Trabalho 2

Simulador de Tráfego em Malha Viária

João Elias Engelbrecht Kayser
Desenv. de Sist. Paralel. e Distrib.
Professor Fernando dos Santos
12/2021

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA

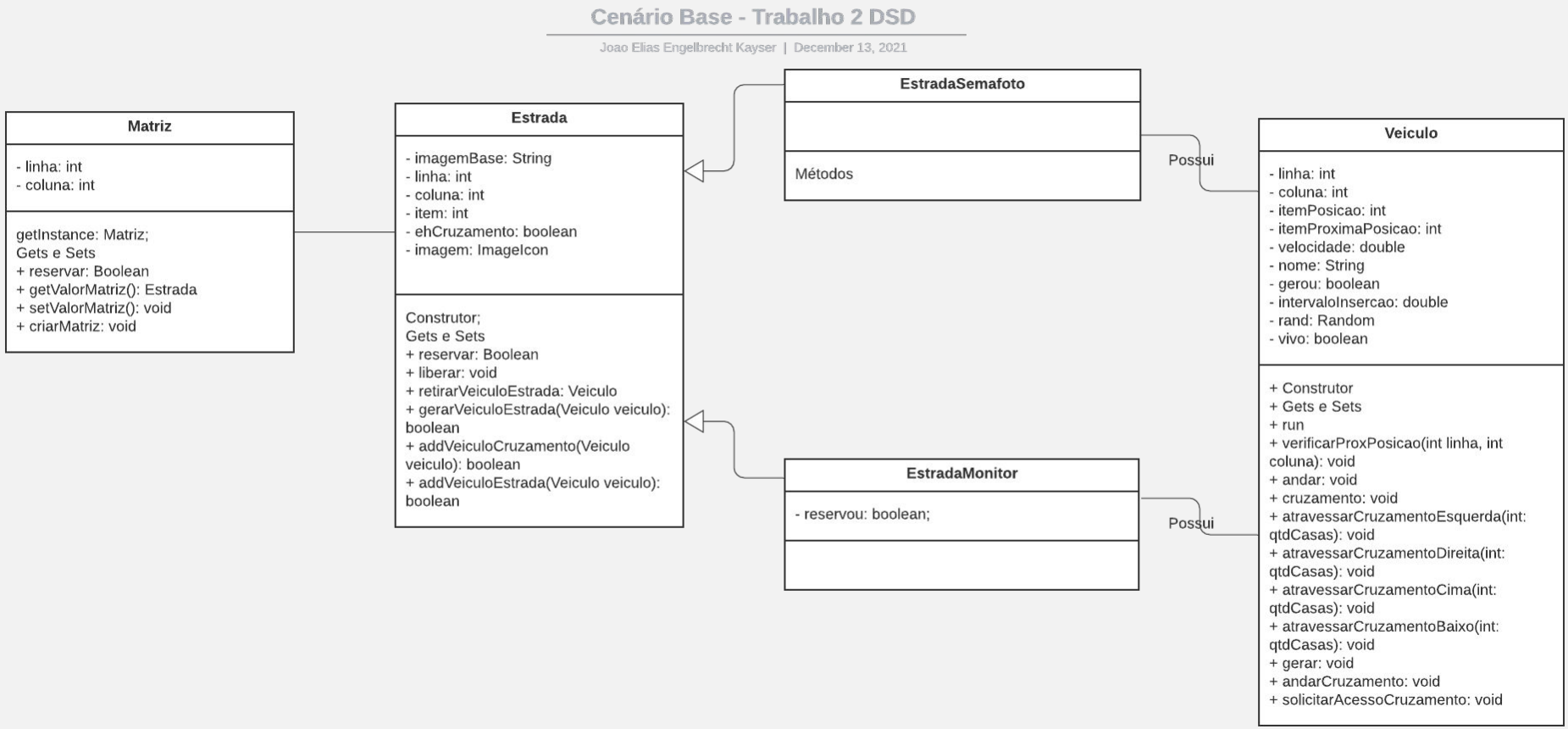
Roteiro de Apresentação

- Introdução;
- Diagrama de Classes;
- Técnicas Utilizadas;
- Comprovação dos Requisitos;
- Problemas e Soluções;
- Trabalho na Prática;
- Conclusão.

