Documentação do Projeto SemaFlux

Introdução

O SemaFlux é um sistema de simulação de tráfego urbano desenvolvido em Java, utilizando a biblioteca JavaFX para criar uma interface gráfica interativa. Ele é projetado para modelar e analisar fluxos de tráfego, estratégias de controle de semáforos e visualização em tempo real, com suporte a mapas personalizados importados via JSON. O projeto é voltado para fins educacionais e analíticos, permitindo a visualização de veículos, análise de congestionamentos e testes de três estratégias de semáforos: Tempo Fixo, Adaptativo e Economia de Energia. Ele requer Java 11 ou superior, Maven 3.6 ou superior para gerenciamento de bibliotecas e aproximadamente 100MB de espaço em disco. O controle de versionamento é feito com Git.

Componentes Principais

O SemaFlux é composto por módulos que separam a lógica de simulação, visualização e gerenciamento de dados:

- **Interface do Usuário**: Inclui um painel lateral com estatísticas, um gráfico de congestionamento (LineChart), legenda para semáforos e controles de zoom e velocidade no rodapé, estilizados com CSS (modern-style.css).
- **Simulação de Tráfego**: Gerencia veículos, semáforos e rotas, com atualizações em tempo real e suporte a mapas JSON (ex.: JoqueiTeresinaPiauiBrazil.json).
- **Gerenciamento de Dados**: Utiliza uma estrutura de grafo com nós (interseções), arestas (ruas) e semáforos, além de estatísticas como tempo de viagem e consumo de combustível.
- **Visualização**: Renderiza o mapa, veículos e semáforos dinamicamente, com animações suaves e tooltips.

Estruturas de Dados Customizadas

O SemaFlux implementa estruturas de dados personalizadas para atender às necessidades específicas da simulação de tráfego, sem depender exclusivamente das coleções padrão do Java (como `ArrayList` ou `HashMap`). Essas estruturas foram projetadas para gerenciar listas dinâmicas, filas de veículos e conexões no grafo de

tráfego, garantindo eficiência e adequação ao domínio da simulação. Abaixo, detalhamos as principais estruturas customizadas:

Estrutura	Localização	Propósito	Características Principais	Uso
ListaLigada	`org.semaflux.sim.core.ListaLigada.java`	Lista ligada genérica para coleções dinâmicas de nós, arestas, semáforos e rotas.	Implementa `Iterable<T>`; métodos: `add`, `addFirst`, `remove`, `get`, `size` (O(1)), `isEmpty` (O(1)); usa `No<T>` aninhado.	Em `Grafo` para `nodesList`, `edgesList`, `trafficLightsList`; em `Veiculo` para `route`.
No	Aninhada em `ListaLigada<T>`	Nó da lista ligada, armazenando dados e referência ao próximo nó.	Campos: `data` (tipo T), `next` (próximo nó); construtor inicializa `data` e `next` como `null`.	Componente interno de `ListaLigada` para construir listas dinâmicas.
Fila	`org.semaflux.sim.core.Fila.java`	Fila FIFO para gerenciar veículos em semáforos.	Campos: `size`, `front`, `back`; métodos: `enqueue`, `dequeue` (O(1)), `peek`, `isEmpty`; usa campo `next` em `Veiculo`.	Em `SinalTransito` para `directionQueues` (norte, leste, sul, oeste).
Aresta	`org.semaflux.sim.core.Aresta.java`	Representa ruas no grafo, com comportamento de lista ligada.	Campos: `id`, `source`, `target`, `length`, `travelTime`, `next`; métodos: `getAverageSpeed`, `isBidirectional`.	Em `No` para `edges` (usando `ListaLigada<Aresta>`), conectando nós no `Grafo`.

1. ListaLigada (Lista Ligada Customizada)

- **Localização**: `org.semaflux.sim.core.ListaLigada.java`.
- **Propósito**: Uma lista ligada simplesmente encadeada genérica, usada para armazenar coleções dinâmicas de objetos, como nós, arestas, semáforos e rotas de veículos no grafo de tráfego.
- **Características Principais**:
    - Implementa a interface `Iterable<T>` para suportar iteração.

    - Métodos principais: add (adiciona ao final), addFirst (adiciona ao início), remove, removeFirst, get, getFirst, getLast, size, isEmpty, contains.
    - Usa uma classe aninhada No<T> para representar cada elemento, com campos data (dado armazenado) e next (referência ao próximo nó).
    - Complexidade de tempo: O(n) para a maioria das operações, exceto size e isEmpty (O(1)).
    - Inclui um iterador para percorrer a lista, facilitando o acesso sequencial.
- **Uso**:
    - Em Grafo, armazena listas de nós (nodesList), arestas (edgesList) e semáforos (trafficLightsList).
    - Em Veiculo, armazena a rota do veículo (route) como uma lista de IDs de nós.
    - Exemplo: A rota de um veículo é representada como uma ListaLigada<String>, onde cada elemento é o ID de um nó no caminho.

