

# Simulador de Coleta de Lixo

Sávio Macêdo e Bruno Moraes

TURMA ADA

# Visão Geral

O sistema é modelado como uma simulação baseada em eventos discretos, onde as operações (coleta, transferência, descarregamento) são representadas por eventos agendados e processados em ordem cronológica. A simulação ocorre em dias, com cada dia incluindo geração de lixo, coleta, transferência e descarregamento.

## Tecnologias e Recursos utilizados

- JAVA
- GIT
- Programação Orientada a Objetos (POO)
- Estruturas de Dados Personalizadas
- Simulação Baseada em Eventos

# Estruturas

## Fila

- A classe `Fila<T>` representa uma estrutura de dados do tipo **Fila (FIFO - First In, First Out)**.
- Ela foi usada no projeto para montar a estrutura de fila de Caminhões na Estação de Transferência

```
103 public void printQueue() {  
104     if (head == null) {  
105         System.out.println("Fila vazia");  
106         return;  
107     }  
108     No<T> atual = head;  
109     while (atual != null) {  
110         System.out.print(atual.valor + " -> ");  
111         atual = atual.prox;  
112     }  
113     System.out.println("[EXIT]");  
114 }  
115  
116 /**  
117  * Retorna o número de elementos na fila.  
118  *  
119  * @return o tamanho da fila  
120  */  
121 public int size() {  
122     return tamanho;  
123 }  
124 }  
125
```

# Estruturas

## Fila Duplamente Encadeada

- Estrutura de dados do tipo **Fila (FIFO)** com nós **duplamente encadeados**.
- Utiliza a classe `No<T>` de uma lista duplamente encadeada (possui `prev` e `prox`).
- Permite inserção no final e remoção no início com eficiência.

```
27 public void enfileirar(T valor) {
28     No<T> novoNo = new No<>(valor);
29
30     if (estaVazia()) {
31         inicio = novoNo;
32         fim = novoNo;
33     } else {
34         fim.setProx(novoNo);
35         novoNo.setPrev(fim);
36         fim = novoNo;
37     }
38
39     tamanho++;
40 }
41
42 /**
43  * Remove e retorna o elemento no início da fila.
44  */
```

# Estruturas

## Lista

- Implementa uma lista simplesmente ou duplamente encadeada com métodos de adicionar , remover , consulta e visualizar.

```
6- public class ListaDuplamenteEncadeada<T> {
7-     private No<T> primeiro;
8-     private No<T> ultimo;
9-     private int tamanho;
10-
11-     public ListaDuplamenteEncadeada() {
12-         this.primeiro = null;
13-         this.ultimo = null;
14-         this.tamanho = 0;
15-     }
16-
17-     public void adicionar(T valor) {
18-         No<T> novoNo = new No<>(valor);
19-
20-         if (estaVazia()) {
21-             primeiro = novoNo;
22-             ultimo = novoNo;
23-         } else {
24-             ultimo.setProx(novoNo);
25-             novoNo.setPrev(ultimo);
26-             ultimo = novoNo;
27-         }
28-
29-         tamanho++;
30-     }
31- }
```

```
257- public void imprimir() {
258-     No<T> atual = head;
259-     while (atual != null) {
260-         System.out.print(atual.getValor() + " ");
261-         atual = atual.getProx();
262-     }
263-     System.out.println("-> NULL");
264- }
265-
266- /**
267-  * (Não utilizado)
268-  * <p>
269-  *
270-  * Imprime todos os elementos da lista do fim ao início.
271-  */
272- public void imprimirReverso() {
273-     No<T> atual = tail;
274-     while (atual != null) {
275-         System.out.print(atual.getValor() + " ");
276-         atual = atual.getPrev();
277-     }
278-     System.out.println("-> NULL");
279- }
280- }
```

# Classes

As classes do projeto foram divididas em Tempo, Zona, Caminhões, Configuração, Estações e Eventos

# Classe Caminhões

Cada uma representa um tipo específico de caminhão com funções diferentes dentro da lógica da simulação

# CaminhõesPequeno.java

Responsável por coletar lixo diretamente nas zonas da cidade.

## Características principais:

- Cada um tem:
  - ID único
  - Capacidade de carga
  - Um número limitado de viagens diárias
  - Uma rota de zonas para visitar e coletar lixo
- Quando está cheio ou termina a rota, vai para uma estação de transferência para descarregar no CaminhaoGrande.

