INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE MONTERREY

CAMPUS PUEBLA

Modelación de sistemas multiagentes con gráficas computacionales (Gpo 501)

TC2008B.501

Modelo de tráfico simple

Profesor:

Luciano García Bañuelos

Roberto Castro Gómez | A01425602

Sebastian Ponce Vaquero | A01737978

22 de Octubre de 2025

Etapa 1: Cruce con semáforos

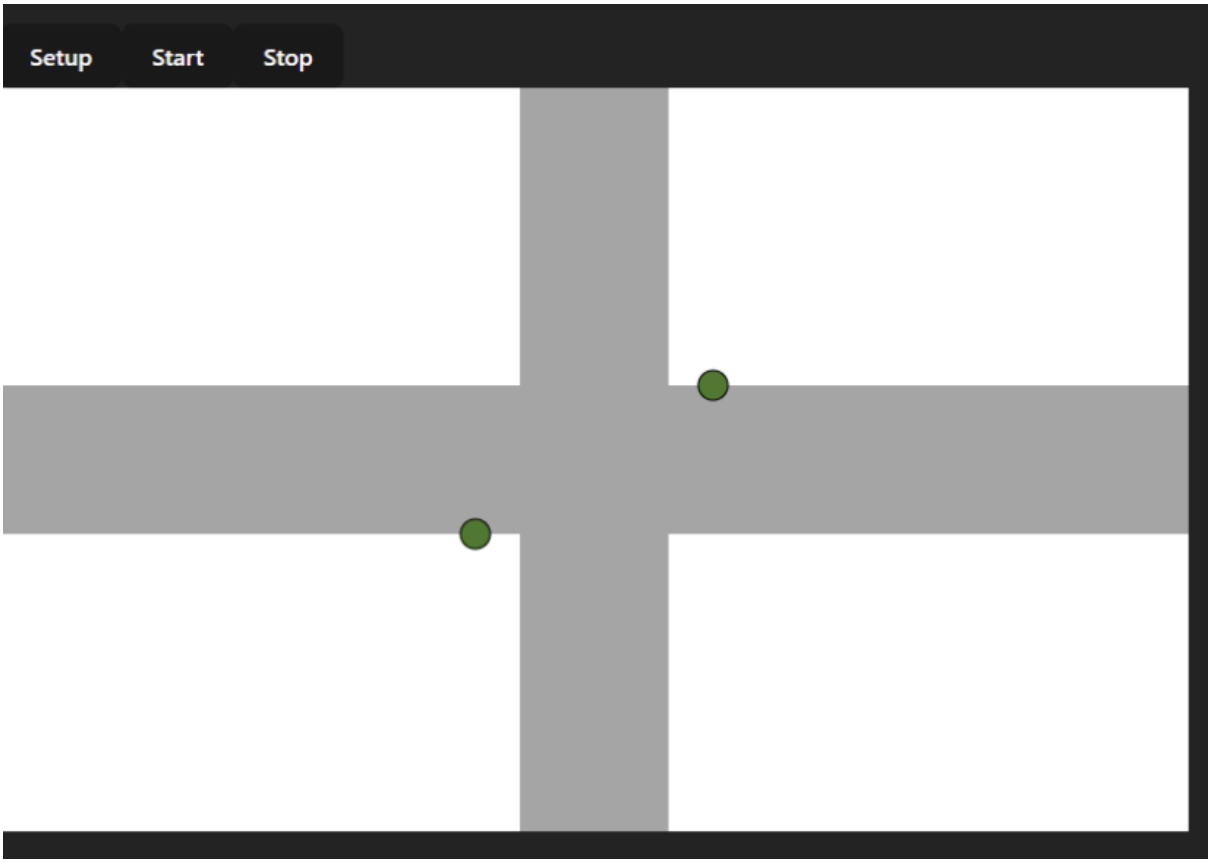
Para implementar únicamente el comportamiento de los semáforos en la intersección, utilizamos un `struct` para definir un nuevo tipo de agente: `TrafficLight`. Este agente cuenta con una ubicación en el espacio continuo del modelo, además de un temporizador para marcar los pasos restantes antes de cambiar de color al siguiente en el ciclo.