2. No (Nó para Lista Ligada)

- **Localização**: Classe aninhada dentro de ListaLigada<T> em org.semaflux.sim.core.ListaLigada.java.
- **Propósito**: Representa um nó individual na ListaLigada, armazenando um elemento de tipo genérico T e uma referência ao próximo nó.
- **Características Principais**:
    - Campos: data (dado do tipo T) e next (referência ao próximo No<T>).
    - Construtor: Inicializa data com o valor fornecido e define next como null.
- **Uso**: Componente essencial da ListaLigada, usado para construir a estrutura encadeada que suporta coleções dinâmicas na simulação.

3. Fila (Fila para Veículos)

- **Localização**: org.semaflux.sim.core.Fila.java.
- **Propósito**: Uma fila FIFO (First In, First Out) projetada para gerenciar veículos esperando em semáforos ou interseções, utilizando uma estrutura encadeada baseada em objetos Veiculo.
- **Características Principais**:
    - Campos: size (número de veículos), front (primeiro veículo), back (último veículo). Cada Veiculo possui um campo next para vincular ao próximo veículo na fila.

    - Métodos: `isEmpty` (verifica se a fila está vazia), `size` (retorna o número de veículos), peek (visualiza o primeiro veículo), enqueue (adiciona ao final), dequeue (remove do início).
    - Complexidade de tempo: O(1) para operações enqueue e dequeue, garantindo eficiência no gerenciamento de filas.
- **Uso**:
    - Em `SinalTransito`, gerencia filas de veículos (`directionQueues`) para cada direção (norte, leste, sul, oeste) em um semáforo.
    - Exemplo: Quando um veículo chega a um semáforo vermelho, ele é adicionado à `Fila` correspondente à sua direção via enqueue.

4. Aresta (Aresta)

- **Localização**: `org.semaflux.sim.core.Aresta.java`.
- **Propósito**: Representa um segmento de estrada no grafo de tráfego, conectando dois nós (interseções). Inclui um campo `next` para suportar um comportamento semelhante a uma lista ligada quando armazenado.
- **Características Principais**:
    - Campos: `id` (identificador), `source` (nó de origem), `target` (nó de destino), `length` (comprimento), `travelTime` (tempo de viagem), oneway (sentido único), `maxspeed` (velocidade máxima), `capacity` (capacidade), `next` (próxima aresta na lista).
    - Métodos: Getters e setters para todos os campos, além de métodos utilitários como `getAverageSpeed` e `isBidirectional`.
    - O campo `next` permite encadear objetos `Aresta` em uma estrutura semelhante a uma lista ligada.
- **Uso**:
    - Armazenado no campo `edges` de `No`, usando `ListaLigada<Aresta>`, para representar conexões entre nós no `Grafo`.
    - Exemplo: Em um nó representando uma interseção, a lista de arestas (`edges`) contém todas as ruas que partem desse nó.

Principais Classes

As classes principais do SemaFlux organizam a lógica da simulação, controle, núcleo e visualização. A tabela abaixo resume suas funções:

Classe	Pacote	Propósito

`InicioSis`	`org.semaflux.sim`	Ponto de entrada, gerencia inicialização e configuração da simulação.
`Dijkstra`	`org.semaflux.sim.control`	Calcula rotas mais curtas usando o algoritmo de Dijkstra.
`leitorJson`	`org.semaflux.sim.control`	Processa arquivos JSON para criar o grafo.
`Aresta`, `No`, `Grafo`, `Fila`, `SinalTransito`, `Veiculo`	`org.semaflux.sim.core`	Representam a rede de tráfego e seus elementos.
`Config`, `Estatisticas`, `Simulador`	`org.semaflux.sim.simulação`	Gerenciam configuração, estatísticas e lógica da simulação.
`Visualizer`	`org.semaflux.sim.visualization`	Renderiza a simulação em tempo real com JavaFX.

Métodos Principais

- **Controle de Simulação**: `iniciarSimulacao` (inicia a simulação), `run` (executa o loop principal em `Simulador`).
- **Cálculo de Rotas**: `calcularRota` (em `Dijkstra`, determina rotas mais curtas).
- **Gerenciamento de Semáforos**: `update` (em `SinalTransito`, ajusta estados com base no tempo).
- **Visualização**: `atualizarElementosDinamicos` (em `Visualizer`, atualiza posições de veículos e semáforos).
- **Estatísticas**: `calculateCurrentCongestion` (em `Estatisticas`, rastreia métricas como congestionamento).

Integração com JSON

O SemaFlux utiliza arquivos JSON para definir a rede de tráfego, incluindo:

- **nodes**: Interseções com ID, latitude e longitude.
- **edges**: Ruas com origem, destino, comprimento e velocidade máxima.
- **traffic_lights**: Semáforos com localização e direções controladas.

A classe `leitorJson` processa esses dados, criando objetos `No`, `Aresta` e `SinalTransito` para o `Grafo`.

Conclusão

A documentação aprimorada do SemaFlux destaca as estruturas de dados customizadas (`ListaLigada<T>`, `No<T>`, `Fila`, `Aresta`), que são essenciais para a eficiência e flexibilidade da simulação. Essas estruturas, implementadas sem depender de coleções padrão do Java, suportam a gestão dinâmica de elementos do grafo, filas de veículos e conexões de ruas, garantindo uma simulação robusta e adaptável.