Smart Traffic Flow Simulation

Relatório Técnico

---

1. Introdução

Este relatório descreve o desenvolvimento do projeto **Smart Traffic Flow Simulation**, um simulador de tráfego urbano implementado em Java com JavaFX. O objetivo principal do projeto foi modelar um sistema de tráfego com veículos autónomos que circulam por estradas e interseções controladas por semáforos inteligentes, aplicando princípios de **Programação Orientada a Objetos (OOP)** e **padrões de design** estudados na unidade curricular.

O sistema permite observar o comportamento dos veículos em tempo real, alternar políticas de controlo de semáforos, recolher estatísticas de desempenho e avaliar o impacto de estratégias adaptativas no fluxo de tráfego. Para além dos requisitos base, o projeto foi concebido com foco em modularidade, extensibilidade e clareza arquitetural.

---

2. Arquitetura do Sistema

A arquitetura do sistema segue uma separação clara de responsabilidades, organizada em pacotes lógicos:

- **model** – contém as classes do domínio do problema (veículos, estradas, interseções, semáforos);
- **controller** – gere o ciclo de simulação, as estratégias de controlo e a coordenação entre entidades;
- **view** – responsável pela visualização gráfica e interação com o utilizador (JavaFX);
- **util** – recolha de métricas e exportação de resultados.

Esta organização promove **baixo acoplamento** e **alta coesão**, facilitando manutenção, testes e futuras extensões.

2.1 Mundo e Ciclo de Simulação

A classe `Simulation` atua como o núcleo do sistema. É responsável por: - inicializar as entidades do mundo (estradas, interseções e veículos); - executar o *loop de simulação* baseado em *ticks*; - atualizar semáforos e veículos de forma ordenada; - aplicar regras de paragem (semáforos e colisões); - recolher dados para estatísticas.

A cada iteração, o sistema atualiza primeiro os semáforos e depois os veículos, garantindo consistência no estado global.

2.2 Modelo de Domínio

- **Vehicle**: representa um veículo autónomo, contendo posição, velocidade, direção e estado de movimento. O veículo reage ao ambiente (semáforos e outros veículos), mas não possui lógica global de controlo.

- **EmergencyVehicle**: especialização de `Vehicle` que ignora semáforos, demonstrando herança e polimorfismo.
- **Road**: agrega os veículos e centraliza a sua atualização.
- **Intersection**: representa uma interseção e coordena dois semáforos perpendiculares (Norte–Sul e Este–Oeste).
- **TrafficLight**: encapsula o estado atual do semáforo e delega o seu comportamento a objetos de estado e estratégia.

---

3. Padrões de Design Utilizados

O projeto utiliza explicitamente dois padrões de design fundamentais: **State Pattern** e **Strategy Pattern**.

3.1 State Pattern

O **State Pattern** é utilizado para modelar os diferentes estados de um semáforo: - `RedState` - `YellowState` - `GreenState`

Cada estado implementa uma interface comum (`LightState`) e define o comportamento específico do semáforo nesse estado, incluindo se os veículos podem ou não atravessar a interseção e quando ocorre a transição para o próximo estado.

Vantagens desta abordagem: - elimina cadeias complexas de `if/else`; - torna as transições explícitas e seguras; - facilita a extensão com novos estados, se necessário.

3.2 Strategy Pattern

O **Strategy Pattern** é utilizado para encapsular diferentes políticas de controlo dos semáforos: - `FixedCycle` – utiliza tempos fixos para cada estado; - `AdaptiveCycle` – ajusta dinamicamente os tempos com base no número de veículos em espera.

A classe `TrafficLight` mantém uma referência para uma estratégia, podendo alternar o comportamento sem necessidade de alterações internas.

Benefícios principais: - separação clara entre o *que* o semáforo faz e *como* decide os tempos; - facilidade em adicionar novas estratégias de controlo; - suporte direto a controlo adaptativo, conforme exigido no enunciado.

---

4. Visualização e Interface Gráfica

A visualização foi implementada com **JavaFX**, utilizando um `Canvas` para desenhar estradas, veículos e semáforos. A interface inclui: - botões de controlo (Start, Stop, Reset); - um *slider* para ajustar a velocidade da simulação; - animação fluida baseada em `AnimationTimer`.

A camada de visualização está completamente separada da lógica de negócio. As classes da *view* apenas consultam o estado do modelo e o representam graficamente, respeitando o princípio de separação de responsabilidades.

---

5. Estratégia de Teste

A validação do sistema foi realizada através de uma combinação de:

- **Testes funcionais manuais**, observando o comportamento da simulação em diferentes cenários;
- **Testes de cenários limite**, como:
- tráfego intenso em apenas uma direção;
- presença simultânea de veículos normais e de emergência;
- alteração dinâmica da velocidade da simulação;
- **Validação visual**, confirmando que:
- os veículos respeitam os semáforos;
- não ocorrem colisões;
- as transições de estado dos semáforos são corretas.

Durante o desenvolvimento, foi identificado e corrigido um problema relacionado com a deteção de aproximação ao cruzamento, que afetava veículos que se deslocavam em sentidos negativos do eixo. A solução consistiu em adaptar as comparações de posição à direção do veículo, sem alterar a arquitetura existente.

---

6. Estatísticas e Resultados

O sistema recolhe métricas relevantes para análise de desempenho do tráfego, incluindo: - número de veículos em espera; - veículos servidos; - tempo médio de espera (dependendo da configuração); - impacto da estratégia adaptativa no fluxo.

Os resultados podem ser exportados para **CSV**, permitindo análise externa em ferramentas como Excel ou Python. Observou-se que a estratégia adaptativa reduz o tempo médio de espera em cenários de tráfego assimétrico, quando comparada com ciclos fixos.

---

7. Extensões Implementadas

Como extensão ao projeto base, foi implementado suporte a **veículos de emergência**, que: - herdam da classe `Vehicle`; - ignoram semáforos e congestionamentos; - demonstram corretamente o uso de herança e polimorfismo.

Esta extensão foi integrada sem necessidade de refatorar o código existente, validando a flexibilidade da arquitetura adotada.

---

8. Conclusão

O projeto **Smart Traffic Flow Simulation** cumpre todos os requisitos propostos no enunciado, demonstrando uma aplicação consistente de princípios de Programação Orientada a Objetos e padrões de design. A arquitetura modular permitiu a implementação de funcionalidades avançadas, como controlo adaptativo e veículos de emergência, sem comprometer a clareza do sistema.

O simulador constitui uma base sólida para futuras extensões, como múltiplas interseções, otimização de tráfego ou integração de inteligência artificial mais avançada, refletindo boas práticas de engenharia de software.

Autores: [Nomes dos alunos]

Instituição: [Nome da instituição]

Ano letivo: [Ano]