Institución educativa:

Instituto Tecnológico y de Estudios Superiores de Monterrey

Alumno – Matrícula:

Jesús Eduardo García Luna – A01739060

Victoria Iluminda Rosales García - A01734739

Fecha:

22 de octubre de 2025

Materia o bloque:

Modelación de sistemas multiagentes con gráficas computacionales

Maestro(s):

Luciano García Bañuelos

Evidencia:

Modelo de tráfico simple

Modelo de tráfico simple

1) ¿Qué construimos?

Un cruce de dos calles (horizontal y vertical) con dos semáforos sincronizados y varios autos circulando en ambos sentidos (hasta 5 por calle).

El objetivo fue que los autos:

- Solo frenen cuando se acercan al semáforo y está en amarillo/rojo.
- No se empalmen (mantener una separación mínima para evitar “choques”).
- Mantengan un comportamiento suave (acelerar/desacelerar de forma gradual) y continúen normal después de cruzar.

Además, agregamos una vista en el frontend para ver el cruce, los semáforos y los autos en vivo, y un panel simple de métricas para revisar velocidades promedio por calle y capturar resultados para escenarios con 3, 5 y 7 autos por calle.

2) ¿Cómo funciona, a grandes rasgos?

- Frontend (React / App.jsx)

  Muestra el cruce, dibuja los semáforos y los autos (ícono pequeño). Tiene botones de Setup / Start / Stop, un control para la velocidad de muestreo y un selector de autos por calle (3, 5, 7). También incluye un panel con promedios de velocidad y una bitácora donde podemos guardar las mediciones de cada escenario.

- Backend (Genie / webapi.jl)
  - Expone:
    - POST /simulations para crear una simulación (puedes pasar cars_per_lane).
    - GET /simulations/:id para avanzar 1 “tick” y regresar el estado (semáforos, autos y métricas).
    - Modelo (Agents.jl / simple.jl)

      Dos tipos de agentes:

      - TrafficLight: semáforo con ciclo México Verde (10) → Amarillo (4) → Rojo (14); total 28 ticks.
      - Vehicle: auto con dirección (:EW o :NS), velocidad actual y velocidad objetivo.

        Usamos un scheduler por tipo para que primero se actualicen los semáforos y después los autos (evita que un auto tome decisiones con el estado viejo del semáforo).

3) Reglas principales (sin enredarnos)

- Zona de frenado cerca del semáforo

  Definimos una distancia llamada STOP_ZONE. Los autos solo empiezan a frenar si el semáforo no está en verde y están dentro de esa zona.

  Lejos del cruce, no frenan, aceleran suavemente hacia su velocidad objetivo.

- Distancia de seguridad entre autos (no empalmarse)

  Siempre calculamos la distancia con el auto de adelante en la misma vía. Si el hueco es pequeño, el auto reduce su velocidad para mantener un mínimo (SAFE_GAP).

  Como el mundo es periódico (tipo “toro”), calculamos el hueco hacia adelante con wrap-around, para que funcione incluso cuando un auto está “casi al final” y el otro “apenas al inicio” del carril.

- Aceleración y frenado suave

  Dos constantes (ACC y DEC) controlan cuánto sube o baja la velocidad por tick. Así evitamos cambios bruscos.

- Spawning sin traslape en el inicio

  Al crear los autos, se elige una posición aleatoria fuera del cruce y se verifica que no queden pegados a los que ya están en la misma vía. Si no hay espacio, se vuelve a intentar.

4) Decisiones de diseño que hicieron la diferencia

- Orden de actualización: ByType((TrafficLight, Vehicle), false) para asegurar que primero actualicen los semáforos y luego los autos.
- Nada de “frenar desde el principio”: la condición de frenado depende de la STOP_ZONE. Esto resolvió el problema de autos detenidos a kilómetros del semáforo.
- Propiedades del modelo accesibles y seguras: guardamos el tamaño del mundo como model.Lx y model.Ly (en Float64) para evitar problemas de tipos y no depender de model.space en tiempo de ejecución.
- Evitar empalmes: cálculo de “auto de adelante” con wrap-around + limitación de velocidad según SAFE_GAP.
- Compatibilidad con el frontend: mantuvimos el formato JSON (cars y lights) para no romper la UI.