## Métodos importantes:

- coletar(int quantidade) → coleta lixo da zona, se couber.
- descarregar() → esvazia o caminhão (geralmente na estação).
- podeRealizarNovaViagem() e registrarViagem() → controlam o número de viagens diárias.
- atualizarProximaZonaAlvo() e avancarParaProximaZona() → controlam o percurso pelas zonas.



```

private String id;
private int capacidadeMaxima;
private int cargaAtual;
private int numeroDeViagensDiarias;

private Lista<Zona> rota;
private int indiceRota = 0;

private Zona zonaAlvo;
private EventoGerarCaminhaoGrande eventoAgendado;

/**
 * Cria um caminhão pequeno com um número, limite de carga, viagens diárias e zonas para visitar.
 *
 * @param id Número único do caminhão
 * @param capacidadeMaxima Quantidade máxima de Lixo que pode carregar (em toneladas)
 * @param numeroDeViagensDiarias Quantas viagens o caminhão pode fazer por dia
 * @param rota Lista de zonas que o caminhão vai visitar, na ordem
 */
public CaminhaoPequeno(String id, int capacidadeMaxima, int numeroDeViagensDiarias, Lista<Zona> rota) {
    this.id = id;
    this.capacidadeMaxima = capacidadeMaxima;
    this.numeroDeViagensDiarias = numeroDeViagensDiarias;
    this.cargaAtual = 0;
    this.rota = rota;
    this.indiceRota = 0;
    this.zonaAlvo = rota.getValor(0); // Começa na primeira zona da lista
}

```

```

/**
 * Muda para a próxima zona na lista, se houver.
 * <p>
 * Também atualiza a zona atual do caminhão.
 */
public void avancarParaProximaZona() {
    if (indiceRota < rota.getTamanho() - 1) {
        indiceRota++; // Avança na lista
        this.zonaAlvo = rota.getValor(indiceRota); // Atualiza a zona atual
    }
}

/**
 * Verifica se o caminhão ainda tem zonas para visitar na lista.
 *
 * @return Verdadeiro se há mais zonas, falso se acabou
 */
public boolean temMaisZonasNaRota() {
    return indiceRota < rota.getTamanho() - 1;
}

```

# CaminhõesGrande.java

Responsável por transportar o lixo das estações de transferência até o aterro sanitário.

## Características principais:

- Capacidade fixa de 20 toneladas de lixo.
- Cada caminhão tem um ID único.
- Só recebe cargas de caminhões pequenos.
- Quando atinge a carga máxima ou é chamado, vai até o aterro e descarrega todo o lixo.
- Não coleta diretamente das zonas.

## Métodos importantes:

- receberCarga(int quantidade) → recebe lixo de um caminhão pequeno.
- estaCheio() → indica se atingiu a capacidade.
- descarregar() → leva o lixo acumulado para o aterro.

```

public class CaminhaoGrande {
    private static int proximoId = 1; // Gera números únicos para cada caminhão
    private int id;
    private final int limiteCarga = 20; // Quantidade máxima de lixo que pode carregar
    private int cargaAtual;
    private boolean carregando;
    private int tempoMaximoEspera;

    /**
     * Cria um novo caminhão grande com um número único e sem lixo carregado.
     */
    public CaminhaoGrande() {
        this.id = proximoId++;
        this.cargaAtual = 0;
        this.carregando = true;
    }
}

```

```

71- public void receberCarga(int quantidade) {
72-     // Adiciona lixo, mas respeita o limite máximo
73-     cargaAtual = Math.min(cargaAtual + quantidade, limiteCarga);
74- }
75-
76- /**
77-  * Esvazia o caminhão no aterro sanitário.
78-  * <p>
79-  * Zera o lixo carregado e marca o caminhão como não carregando mais.
80-  */
81- public void descarregar() {
82-     System.out.println("Caminhão grande #" + id + " foi para o aterro com " + cargaAtual + " unidades de lixo.");
83-     cargaAtual = 0; // Zera a carga
84-     carregando = false; // Finaliza o carregamento
85- }
86- }

```

# Classe de Zonas

- Associar zonas a estações de transferência.
- Armazenar e fornecer acesso às zonas cadastradas.
- Gerenciar quais caminhões pequenos ainda podem fazer viagens.