Introdução

O presente trabalho demonstra de forma prática a utilização de Semáforos, Monitores e Thread para uma Simulação de Tráfego em Malha Viária.

Diagrama de Classes



UDESC

Técnicas Utilizadas

- ExecutorService
- Observer;
- MVC.

Comprovação dos Requisitos

- Cada veículo deve ser um thread.

```
 */
public class Veiculo extends Thread {
```

Comprovação dos Requisitos

- O veículo se movimenta pela malha, uma posição por vez, respeitando o sentido de fluxo da pista. O veículo só pode se mover caso a posição à frente esteja livre.

```
public void andar() {
    switch (itemPosicao) {
        case 1:
            verificarProxPosicao(linha - 1, coluna);
            if (vivo) {
                if (matriz.getValorMatriz(linha - 1, coluna).getItem() <= 4 && !matriz.getValorMatriz(linha - 1, coluna).estaOcupado()) {
                    matriz.getValorMatriz(linha - 1, coluna).addVeiculoEstrada(matriz.getValorMatriz(linha, coluna).retirarVeiculoEstrada());
                } else {
                    cruzamento();
                }
            }
            break;
```

```
@Override
public boolean estaOcupado() {
    return veiculo != null;
}
```

UDESC

Comprovação dos Requisitos

- Ao se deparar com um cruzamento: Deve escolher, aleatoriamente, uma das vias de saída do cruzamento para seguir viagem. A escolha deve ser feita antes do veículo ingressar no cruzamento

```
163     private void cruzamento() {
164         List<Estrada> estradas = new ArrayList<>();
165         Estrada selecionada;
166         switch (itemProximaPosicao) {
167             case 5:
168                 estradas.add(matriz.getValorMatriz(linha - 2, coluna - 2)); // ir pra esquerda
169                 estradas.add(matriz.getValorMatriz(linha - 3, coluna)); // ir pra direita
170                 selecionada = estradas.get(rand.nextInt(2)); //escolhe aleatoriamente
171                 if (selecionada.getItem() == estradas.get(0).getItem()) {
172                     atravessarCruzamentoEsquerda(3); //3 = numero de casas que vai andar
173                 } else {
174                     atravessarCruzamentoCima(2);
175                 }
176                 break;
```

Comprovação dos Requisitos

- Só deve se mover pelo cruzamento se todas as posições por onde vai passar estiverem totalmente livres (exclusão mútua).

```java
private List<Estrada> solicitarAcessoCruzamento(List<Estrada> estradasParaMover) {
    boolean reservou = false;
    List<Estrada> estradasReservadas = new ArrayList<>();
    for (Estrada estrada : estradasParaMover) {
        try {
            reservou = estrada.reservar();
        } catch (Exception ex) {

        }
        if (!reservou) {
            //Liberar todos que tentou acessar
            for (Estrada estradaRemover : estradasReservadas) {
                estradaRemover.liberar();
            }
            estradasReservadas.clear();
            break;
        } else {
            estradasReservadas.add(estrada);
            reservou = false;
        }
    }
    return estradasReservadas;
}
```

```java
@Override
public boolean reservar() {
    if (!estaOcupado()) {
        this.veiculo = new Veiculo(linha, coluna, item, linha, coluna, linha);
        return true;
    }
    return false;
}
```

UDESC

Comprovação dos Requisitos

- Não deve bloquear o cruzamento de outros veículos (ficar parado no cruzamento).

```
@Override
public synchronized boolean addVeiculoCruzamento(Veiculo veiculo) {
    boolean adicionou = false;
    try {
        if (veiculo != null) {
            this.imagem = new ImageIcon("assets/veiculo.png");
            veiculo.setColuna(this.coluna);
            veiculo.setLinha(this.linha);
            veiculo.setItemPosicao(this.item);
            this.veiculo = veiculo;
            adicionou = true;
        }
    } catch (Exception e) {
        e.printStackTrace();
    }
    return adicionou;
}
```

