()) {
            for (EstradaCelula e : estradasAtravessarCruzamento) {
                moverParaCelula(e, false);
                if (e.isCruzamento()) {
                    atualizarInterfaceGrafica();
                    Thread.sleep(this.velocidade);
                }
            }
        }
    }
}
```

Thread Carro

```java
public List<EstradaCelula> monitorPercorrerCruzamento(List<EstradaCelula> estradasAtravessarCruzamento) {
    List<EstradaCelula> cruzamentosReservados = new ArrayList<>();
    for (EstradaCelula e : estradasAtravessarCruzamento) {
        if (e.getLock().tryLock()) {
            cruzamentosReservados.add(e);
        } else {
            liberarCruzamentosReservados(cruzamentosReservados);
            break;
        }
    }
    return cruzamentosReservados;
}
```

Thread Carro

```java
private void moverParaCelula(EstradaCelula est, boolean testar) {
    boolean reservado = false;

    if (testar) {
        try {
            do {
                if (exclusaoMutuaTipo == ExclusaoMutuaTipo.MONITOR) {
                    // MONITOR
                    if (est.getLock().tryLock()) {
                        reservado = true;
                    } else {
                        sleep(random.nextInt(500));
                    }
                } else {
                    // SEMAFORO
                    if (est.tentarEntrarEstrada()) {
                        reservado = true;
                    } else {
                        sleep(random.nextInt(500));
                    }
                }
            } while (!reservado);
        } catch (InterruptedException e) {
            throw new RuntimeException(e);
        }
    }

    estrada.setCarro(null);
    est.setCarro(this);

    if (exclusaoMutuaTipo == ExclusaoMutuaTipo.MONITOR && estrada.getLock().isHeldByCurrentThread()) {
        estrada.getLock().unlock();
    } else if(exclusaoMutuaTipo == ExclusaoMutuaTipo.SEMAFORO) {
        estrada.liberarEstrada();
    }

    estrada = est;
}
```

Thread Carro

```java
public class EstradaCelula {
    private Random random;
    private Carro carro;
    private int direcao;
    private int col;
    private int lin;
    private MalhaTableModel malha;
    private boolean cruzamento;
    private Semaphore mutex;
    private final ReentrantLock lock = new ReentrantLock(); // MONITOR
```

```java
public boolean tentarEntrarEstrada(){
    return mutex.tryAcquire();
}

public void liberarEstrada(){
    mutex.release();
}
```

```java
public ReentrantLock getLock() {
    return lock;
}
```

Model EstradaCelula

Simulador de Tráfego em Malha Viária

65DSD • Desenvolvimento de Sistemas Paralelos e Distribuídos