A classe `Zona` modela:

- O nome da zona
- A quantidade de lixo gerado por dia (aleatório entre mínimo e máximo)
- O lixo acumulado que precisa ser coletado

```

8- public class Zonas {
9-
10-    /**
11-     * Cria a zona Sul com parâmetros de geração de lixo.
12-     *
13-     * @return Zona Sul configurada
14-     */
15-    public static Zona zonaSul() {
16-        return new Zona("Sul", configuracao.LIXO_MIN_SUL, configuracao.LIXO_MAX_SUL);
17-    }
18-
19-    /**
20-     * Cria a zona Norte com parâmetros de geração de lixo.
21-     *
22-     * @return Zona Norte configurada
23-     */
24-    public static Zona zonaNorte() {
25-        return new Zona("Norte", configuracao.LIXO_MIN_NORTE, configuracao.LIXO_MAX_NORTE);
26-    }
27-
28-    /**
29-     * Cria a zona Centro com parâmetros de geração de lixo.
30-     *
31-     * @return Zona Centro configurada
32-     */
33-    public static Zona zonaCentro() {
34-        return new Zona("Centro", configuracao.LIXO_MIN_CENTRO, configuracao.LIXO_MAX_CENTRO);
35-    }
36-
37-    /**

```

```

8- public class Zona {
9-
10-    private String nome;
11-    private int lixoMinimo;
12-    private int lixoMaximo;
13-    private int lixoAcomulado;
14-
15-    /**
16-     * Cria uma zona com nome e limites de geração de lixo.
17-     *
18-     * @param nome Nome da zona
19-     * @param lixoMinimo Mínimo de lixo gerado por dia
20-     * @param lixoMaximo Máximo de lixo gerado por dia
21-     */
22-    public Zona(String nome, int lixoMinimo, int lixoMaximo) {
23-        this.nome = nome;
24-        this.lixoMinimo = lixoMinimo;
25-        this.lixoMaximo = lixoMaximo;
26-        this.lixoAcomulado = 0;
27-    }
28-
29-    /**
30-     * Gera uma quantidade aleatória de lixo diário.
31-     */
32-    public void gerarLixoDiario() {
33-        this.lixoAcomulado = new Random().nextInt(lixoMaximo - lixoMinimo + 1) + lixoMinimo;
34-        System.out.println("[Zona] " + nome + " gerou " + lixoAcomulado + " toneladas de lixo.");
35-    }
36-
37-    /**

```

```

10 public class GerenciadorZonas {
11
12     private static EstacaoDeTransferencia estacaoA;
13     private static EstacaoDeTransferencia estacaoB;
14     private static Lista<Zona> zonas;
15     private static Lista<CaminhaoPequeno> caminhos;
16
17     /**
18      * Configura as estações A (Leste, Norte, Centro) e B (Sul, Sudeste).
19      *
20      * @param a Estação A
21      * @param b Estação B
22      */
23     public static void configurar(EstacaoDeTransferencia a, EstacaoDeTransferencia b) {
24         estacaoA = a;
25         estacaoB = b;
26     }
27
28     /**
29      * Retorna a estação correspondente a uma zona.
30      *
31      * @param zona Zona a ser verificada
32      * @return Estação responsável
33      * @throws IllegalArgumentException se a zona for desconhecida
34      */
35     public static EstacaoDeTransferencia getEstacaoPara(Zona zona) {
36         String nome = zona.getNome().toLowerCase();
37
38         if (nome.equals("leste") || nome.equals("norte") || nome.equals("centro")) {
39             return estacaoA;
40         }
41
42         if (nome.equals("sul") || nome.equals("sudeste")) {
43             return estacaoB;
44         }
45
46         throw new IllegalArgumentException("Zona desconhecida: " + zona.getNome());
47     }
48 }

```

# Classe GerenciadorTempo.java

Essa classe centraliza todas as operações relacionadas à **gestão de tempo**, incluindo formatação de horários, cálculo de durações e simulação de efeitos de trânsito (como congestionamentos).