En cada paso de `agent_step!`, el temporizador `timer::Int` se decrementa. Cuando llega a 0, el color `color::LightColor` actual del semáforo es modificado. Para ello creamos un `enum` especial `LightColor` con los 3 colores, donde cada uno estaba asociado a la cantidad de pasos que se mostraría antes de cambiar al siguiente color: `Green`, `Yellow`, y `Red`.

```
@enum LightColor Green Yellow Red


@agent struct TrafficLight(ContinuousAgent{2,Float64})
    color::LightColor
    timer::Int
end
```

Ejecución:

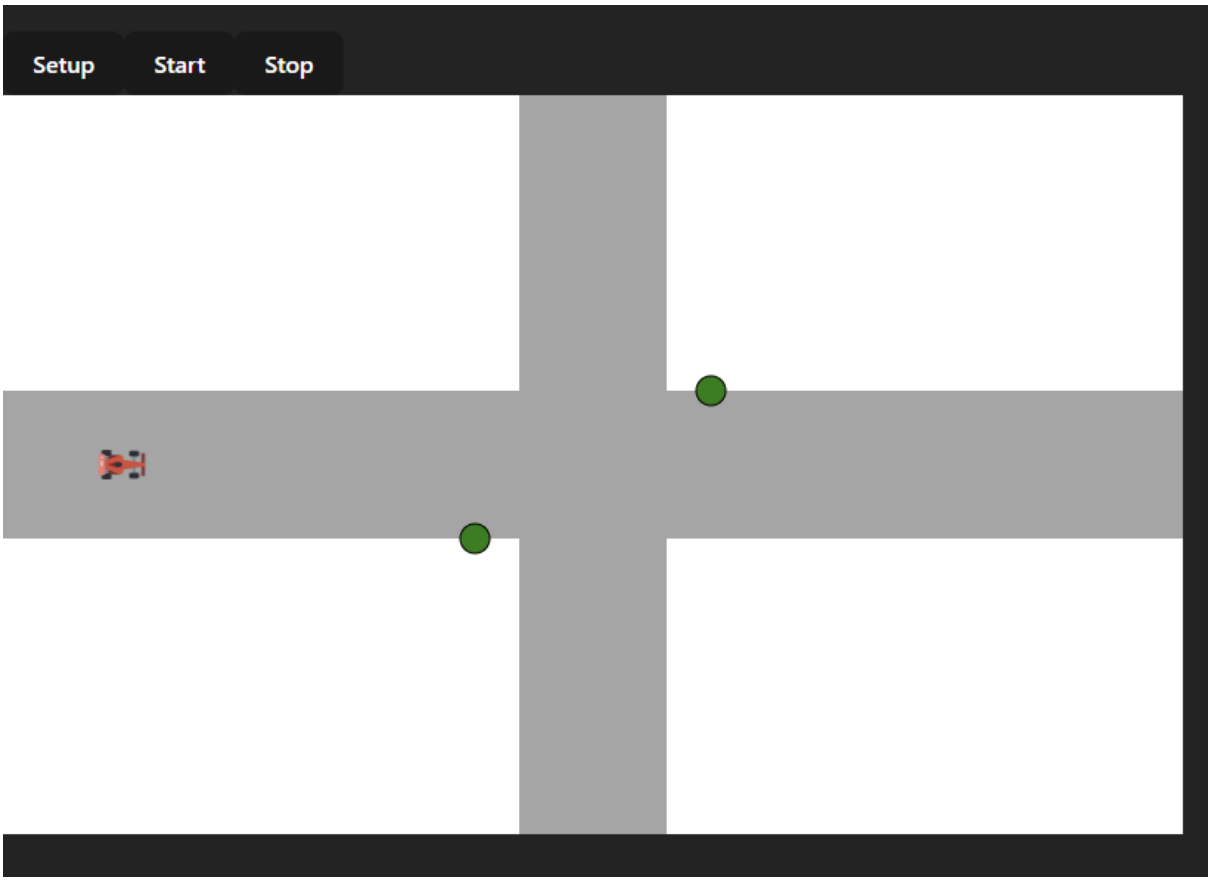


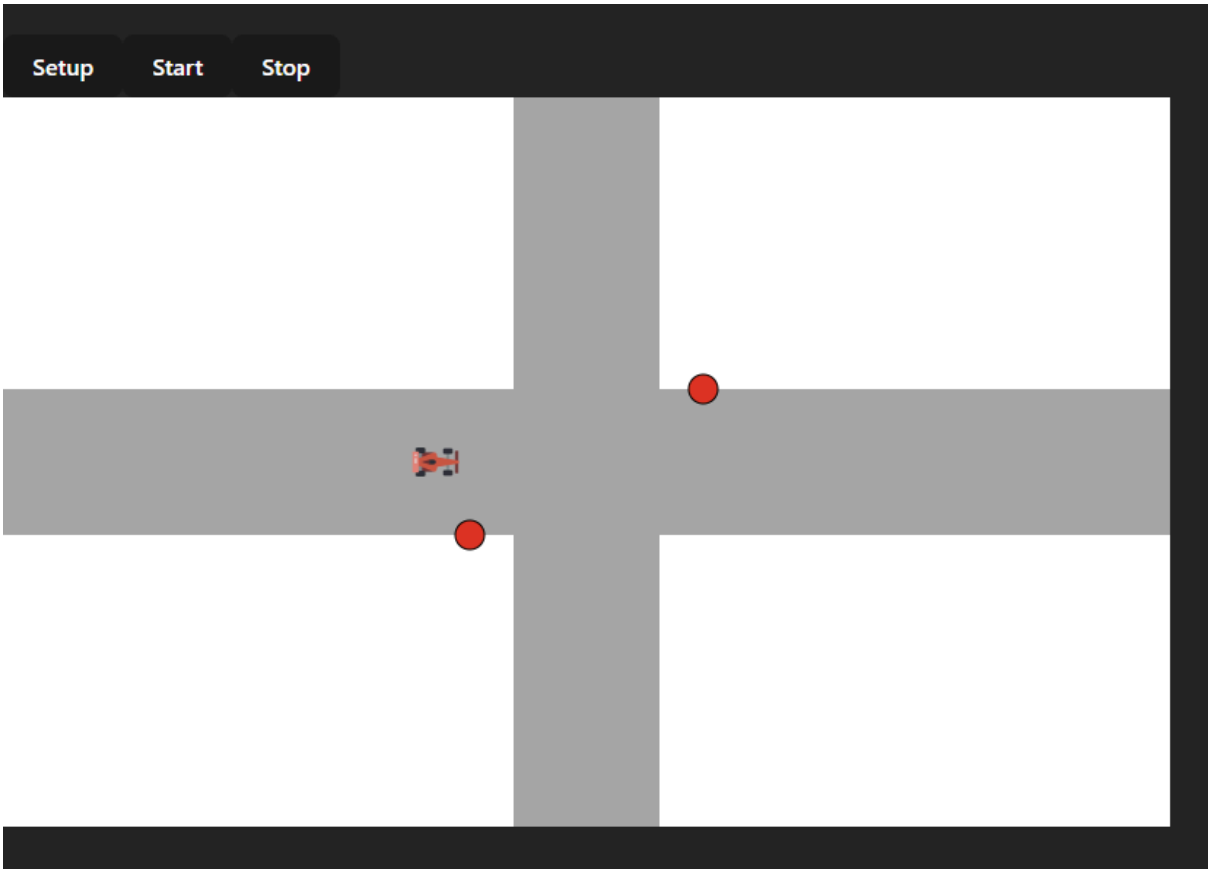
Etapa 2: Un solo auto en la implementación

En este caso agregamos la primera etapa de interacción entre el agente de vehículo y los semáforos. Para simplificar el problema, únicamente se utilizó un vehículo que se maneja de manera horizontal. A partir de una iteración con distintos errores relacionados a utilizar `Union` junto con tipos de agentes distintos en Julia, decidimos crear un "meta-agente" que fusionaba los atributos de ambos agentes. Este agente, `TrafficAgent`, cuenta con un atributo de `role` para definir si el agente es un semáforo o un vehículo. Finalmente, agregamos reglas sencillas para definir el comportamiento requerido (detenerse en una luz roja o amarilla dentro de una distancia de frenado).

Además, agregamos un atributo de `direction::LightDirection` para definir si el semáforo afectará a los vehículos en el camino horizontal o en el camino vertical. Para sincronizar el comportamiento de manera correcta, agregamos un "wrapper" para `agent_step!`, llamado `step_model!`. `step_model!` nos permitió asegurarnos de que estamos actualizando primero los semáforos y después los vehículos. De esta forma, garantizamos que los semáforos sean actualizados antes de que los vehículos decidan frenar o no.

