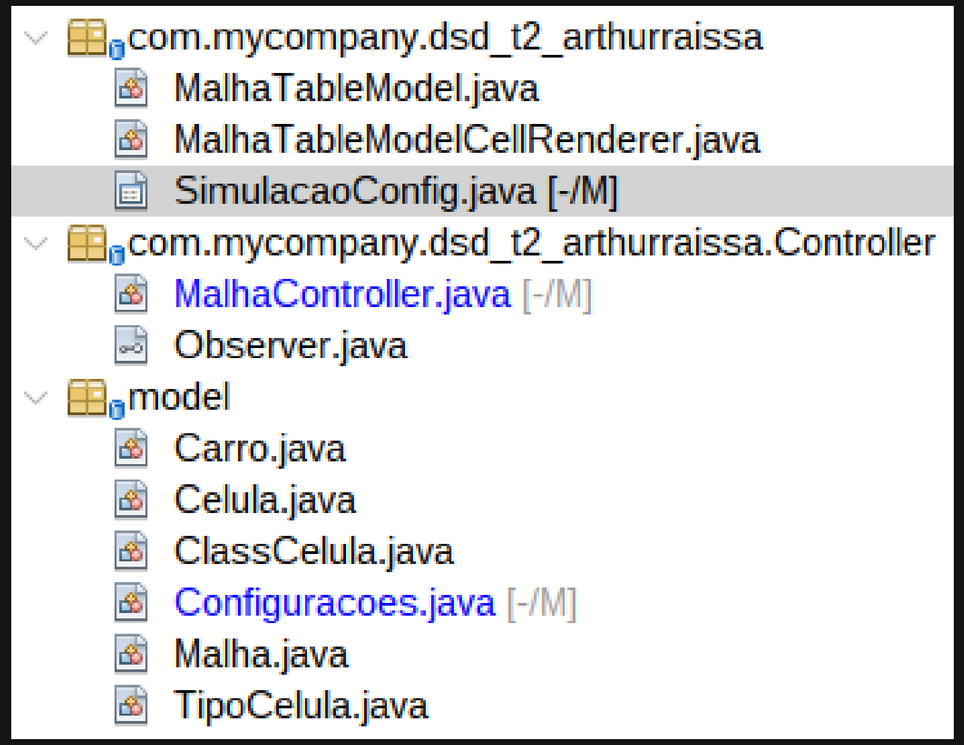
Simulador de Tráfego em Malha Viária

65DSD: T2

Arthur Espindola da Cruz
Raissa Duarte

MVC*



Model:

- Carro
- Celula
- ClassCelula
- ...

Controller:

- MalhaController
- Observer

View:

- MalhaTabelModel(Info Gerais Malha e valor específico)
- MalhaTabelModelCellRenderer(Definição Icone Malha)
- SimulacaoConfig

Observer -Malha Controle

Possui 3 Métodos:

- atuIconeDaCelula,
- atuQuantidadeCarrosMalha
- Encerrar

Servindo respectivamente para:

- Atualizar os Ícones da célula, na entrada e saída de veículos
- Atualiza a Quantidade de Carros no sistema, na entrada e na saída de veículos
- Encerrar as rotinas (os veículos permanecem no sistema até a hora da próxima movimentação)

```java
public interface Observer {
    void atuIconeDaCelula(Celula celula);
    void atuCarrosNaMalha(int qtdCarrosMalha);
    void encerrar();
}
```

SimulacaoConfig.java [-/M] × Configuracoes.java [-/M] × MalhaTableModel.java × MalhaTableMod

sign History

```java
@Override
public void atuIconeDaCelula(Celula celula) {
    MalhaTableModel malhaTableModel = (MalhaTableModel) table.getModel();
    malhaTableModel.fireTableCellUpdated(celula.getLinha(), celula.getColuna());
    malhaTableModel.fireTableDataChanged();
}
```