- Multiplicadores de tempo definidos na configuração (ex: trânsito mais lento).
- Ajusta cada minuto da viagem com base no tráfego.

```
9 public class GerenciadorTempo {
10     // Horários de pico (em minutos desde 00:00)
11     private static final int PICO_MANHA_INICIO = 360; // 06:00
12     private static final int PICO_MANHA_FIM = 540; // 09:00
13     private static final int PICO_TARDE_INICIO = 1020; // 17:00
14     private static final int PICO_TARDE_FIM = 1260; // 20:00
15     private static final int HORA_INICIAL_SIMULACAO = 420; // 07:00
16
17     /**
18      * Formata minutos desde 07:00 como horário HH:mm.
19      *
20      * @param minutosDecorridos Minutos desde o início da simulação
21      * @return String no formato "HH:mm"
22      * @throws IllegalArgumentException se minutos forem negativos
23      */
24     public static String formatarHorarioSimulado(int minutosDecorridos) {
25         if (minutosDecorridos < 0) {
26             throw new IllegalArgumentException("Minutos decorridos não podem ser negativos");
27         }
28         int minutosTotais = HORA_INICIAL_SIMULACAO + minutosDecorridos;
29         int horas = minutosTotais / 60;
30         int minutos = minutosTotais % 60;
31         return String.format("%02d:%02d", horas, minutos);
32     }
33 }
34
```

```
87 public static TempoDetalhado calcularTempoDetalhado(int tempoSimulacao, int cargaToneladas, boolean isDescarregamento) {
88     if (tempoSimulacao < 0 || cargaToneladas < 0) {
89         throw new IllegalArgumentException("Parâmetros devem ser não negativos");
90     }
91
92     // Verifica horário de pico
93     int minutosTotais = HORA_INICIAL_SIMULACAO + tempoSimulacao;
94     boolean emPico = isPeriodoCongestionado(minutosTotais);
95
96     // Seleciona tempos mínimo e máximo
97     int tempoMinimo = emPico ? configuracao.TEMPO_MIN_PICO : configuracao.TEMPO_MIN_FORA_PICO;
98     int tempoMaximo = emPico ? configuracao.TEMPO_MAX_PICO : configuracao.TEMPO_MAX_FORA_PICO;
99
100     // Gera tempo base
101     int tempoBaseViagem = ThreadLocalRandom.current().nextInt(tempoMinimo, tempoMaximo + 1);
102
103     // Calcula tempos
104     int tempoViagem = estimarTempoViagem(tempoSimulacao, tempoBaseViagem);
105     int tempoOperacao = isDescarregamento ? 0 : cargaToneladas * configuracao.TEMPO_COLETA_POR_TONELADA;
106     int tempoAdicionalCarga = isDescarregamento ? (int) (tempoViagem * 0.3) : 0;
107     int tempoTotalOperacao = tempoOperacao + tempoViagem + tempoAdicionalCarga;
108
109     return new TempoDetalhado(tempoOperacao, tempoViagem, tempoTotalOperacao);
110 }
111
112 /**
113  * Verifica se o tempo está em período de pico.
114  */
115
```

# Classe TempoDetalhado.java

```
6 public class TempoDetalhado {
7
8     /** Tempo de coleta (em minutos) */
9     public final int tempoColeta;
10
11     /** Tempo de deslocamento (em minutos) */
12     public final int tempoDeslocamento;
13
14     /** Tempo extra por carga máxima (em minutos) */
15     public final int tempoExtraCarregado;
16
17     /** Tempo total da operação (em minutos) */
18     public final int tempoTotal;
19
20     /**
21      * Cria uma estrutura com tempos detalhados.
22      *
23      * @param tempoColeta Tempo de coleta (em minutos)
24      * @param tempoDeslocamento Tempo de deslocamento (em minutos)
25      * @param tempoExtraCarregado Tempo extra por carga (em minutos)
26      */
27     public TempoDetalhado(int tempoColeta, int tempoDeslocamento, int tempoExtraCarregado) {
28         this.tempoColeta = tempoColeta;
29         this.tempoDeslocamento = tempoDeslocamento;
30         this.tempoExtraCarregado = tempoExtraCarregado;
31         this.tempoTotal = tempoColeta + tempoDeslocamento + tempoExtraCarregado;
32     }
33 }
```