Ejecución:





Etapa 3: Completemos la simulación

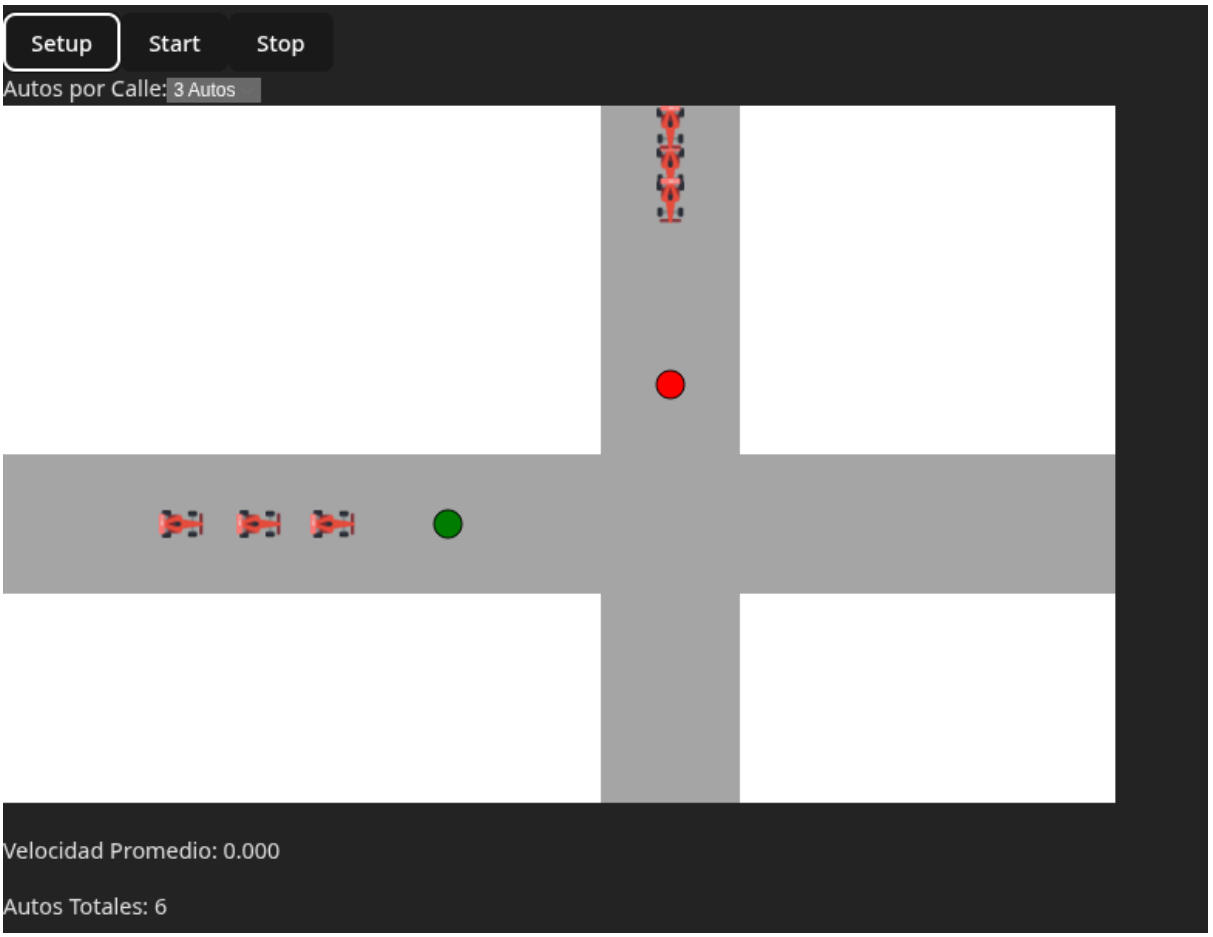
Para esta etapa, se implementó un aumento en el número de vehículos por calle, así como la detección de vehículos para evitar las colisiones y una implementación de la velocidad promedio de los vehículos.
Para implementar la primera parte del problema, se implementó en el frontend una selección de número de vehículos con valores 3, 5 y 7.

