ControladorAdaptativo

Introducción

`ControladorAdaptativo` es una simulación de control de tráfico que modela varios semáforos y flujos de vehículos. Su objetivo principal es comparar y evaluar políticas de selección de semáforos para optimizar el paso de vehículos, reduciendo colas y mejorando métricas de rendimiento en escenarios simulados. Resuelve el problema de decidir qué semáforo debe abrirse en cada paso temporal atendiendo a métricas como la longitud de cola y reglas de prioridad.

Descripción del Funcionamiento

El sistema está compuesto por los siguientes módulos principales:

- `Semaforo`: representa un semáforo con su cola de vehículos (FIFO) y operaciones para encolar y desencolar vehículos.
- `ArrivalGenerator`: genera llegadas de vehículos de forma estocástica y permite sembrar una cantidad inicial por semáforo.
- `Selector` (por ejemplo `MaxQueueSelector`): observa el estado de las colas y sugiere el semáforo candidato a procesar según una política.
- `Policy` (por ejemplo `CompareWithMaxPolicy`): decide si se procesa el semáforo seleccionado (permitir salidas) en función del selector y reglas adicionales.
- `TrafficRunner`: orquesta la simulación, ejecuta el bucle temporal (ticks), aplica la política, procesa las salidas en los intervalos configurados y recopila estadísticas.

Flujo general de la simulación:

1. Se crean N instancias de `Semaforo`.
2. `ArrivalGenerator` siembra una cantidad inicial de vehículos por semáforo y genera llegadas durante la simulación.
3. En cada tick, `Selector` evalúa el estado de las colas y propone un candidato.
4. `Policy` aplica la lógica de decisión sobre si permitir salidas en el semáforo candidato.
5. `TrafficRunner` procesa las salidas en los intervalos definidos y actualiza las métricas.
6. Al finalizar la duración configurada se devuelve un resumen con llegadas, salidas y colas finales por semáforo.

Implementación de Estructuras de Datos

Estructuras utilizadas y justificación:

- Arreglos (`Semaforo[]`): se usan para almacenar la colección de semáforos cuando su número es fijo al iniciar la simulación. Proporcionan acceso por índice de forma rápida y simple.
- Colas (`Queue<T>` o una cola propia dentro de `Semaforo`): modelan el orden FIFO de los vehículos en cada semáforo, que es el comportamiento natural de llegada/salida.
- Arrays de contadores (`int[]`): para almacenar estadísticas por semáforo (`ArrivalsPerSemaforo`, `DeparturesPerSemaforo`, `FinalQueue`) se usan arrays índices por semáforo.
- Tipos primitivos (enteros, TimeSpan): para tiempos, contadores y configuraciones de la simulación.

Justificación: la lógica actual requiere operaciones de encolar/desencolar y accesos por índice; por tanto las colas y arreglos son las estructuras más adecuadas. No se emplean árboles ni grafos en la implementación base; si se modelara una red de intersecciones con dependencias avanzadas, un grafo sería la estructura adecuada.

Capturas de Código y Resultados

Fragmento representativo de `Program.cs` (configuración y ejecución de la simulación):

```
// Crear semáforos
var semaforos = new Semaforo[N];
for (int i = 0; i < N; i++) {
    semaforos[i] = new Semaforo();
}

// Sembrar vehículos inicialmente
var generator = new ArrivalGenerator();
generator.Sow(semaforos, siembraInicial);

// Configurar selector y política
var selector = new MaxQueueSelector(semaforos);
var policy = new CompareWithMaxPolicy(umbral);

// Ejecutar simulación
var runner = new TrafficRunner(semaforos, generator, selector, policy);
runner.Run(ticks);
```

Resultados obtenidos (valores simulados):

- Semáforo 1: Llegadas: 150 | Salidas: 120 | Cola Final: 30
- Semáforo 2: Llegadas: 100 | Salidas: 80 | Cola Final: 20
- Semáforo 3: Llegadas: 200 | Salidas: 180 | Cola Final: 50

Ejemplo de salida en consola (valores simulados):

```
Semáforo 1:  Llegadas: 150 | Salidas: 120 | Cola Final: 30
Semáforo 2:  Llegadas: 100 | Salidas: 80  | Cola Final: 20
Semáforo 3:  Llegadas: 200 | Salidas: 180 | Cola Final: 50
```

Nota: los números varían entre ejecuciones por la naturaleza estocástica del generador de llegadas.

Conclusiones

Aprendizajes y retos:

- La simulación permite comparar políticas de control usando métricas sencillas (longitud de cola, tasa de salida) y validar hipótesis sobre la eficiencia de cada política.

- Coordinar generación de llegadas (procesos asíncronos) con un bucle de ticks determinista fue un reto para mantener coherencia en la simulación y facilitar reproducibilidad.

Mejoras posibles:

- Añadir configuración externa (JSON o parámetros de línea de comandos) para experimentar más fácilmente con parámetros.
- Implementar y comparar más políticas (tiempo fijo, rotación ponderada, políticas adaptativas o aprendizaje por refuerzo).
- Incluir visualización en tiempo real (gráficos o una UI) para monitorizar la evolución de colas.
- Extender la topología a una red de intersecciones modelada como grafo para simular dependencias y rutas más realistas.