# Classe Configuracao

Centraliza todos os parâmetros fixos do simulador de coleta de lixo. Em outras palavras, é uma **classe de configuração global** que contém **constantes** utilizadas em várias partes do sistema. Isso facilita a **manutenção**, a **leitura** do código e a **alteração de valores** sem precisar sair procurando em vários arquivos.

```
// ===== TEMPOS DE TRABALHO =====  
  
/**  
 * Minutos para descarregar 1 tonelada de lixo.  
 */  
public static final int TEMPO_DESCARGA_POR_TONELADA = 3;  
  
/**  
 * Minutos para coletar 1 tonelada de lixo.  
 */  
public static final int TEMPO_COLETA_POR_TONELADA = 10;  
  
// = TEMPO DE VIAGENS DOS CAMINHÕES  
  
/**  
 * Tempo mínimo de viagem em horários de muito trânsito (em minutos).  
 */  
public static final int TEMPO_MIN_PICO = 20;  
  
/**  
 * Tempo máximo de viagem em horários de muito trânsito (em minutos).  
 */  
public static final int TEMPO_MAX_PICO = 60;  
  
/**  
 * Tempo mínimo de viagem em horários tranquilos (em minutos).  
 */  
public static final int TEMPO_MIN_FORA_PICO = 20;  
  
/**  
 * Tempo máximo de viagem em horários tranquilos (em minutos).  
 */  
public static final int TEMPO_MAX_FORA_PICO = 35;
```

```
49 // =GERAÇÃO DE LIXO POR ZONA
50
51 /**
52  * Menor quantidade de lixo gerada por dia na zona Sul (em toneladas).
53  */
54 public static final int LIXO_MIN_SUL = 20;
55
56 /**
57  * Maior quantidade de lixo gerada por dia na zona Sul (em toneladas).
58  */
59 public static final int LIXO_MAX_SUL = 40;
60
61 /**
62  * Menor quantidade de lixo gerada por dia na zona Norte (em toneladas).
63  */
64 public static final int LIXO_MIN_NORTE = 15;
65
66 /**
67  * Maior quantidade de lixo gerada por dia na zona Norte (em toneladas).
68  */
69 public static final int LIXO_MAX_NORTE = 30;
70
71 /**
72  * Menor quantidade de lixo gerada por dia na zona Centro (em toneladas).
73  */
74 public static final int LIXO_MIN_CENTRO = 10;
75
76 /**
```

# Multiplicadores de tempo

Define o quanto o tempo de viagem **aumenta ou se mantém**:

- Em horário de pico → `MULTIPLICADOR_TEMPO_PICO = 1.5`
- Fora do pico → `MULTIPLICADOR_TEMPO_FORA_PICO = 1.0`

Útil para ajustar o tempo final de viagem de acordo com o trânsito.

```
103  /**  
104   * Aumento do tempo de viagem em horários de muito trânsito.  
105   */  
106   public static final double MULTIPLICADOR_TEMPO_PICO = 1.5;  
107  
108  /**  
109   * Tempo normal de viagem em horários tranquilos.  
110   */  
111   public static final double MULTIPLICADOR_TEMPO_FORA_PICO = 1.0;  
112
```

# Classe EstacaoDeTransferencia

Simular a **logística realista** de uma **estação de transbordo** de lixo, onde:

- Há filas.
- Há tempos de espera.
- Há limitações de capacidade.
- Há agendamentos de recursos (caminhões grandes).

```
16 public class EstacaoDeTransferencia {
17     private String nomeEstacao;
18     private CaminhaoGrande caminhaoGrandeAtual;
19     private Fila<CaminhaoPequeno> filaCaminhoes = new Fila<>();
20
21     /**
22      * Cria uma estação de transferência.
23      *
24      * @param nomeEstacao Nome da estação
25      */
26     public EstacaoDeTransferencia(String nomeEstacao) {
27         this.nomeEstacao = nomeEstacao;
28         this.caminhaoGrandeAtual = new CaminhaoGrande();
29     }
30 }
```

```
36 public void gerarNovoCaminhaoGrande(int tempoAtual) {  
37     this.caminhaoGrandeAtual = new CaminhaoGrande();  
38     System.out.println("[ESTAÇÃO " + nomeEstacao + "] Novo caminhão grande criado.");  
39     descarregarFilaEspera(tempoAtual);  
40 }  
41  
42 /**  
43  * Retorna o nome da estação.  
44  *  
45  * @return Nome da estação  
46  */  
47 public String getNomeEstacao() {  
48     return nomeEstacao;  
49 }  
50
```

```

75 public void receberCaminhaoPequeno(CaminhaoPequeno caminhao, int tempoAtual) {
76     // Início da tabela
77     System.out.println("+-----+");
78     System.out.println("|          DESCARREGAMENTO NA ESTAÇÃO          |");
79     System.out.println("+-----+");
80     System.out.printf("| %-18s | %-28s |\n", "Horário de Chegada", GerenciadorTempo.formatarHorarioSimulado(tempoAtual));
81     System.out.printf("| %-18s | %-28s |\n", "Estação", nomeEstacao);
82     System.out.printf("| %-18s | %-28s |\n", "Caminhão", caminhao.getId());
83     System.out.printf("| %-18s | %-28s |\n", "Status", "Chegada confirmada");
84
85     // Sem caminhão grande ou caminhão grande cheio
86     if (caminhaoGrandeAtual == null || caminhaoGrandeAtual.estaCheio()) {
87         filaCaminhoes.enqueue(caminhao);
88         System.out.printf("| %-18s | %-28d |\n", "Tamanho da Fila", filaCaminhoes.size());
89
90         // Agenda criação de caminhão grande, se não agendado
91         if (caminhao.getEventoAgendado() == null) {
92             int tempoLimite = tempoAtual + 100;
93             EventoGerarCaminhaoGrande evento = new EventoGerarCaminhaoGrande(tempoLimite, this);
94             AgendaEventos.adicionarEvento(evento);
95             caminhao.setEventoAgendado(evento);
96             System.out.printf("| %-18s | %-28s |\n", "Evento Agendado", "Caminhão grande às " + GerenciadorTempo.formatarHorarioSimulado(tempoLimite));
97         }
98     } else {
99         // Descarrega diretamente
100         if (caminhao.getEventoAgendado() != null) {
101             AgendaEventos.removerEvento(caminhao.getEventoAgendado());
102             caminhao.setEventoAgendado(null);
103             System.out.printf("| %-18s | %-28s |\n", "Evento Cancelado", "Geração de caminhão grande");