MalhaController.java × SimulacaoConfig.java [-/M] × Configuracoe

story

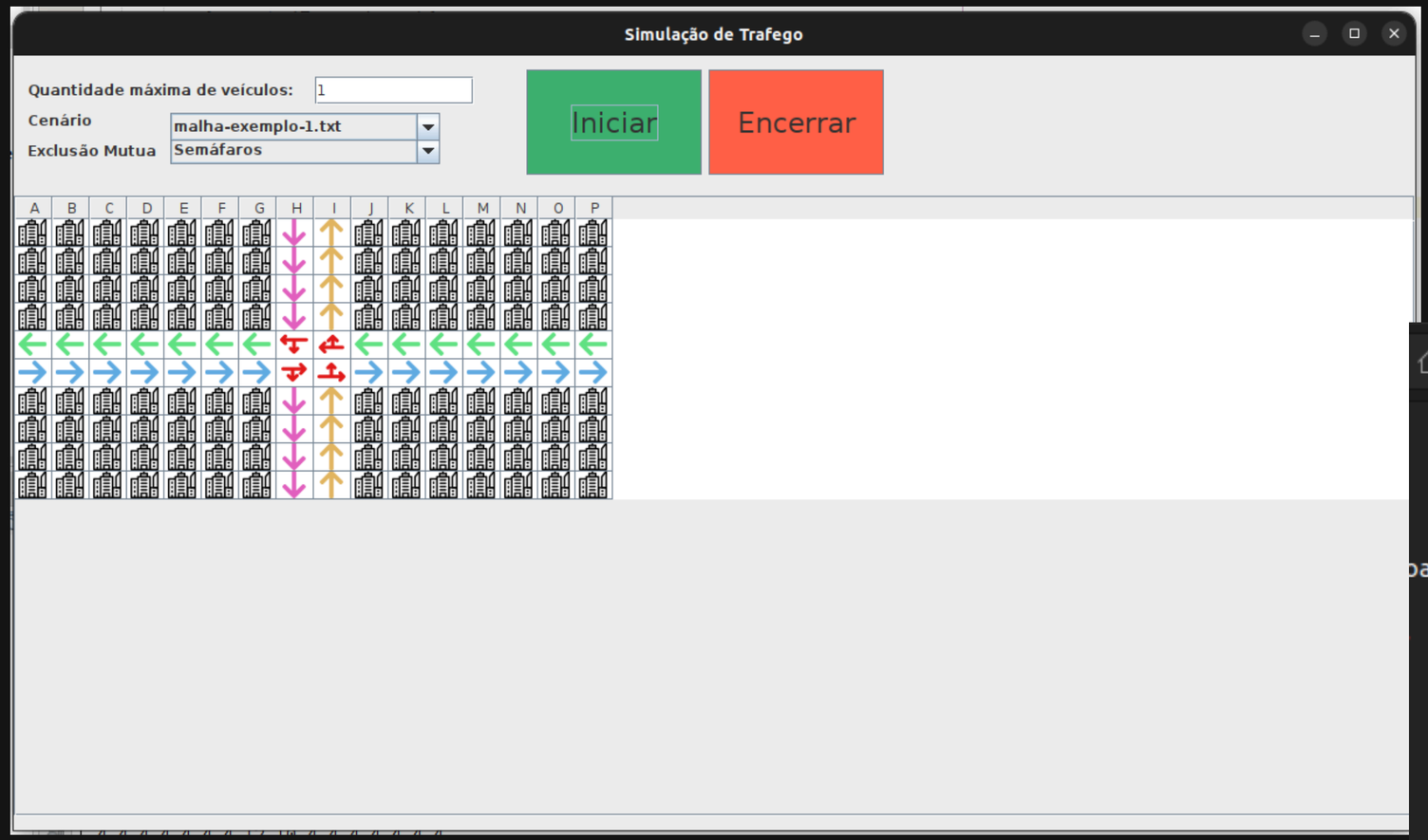
```java
public void atualizarQuantidadeDeCarrosDaMalha(){
    for (Observer obs: observers){
        obs.atuCarrosNaMalha(this.getQtdCarrosCirculacao());
    }
}
```