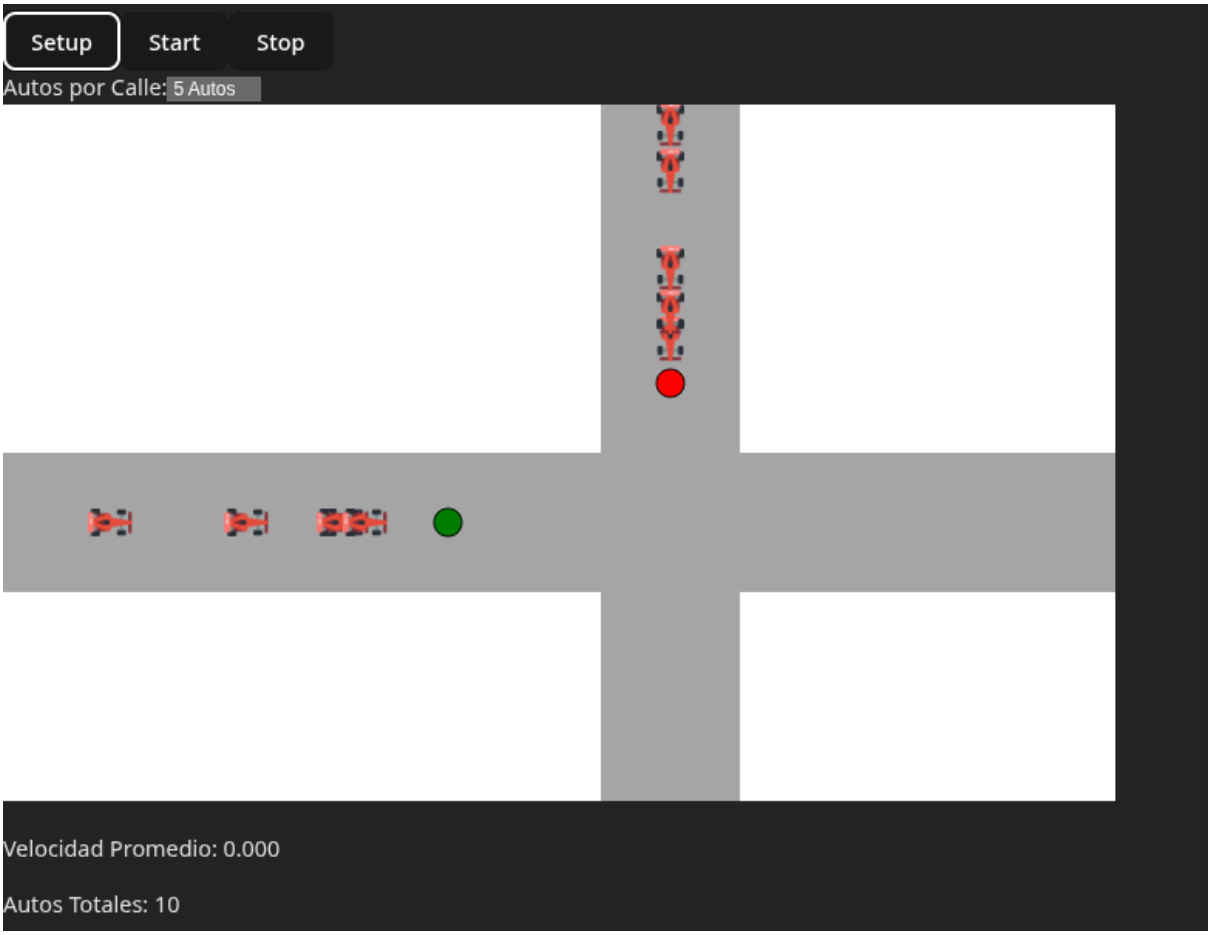
```
<option value={3}>3 Autos</option>
<option value={5}>5 Autos</option>
<option value={7}>7 Autos</option>
```