```

# Classe de Eventos

As seguintes classes trabalham juntas com o **agendador de eventos** (`AgendaEventos`) para controlar o que acontece, quando acontece e com quem acontece na simulação.

- `AgendaEventos.java`
- `Evento.java`
- `EventoColeta.java`
- `EventoDistribuidorDeRotas.java`
- `EventoEstacaoTransferencia.java`
- `EventoGeracaoLixoZona.java`
- `EventoGerarCaminhaoGrande.java`
- `EventoTransferenciaParaEstacao.java`



- O **lixo é gerado** nas zonas → `EventoGeracaoLixoZona`.
- Um caminhão pequeno é enviado para **coletar lixo** → `EventoColeta`.
- Caminhão cheio é **enviado para a estação** → `EventoTransferenciaParaEstacao`.
- Chegando na estação → `EventoEstacaoTransferencia`.
- Se não houver caminhão grande disponível → `EventoGerarCaminhaoGrande` é agendado.
- A classe `AgendaEventos` garante que os eventos aconteçam na ordem correta do tempo.

# Resultados Obtidos

- Durante a simulação, inicialmente foi realizada a geração de resíduos em cada zona (Sul, Sudeste, Centro, Leste e Norte), permitindo observar a distribuição do lixo gerado pelo sistema urbano. Em seguida, o sistema apresentou relatórios detalhados para cada caminhão envolvido no processo de coleta, identificados por ID.

Esses relatórios incluíram estatísticas fundamentais, como:

- Tempo de coleta,
- Tempo de trajeto,
- Quantidade de carga coletada,
- Zona atendida,
- Horário de início e término da operação,
- E o número de viagens restantes para o dia.

Após cada operação de coleta e transporte até a estação, também foram apresentados dados sobre:

- Carga descarregada,
- Tempo de descarga,
- Horário da conclusão da atividade,
- E a próxima ação do caminhão, como retorno à coleta ou finalização do expediente.

Além disso, o sistema identificou automaticamente o momento em que os caminhões grandes (de maior capacidade) atingiam sua carga máxima e se dirigiam ao aterro sanitário, informando claramente o volume transportado.

Esses dados demonstram que o sistema foi capaz de simular com precisão uma operação realista de coleta, transporte e descarregamento de resíduos sólidos urbanos. Conclui-se que o modelo implementado fornece uma visão clara da logística envolvida, auxiliando na tomada de decisões e no dimensionamento de recursos para futuras implementações reais ou acadêmicas



# Desafios

- Seguir a ordem cronológica dos eventos: para garantir que todos os eventos sejam adicionados, removidos e executados na ordem correta, sem conflitos ou sobreposições, foi muito complexo.
- Gerenciar a dinâmica do tempo: para garantir que os eventos sejam agendados com tempos realistas, e tempos de trajeto), porem com alguns erros nos cálculos de tempo ou na lógica de eventos.
- Gestão de fila: para caminhões pequenos chegando às estações de transferência, garantindo que descarreguem sem bloqueios, enquanto lidam com a capacidade limitada dos caminhões grandes (20 toneladas) e tempos dinâmicos de espera e descarga.

# FIM