SMART TRAFFIC SIMULATION

Programação Orientada a Objetos

Link do Repositório:
https://github.com/FranciscoMoroni/SmartTrafficSimulation

Francisco Moroni 30014347, Mariana Pinto 30014891 e Nuno Monteiro 30014322

1. Introdução

Este projeto é um simulador de trânsito feito em Java. O objetivo foi criar um ambiente onde carros e peões interagem de forma realista, e testar como semáforos inteligentes podem ajudar veículos de emergência a passar mais depressa.

2. Arquitetura do Sistema

A aplicação adota o padrão de arquitetura Model-View-Controller (MVC)

2.1. Camada Model

A camada Model contém a representação do mundo. As classes principais incluem:

- World: Atua como o contentor global de todas as entidades (Roads, Vehicles, TrafficLights). Gere o ciclo de vida dos objetos e como são removidos quando saem do campo de visão.
- Vehicle: Classe abstrata que define as propriedades físicas (posição, velocidade, aceleração, distância de segurança).
- Road: Define como é a rodoviária e os limites de circulação.

2.2. Camada View (Interface)

Implementada em JavaFX, usado para a renderização.

- Utilização de um Canvas.
- A classe CanvasView lê os dados do Modelo e traduz coordenadas lógicas em pixels no ecrã.

2.3. Camada Controller

Faz a ponte entre a interação do utilizador e a simulação.

- SimulationController: Gere o AnimationTimer. Em cada "tick" de tempo, solicita ao Modelo para calcular as novas posições e ordena que redesenhe o cenário.

3. Implementação do Padrão State

A gestão dos semáforos é um dos pontos mais importantes do projeto. Em vez de utilizarmos uma variável inteira ou uma enumeração com vários if/else dentro de um ciclo, usamos o State.

3.1. A Interface LightState

Definimos uma interface que obriga todos os estados (Verde, Amarelo, Vermelho) a implementar o método update(TrafficLight light). Isto permite que o objeto TrafficLight não precise de saber que cor está a exibir, diz apenas qual é a responsabilidade do estado atual.

3.2. As Classes de Estado:

- RedState: Bloqueia a passagem dos veículos. Quando o temporizador interno termina, vai automaticamente para o GreenState.
- GreenState: Permite o movimento. É aqui que a lógica de prioridade pode intervir, como por exemplo detectar a ambulância.
- YellowState: Serve como estado de transição e aviso.

Benefício: Esta abordagem permite que se no futuro quisermos adicionar um estado de luz intermitente, basta criar uma nova classe que implemente a interface, sem ter de alterar nenhuma linha de código nos semáforos existentes.

4. Gestão de Fluxo com o Padrão Strategy

O Strategy gere a "inteligência" do cruzamento. Criámos uma estrutura que permite mudar o comportamento do tráfego em tempo real.

4.1. Estratégia de Ciclo Fixo (FixedCycleStrategy)

É a estratégia base. Os semáforos mudam com base em tempos rígidos.

4.2. Estratégia de Emergência (EmergencyPriorityStrategy)

O sistema corre um algoritmo:

1. Verifica se existe algum objeto do tipo EmergencyVehicle num raio de X metros do cruzamento.
2. Caso exista, a estratégia comunica com o TrafficLight e força a transição imediata para GreenState, mesmo que o ciclo fixo ainda não tenha terminado.
3. Após a passagem da ambulância, o sistema retorna o ciclo normal.

5. Lógica de Simulação de Veículos

A classe Vehicle é onde tudo acontece. Em vez de os carros andarem simplesmente a uma velocidade fixa, fizemos com que eles reagissem ao que está à volta.

5.1. O Algoritmo de Deteção (Look-ahead)

Cada veículo possui um "vetor de visão". Em cada iteração do ciclo de simulação, o veículo executa o método scanEnvironment():

- **D**eteção de Veículos: O veículo verifica a lista de entidades no World. Se a distância para o veículo da frente for inferior à SafeDistance, é calculada uma força de travagem proporcional à proximidade.
- Interação com Semáforos: O veículo identifica o semáforo mais próximo na sua rota. Se o estado do semáforo for RedState ou YellowState, o veículo trata o semáforo como um obstáculo fixo e inicia o processo de paragem.

6. EmergencyVehicle

A classe EmergencyVehicle herda de Vehicle, mas introduz comportamentos únicos através de Polimorfismo.

6.1. Comunicação Objeto-a-Objeto

Ao contrário dos veículos normais a Ambulância atua como um Emissor de Eventos.

- Quando a posição da ambulância entra no perímetro de influência de um TrafficLight, ela invoca o método de interrupção na TrafficStrategy.
- O sistema de prioridade suspende o temporizador do estado atual e faz a transição para GreenState.

7. Monitorização e Extração de Resultados

Implementámos um sistema de log estatístico na classe StatisticsManager.

7.1. Variáveis Monitorizadas

O sistema regista em tempo real os seguintes indicadores:

- Waiting Time (Tempo de Espera): Contador acumulado de cada veículo sempre que a sua velocidade é inferior a 0.1 m/s.
- Throughput: Número de veículos que conseguem atravessar o cruzamento num intervalo de 60 segundos de simulação.
- Emergency Response Time: Tempo que um veículo de emergência demora a percorrer o cenário com e sem a estratégia de prioridade ativa.

7.2. Exportação para CSV

Utilizámos a biblioteca java.io.PrintWriter para gerar o relatório . Ao final de cada execução, o programa guarda um ficheiro chamado simulation_metrics.csv.

- Estrutura do Ficheiro: O ficheiro segue o formato Timestamp; VehicleID; AverageWaitTime; StrategyUsed.

8. Peões e Movimentação

Os peões são tratados como entidades independentes no World, mas com restrições de movimento diferentes dos veículos:

- Comportamento: Tal como os veículos, os peões verificam o estado do semáforo pedonal.

8.1. Lógica de Passadeiras (Crosswalk):

As passadeiras funcionam como zonas de interseção crítica.

- Deteção: Quando um peão entra na zona de limite de uma Crosswalk, o sistema dispara um sinal de ocupação.
- Prioridade: O Vehicle executa uma verificação constante. Se detetar um peão na passadeira o veículo para.

9. Gestão de Configuração e Controlo de Versão (Git)

O sucesso deste projeto deveu-se à utilização do Git para coordenar as contribuições dos três elementos do grupo.

- Divisão de Tarefas:
    - Francisco Moroni: Implementação da lógica física de movimento e arquitetura base do World.
    - Nuno Monteiro: Interface gráfica JavaFX e otimização da renderização no Canvas.
    - Mariana Pinto: Desenvolvimento dos padrões comportamentais (State e Strategy) e sistema de ficheiros.

10. Conclusão

O projeto cumpre todos os objetivos propostos. Conseguimos uma simulação realista onde carros, peões e ambulâncias funcionam. O ficheiro CSV gerado confirma que a nossa lógica de prioridade funciona e ajuda a reduzir o tempo de resposta. O código ficou limpo e pronto para, no futuro, receber novas funcionalidades como por exemplo rotundas.