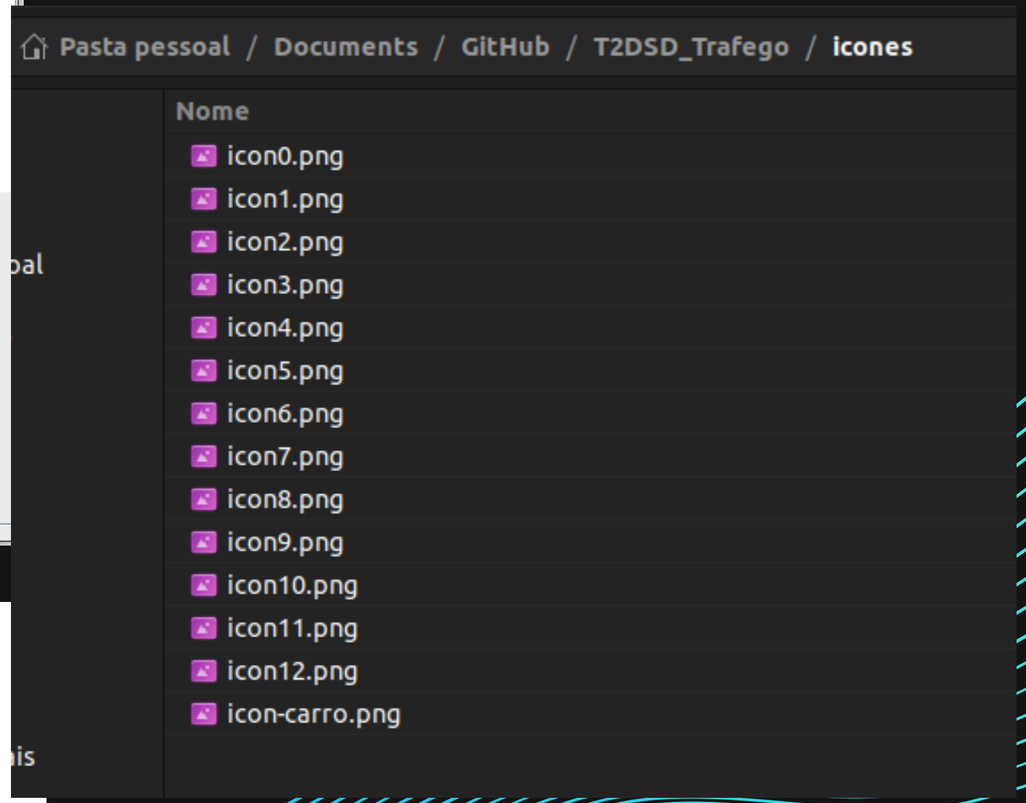
MalhaController.java × SimulacaoCo

story

```java
public void encerrarSimulacao(){
    for (Observer obs: observers){
        obs.encerrar();
    }
}
```

Preenchendo a Malha






```java
public String getIcon(){
    if (this.carroAtual != null) {
        return Configuracoes.ICONS_PATH + "icon-carro.png";
    }
    else {
        return Configuracoes.ICONS_PATH + "icon" + this.tipo + ".png";
    }
}
```

MalhaController

Lógica para o bom funcionamento do Sistema

- run(): Inicializa a execução da simulação.
- inicializar(): Inicializa a simulação e controla sua execução.
- AtualizarCelula(int linha, int coluna): Verifica e atualiza uma célula da malha.
- adicionarNovoCarroAMalha(Celula celulaInicial): Adiciona um novo carro à malha.
- removerCarroDaMalha(Carro carro): Remove um carro da malha.
- anexarObserver(Observer observer): Adiciona um observador à lista de observadores.
- getQtdCarrosCirculacao(): Retorna a quantidade de carros em circulação.
- atualizarIconeDaCelula(Celula celula): Notifica os observadores sobre a atualização do ícone da célula.
- atualizarQuantidadeDeCarrosDaMalha(): Notifica os observadores sobre a quantidade de carros na malha.
- encerrarSimulacao(): Encerra a simulação e notifica os observadores.

ReservarPosição e LiberarPosição

No nosso exemplo o semáforo está permitindo somente 1 Thread por vez

Dessa forma, a lógica segue para os dois tentando "lockar" a area critica para poder se movimentar

Carro

Ao chegar na Região critica, o carro tenta resever/lockar todas as possições necessárias para chegar no destino, caso não consiga ele espera 100 a 1100 milisegundos e tenta novamente

Ao locomover em Estrada Comum na Região critica, o carro tenta resever/lockar a próxima célula para chegar no destino, caso não consiga ele espera 100 a 500 milisegundos e tenta novamente

Carro.java [-/M] × | MalhaTableModelCellRenderer.java × | Observer.java × | App.java × | ClassCelula.jav

History

```
    }

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


            if (proximaCelula == null)
                sairDaMalha();


            else if (proximaCelula.getClassificacao().equals(ClassCelula.CRUZAMENTO))
                this.locomoverRegiaoCritica(proximaCelula);


            else
                locomoverEstradaComum(proximaCelula);
        }
        this.finalizar();
    }
```

```
    private void locomoverRegiaoCritica(Celula proximaCelula){

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

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

Dificuladades Encontradas

Algumas vezes mesmo com o número de carros o sistema não o mantém

Popular a Malha com icones sugestivos

Estruturação do Projeto, como cada classe deveria se comportar/comunicar com outra

Tentamos fazer em MVC, mas devido a complexidade do problema, algumas classes e lógicas ficaram fora de posição