Comprovação dos Requisitos

- Novos veículos são inseridos nos pontos de entrada da malha (ver especificação da malha)

```java
private void gerar() {
    boolean adicionou = false;
    while (!adicionou) {
        try {
            adicionou = cdc.gerar(this);
            sleep((long) this.intervaloInsercao);
        } catch (Exception ex) {
            Logger.getLogger(Veiculo.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    gerou = true;
    Gerenciador ger = Gerenciador.getInstance();
    ger.gerarVeiculos();
}
```

```java
public boolean gerar(Veiculo veiculo) throws Exception {
    int orientacao = rand.nextInt(4);
    boolean adicionou = false;
    switch (orientacao) {
        case 0:
            adicionou = nascerSul(veiculo);
            break;
        case 1:
            adicionou = nascerOeste(veiculo);
            break;
        case 2:
            adicionou = nascerNorte(veiculo);
            break;
        case 3:
            adicionou = nascerLeste(veiculo);
            break;
    }
    Gerenciador ger = Gerenciador.getInstance();
    ger.notificarEstradaAlterada();
    return adicionou;
}
```

UDESC

Comprovação dos Requisitos

- Ao atingir um ponto de saída (ver especificação da malha), o veículo deve ser encerrado.

```
public void verificarProxPosicao(int linha, int coluna) {
    try {
        itemProximaPosicao = matriz.getValorMatriz(linha, coluna).getItem();
    } catch (ArrayIndexOutOfBoundsException ex) {
        matriz.getValorMatriz(this.linha, this.coluna).retirarVeiculoEstrada();
        this.vivo = false;
        Gerenciador ger = Gerenciador.getInstance();
        ger.verificarFim();
        ger.gerarVeiculos();

    }
}
```

Comprovação dos Requisitos

- Veículos possuem velocidades de movimentação diferente (tempo de sleep a cada passo).

```
private boolean nascerSul(Veiculo veiculo) throws Exception {
    List<Integer> posicoes = new ArrayList<>();
    for (int i = 0; i < matriz.getColuna(); i++) {
        if (matriz.getValorMatriz(matriz.getLinha() - 1, i).getItem() == 1) {
            posicoes.add(i);
        }
    }
    int colunaNascer = rand.nextInt(posicoes.size());
    boolean add = matriz.getValorMatriz(matriz.getLinha() - 1, posicoes.get(colunaNascer)).addVeiculoEstrada(veiculo);
    if(add){
        veiculo.setVelocidade(geradorVelocidade());
        addVeiculo(veiculo);
    }
    return add;
```

```
private void andarCruzamento(List<Estrada> estradasReservadas) {
    for (Estrada estradaAdicionar : estradasReservadas) {
        matriz.getValorMatriz(estradaAdicionar.getLinha(), estradaAdicionar.getColuna()).addVeiculoCruzamento(matriz.getValorMatriz(1
        try {
            sleep((long) velocidade);
        } catch (InterruptedException ex) {
            Logger.getLogger(Veiculo.class.getName()).log(Level.SEVERE, null, ex);
        }
        Gerenciador ger = Gerenciador.getInstance();
        ger.notificarEstradaAlterada();
    }
}
```

UDESC

Comprovação dos Requisitos

- Deve ser carregada de um arquivo texto

```
public class Leitor {

    //Lê a matriz e cria o mapa que a JTable irá ler
    public void lerMatriz(File arquivo, int modo) throws FileNotFoundException {
        Scanner in = new Scanner(arquivo);
        Estrada estrada = new Estrada();
        int linha = Integer.parseInt(in.next().trim());
        int coluna = Integer.parseInt(in.next().trim());
        System.out.println("Linha: " + linha + " Coluna: " + coluna);
        Matriz matriz = Matriz.getInstance();
        matriz.criarMatriz(linha, coluna);
        matriz.setLinha(linha);
        matriz.setColuna(coluna);
//          if (modo == 1) {
```