5) Cómo correr todo (paso a paso)

1. Backend (Julia)
   a. Abre webapi.jl en VS Code o el REPL.
   b. Ejecuta el archivo. El servidor abre en http://localhost:8000.
   c. Dependencias: Agents, StaticArrays, Genie, Random (Random viene en Julia).
2. Frontend (React)
   a. En la carpeta del front: npm install

b. Luego: npm run dev
c. Abre la app en tu navegador.
d. Flujo:

a) Elige autos por calle (3, 5 o 7)

b) Pulsa Setup (se crea la simulación)

c) Pulsa Start (empieza el loop de actualización)

d) Puedes ajustar la frecuencia de muestreo (Hz) y pulsar Stop cuando quieras.

6) Visualizaciones y métricas

En la parte derecha/abajo verás:

- Promedios instantáneos de velocidad por calle (EW y NS).
- Promedio acumulado (se va actualizando mientras corre el escenario actual).
- Bitácora de capturas: cuando estés satisfecho con la estabilización, pulsa Guardar muestra. Repite para 3, 5 y 7 autos por calle y tendrás tres filas con tus resultados.

Nota: No incluimos números “duros” en el documento porque dependen de tu corrida. La idea es que tú captures tus promedios y los reportes con la tabla.

7) Parámetros que puedes mover si quieres “tunear”

- ACC (aceleración) y DEC (desaceleración): velocidad a la que sube/baja el “speed” por tick.
- STOP_ZONE: qué tan lejos del semáforo empieza a frenar el auto si ve amarillo o rojo.
- SAFE_GAP: distancia mínima con el auto de adelante.
- LOOKAHEAD: cuánto “alcanza a ver” al auto de adelante.
- Rango de velocidad objetivo al inicio (por defecto entre 0.4 y 0.9).
- Tamaño del mundo y escala del SVG en el front.

8) Problemas comunes y cómo los resolvimos

- “No puedo redefinir Car”

  Si el REPL ya tenía un tipo con ese nombre, lo renombramos a Vehicle o reiniciamos el REPL.

- Choque con Base.instances

  Cambiamos el diccionario global de simulaciones a sim_store en webapi.jl.

- model.properties / model.space

  Evitamos accesos ambiguos. Usamos model.cx, model.stop_x_ew, etc., y guardamos model.Lx/model.Ly en Float64 para cálculos robustos.

- Autos frenándose desde el inicio del carril

Introdujimos STOP_ZONE para que solo frenen cerca del semáforo.

- Empalmes

  Cálculo de hueco hacia adelante con wrap-around + SAFE_GAP y una verificación en el spawn para no nacer pegados.

9) Qué entregamos y cómo usarlo

- Código del modelo (simple.jl) con semáforos y múltiples autos por calle.
- API (webapi.jl) con rutas POST/GET, serialización plana (cars, lights) y métricas.
- Frontend (App.jsx) con selector de autos/calle, controles, vista del cruce y panel de métricas.

Con esto puedes:

- Correr escenarios con 3, 5 y 7 autos por calle.
- Capturar los promedios y compararlos.
- Ajustar parámetros si quieres experimentar con “tráfico denso” vs “ligero”.

10) Siguientes pasos (si damos otro salto)

- Agregar más carriles por sentido (dos por lado) y reglas de cambio de carril simples.
- Introducir ruido: distintos perfiles de conductor (prudente, agresivo).
- Guardar las métricas en CSV desde el backend.
- Añadir gráficas en el frontend (línea o barras) para ver la evolución de la velocidad y comparar escenarios.

11) Reflexiones

Reflexión — Jesús Eduardo García Luna

Al principio me atoré con cuándo debían frenar los coches. Si frenaban desde que el semáforo no estaba en verde, se quedaban parados desde muy lejos y se veía raro. Con la zona de frenado ya cerca del cruce todo cambió: avanzan normal y solo se preparan al final. También me costó el tema de que el mundo “da la vuelta”; entender ese wrap-around me ayudó a que no se empalmaran. Y ordenar que primero actualicen los semáforos y luego los autos evitó que “se comieran” el alto. Me quedo con esto: con pocas reglas bien puestas el modelo se siente mucho más real.

Reflexión — Victoria Iluminda Rosales García

Yo me enfoqué en que el tráfico se viera natural sin meterle demasiada ciencia. Acelerar y frenar de forma suave, y darles velocidades objetivo distintas ya hace una gran diferencia. También noté que si no respetas el orden (primero luces, luego autos), salen cosas raras. El panel para capturar promedios me sirvió para comparar rápido 3, 5 y 7 coches por calle. Siguiente paso que me gustaría: dos carriles por sentido y probar cambios de carril sencillos.