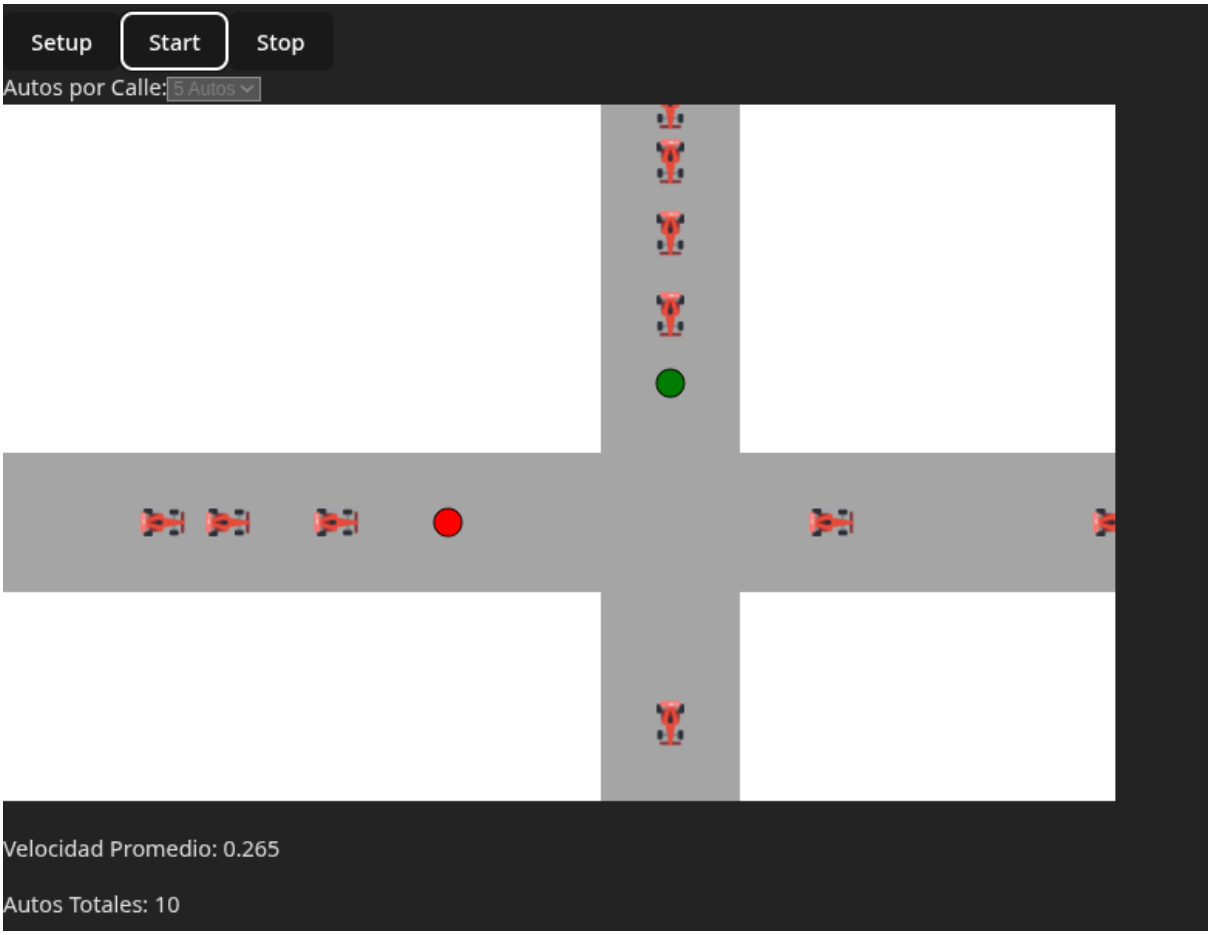
Estos valores eran enviados al backend donde utilizando un ciclo para los n coches, se colocaban en posiciones aleatorias previas al semáforo correspondiente, adicionalmente se implementó un identificador de horizontal y vertical para poder definir las direcciones de los vehículos.
Para la detección de vehículos se tuvo que implementar una detección de todos los vehículos en el carril (misma dirección) y una detección de aquellos vehículos que estuvieran adelante, esto mediante cálculos de distancias con una referencia de un valor seguro de frenado.

Finalmente para la implementación de la velocidad promedio, se tuvieron que obtener los valores de la velocidad correspondiente a cada coche en el webapi.jl para su correcto cálculo de promedio.

Ejecución:




Setup
Start
Stop
Autos por Calle:
5 Autos
Velocidad Promedio: 0.000
Autos Totales: 10



Repositorio de Github:

https://github.com/Otrebor280971/Traffic_simulation