UDESC

Comprovação dos Requisitos

- Nas duas primeiras linhas estão a quantidade de linhas e colunas da malha, respectivamente.

```
public class Leitor {

    //Lê a matriz e cria o mapa que a JTable irá ler
    public void lerMatriz(File arquivo, int modo) throws FileNotFoundException {
        Scanner in = new Scanner(arquivo);
        Estrada estrada = new Estrada();
        int linha = Integer.parseInt(in.next().trim());
        int coluna = Integer.parseInt(in.next().trim());
        System.out.println("Linha: " + linha + " Coluna: " + coluna);
        Matriz matriz = Matriz.getInstance();
        matriz.criarMatriz(linha, coluna);
        matriz.setLinha(linha);
        matriz.setColuna(coluna);
//          if (modo == 1) {
```

Comprovação dos Requisitos

- Identificação de pontos de entrada e de saída de veículos:

```
public boolean gerar(Veiculo veiculo) throws Exception {
    int orientacao = rand.nextInt(4);
    boolean adicionou = false;
    switch (orientacao) {
        case 0:
            adicionou = nascerSul(veiculo);
            break;
        case 1:
            adicionou = nascerOeste(veiculo);
            break;
        case 2:
            adicionou = nascerNorte(veiculo);
            break;
        case 3:
            adicionou = nascerLeste(veiculo);
            break;
    }
    Gerenciador ger = Gerenciador.getInstance();
    ger.notificarEstradaAlterada();
    return adicionou;
}
```

UDESC

Comprovação dos Requisitos

- Limitar quantidade de veículos:

```
public void gerarVeiculos() {
    if ((veiculosNoMundo < quantidadeDeVeiculos) && emAndamento) {
        Veiculo Veiculo = new Veiculo(0, 0, 0, 0, quantidadeDeVeiculos, intervaloInsercao);
        executor.execute(Veiculo);
        VeiculosGerados++;
        veiculosNoMundo++;
    }
}
```

Comprovação dos Requisitos

- Usuário informar um intervalo de inserção de veículos

```
private void gerar() {
    boolean adicionou = false;
    while (!adicionou) {
        try {
            adicionou = cdc.gerar(this);
            sleep((long) this.intervaloInsercao);
        } catch (Exception ex) {
            Logger.getLogger(Veiculo.class.getName()).log(Level.SEVERE, null, ex);
        }
    }
    gerou = true;
    Gerenciador ger = Gerenciador.getInstance();
    ger.gerarVeiculos();
}
```

Comprovação dos Requisitos

- Iniciar simulação

```
public void iniciarSimulacao(int qtdVeiculos, double intervalo, int modo) {
    this.quantidadeDeVeiculos = qtdVeiculos;
    this.intervaloInsercao = intervalo;
    this.modo = modo;
    this.emAndamento = true;
    matriz.setMatriz(null);
    lerMatriz(modo);

    notificarEstradaAlterada();

    gerarVeiculos();
}
```

Comprovação dos Requisitos

- Encerrar simulação

```
public void encerrarSimulacaoImediatamente(){
    emAndamento = false;
    matriz.setMatriz(null);
    lerMatriz(modo);
    notificarEstradaAlterada();
    for (Observer.Observer obs : observadores) {
        obs.notificarSimulacaoEncerrada();
    }
}
```

Comprovação dos Requisitos

- Encerrar e aguardar veículos saírem da malha.

```
public void encerrarSimulacao(){
    emAndamento = false;
    for (Observer.Observer obs : observadores) {
        obs.notificarSimulacaoEncerrada();
    }
}
```

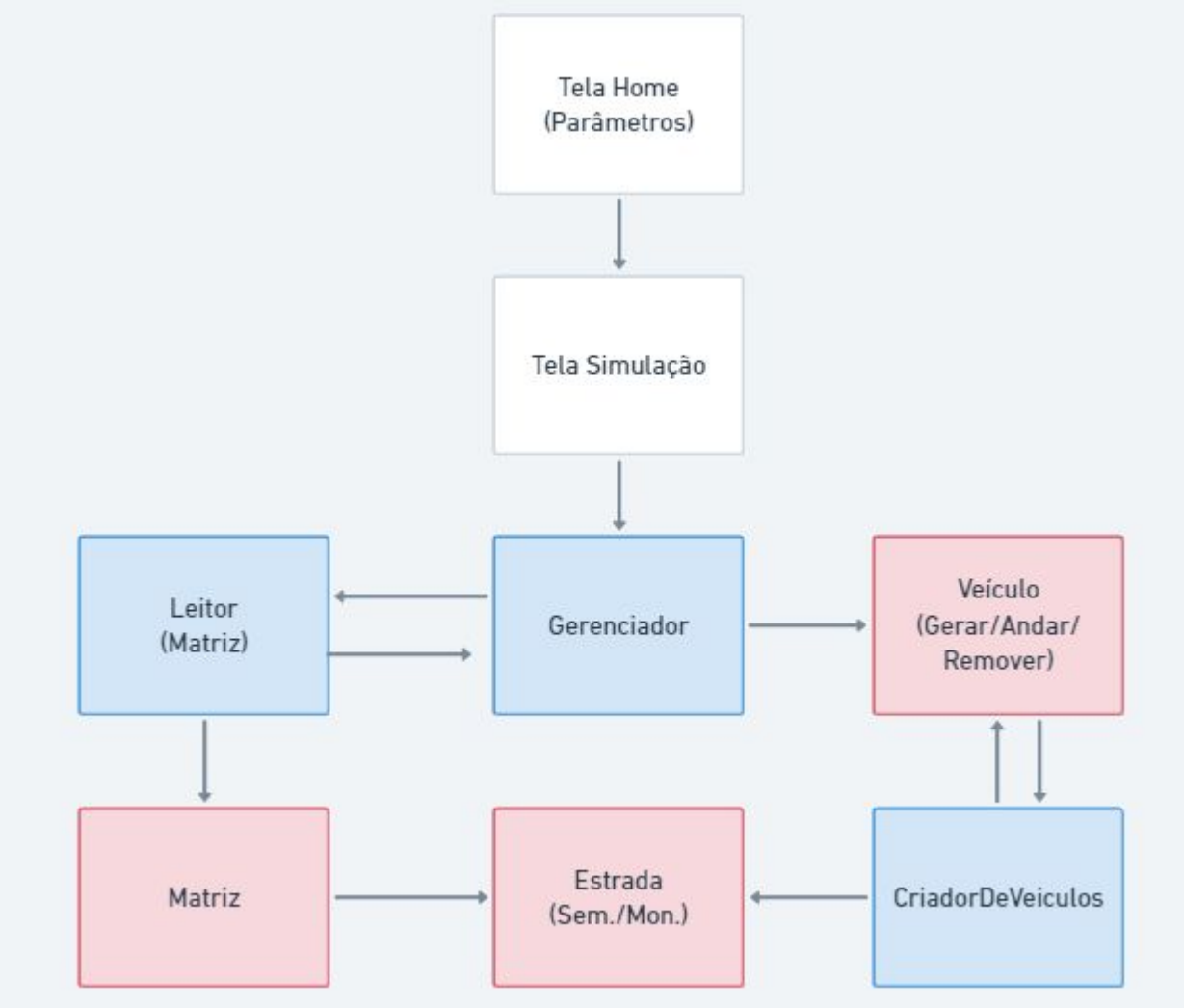
Comprovação dos Requisitos

- Deve suportar tanto semáforos quanto monitores

Problemas

- Interface Gráfica;
- Via Dupla;
- Debug.

Prática



UDESC

Prática

Conclusão

Semáforos e Monitores são ótimas ferramentas para controlar o desenvolvimento de sistemas paralelos realizando a consistência da regra de negócio da aplicação.

Trabalho 2

Simulador de Tráfego em Malha Viária

João Elias Engelbrecht Kayser
Desenv. de Sist. Paralel. e Distrib.
Professor Fernando dos Santos
12/2021

UDESC
UNIVERSIDADE
DO ESTADO DE
SANTA CATARINA