Tecnológico de Monterrey

Reflexión

Andres Eduardo Nowak de Anda A01638430

12/01/2021

Introducción:

En esta reflexión hablaremos de cómo es que funciona nuestro programa, el backend y el front end y también sobre los datos que obtuvimos al correrlo.

Backend:
Para el backend decidimos utilizar websockets, ya que los websockets es algo que fue diseñado para conexión en tiempo real, no como http y aparte es mas eficiente porque no tiene que hacer un wrap para poder mandar todo el html, si no aqui solo mandamos el json y solo el json y también sentíamos que podía hacer más facil la simulación.

Después la ciudad para crearla primero creamos un objeto ciudad, en donde aquí agregamos calles de un punto a otro y donde se intersectan es donde se agregarían los semáforos y aparte estas calles contienen los datos de direcciones para que los coches sepan a donde se pueden mover en cada punto. Después creamos nuestra ciudad modelo que recibe esta ciudad como parámetro y en base a esta ciudad creamos nuestro modelo, los coches deciden de manera random donde quieren aparecer de una lista de spawn points ya definida, y los semáforos los ponemos donde hay intersecciones. En este modelo hay tres agentes; semáforos, carros y calles, las calles existen solo para poder darle información al carro de cómo se puede mover, el carro checa los momentos en los que se puede mover (checa que si hay coches en frente o si hay un semáforo para saber si se puede mover), después tenemos los semáforos que estos se basan en un modelo de master y seguidores, cuando llegan los coches a las intersecciones uno de los semáforos ganará poder ser el main, y los demás semáforos se basaran en el main para saber qué es lo que pueden hacer.

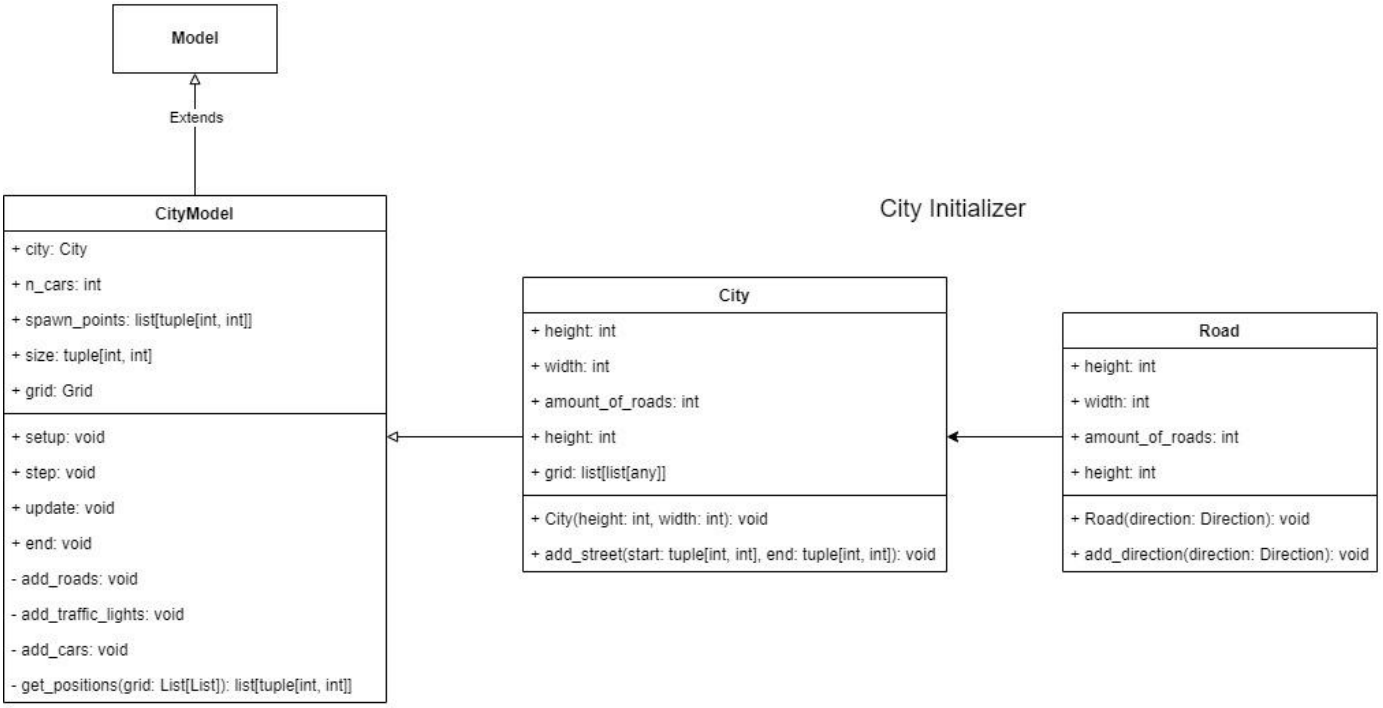
Front end:
Como nuestro backend puede crear ciudades con diferentes formas, hicimos que la ciudad en unity se pueda instanciar en base a la información del backend, con esto primero creamos nuestra ciudad y creamos su nav mesh, despues instanciamos los prefabs y semaforos y comenzamos a hacer las peticiones seguidas al server para que nos de las nuevas posiciones y en base a esto hacer los cambios en el modelo de unity (mover los coches y cambiar de color los semaforos), de esta manera podemos tener codigo que no solo se basa en un solo diseño. Y para las calles para poder crearlas usamos tiling y offset para poder repetir las texturas de las calles, para mover las llantas utilizamos rotación y usamos post processing para aplicar efectos a la cámara.

Resultados:
De resultados pudimos ver que nuestra simulación en base a diez coches llega a tardar en promedio 0.023 segundos y tarda 75 steps, y el tiempo dedicado en los cruces es de 23 steps aproximadamente.
Como pueden ver llega a tardar más steps que la cantidad de coches que hay porque los semáforos llegan a funcionar solo en base a un tiempo hardcodeado, todos corren por esa misma cantidad de tiempo, entonces aunque ya no haya coches en la otra intersección, se quedan esperando aunque no sea necesario.
Ahora para poder reducir esta cantidad de tiempo, podríamos hacer que negocien los tiempos que tienen que durar, por decir en mi proyecto de la m3 hice que los semáforos chequen que tienen un coche en su carril y cada vez que ven que tienen uno, multiplican el tiempo restante que tienen de rojo * 0.93 y cuando al semáforo le llega a 0 el tiempo de rojo le dice al otro que se pongo rojo y yo me pongo verde, en este caso no lo hicimos por como ya habíamos implementado los semáforos y la lógica que había hecho en mi codigo no había quedado tan bien, entonces al final no lo llegamos a usar, pero esa sería una manera, el chiste sería hacer que los semáforos negocien.

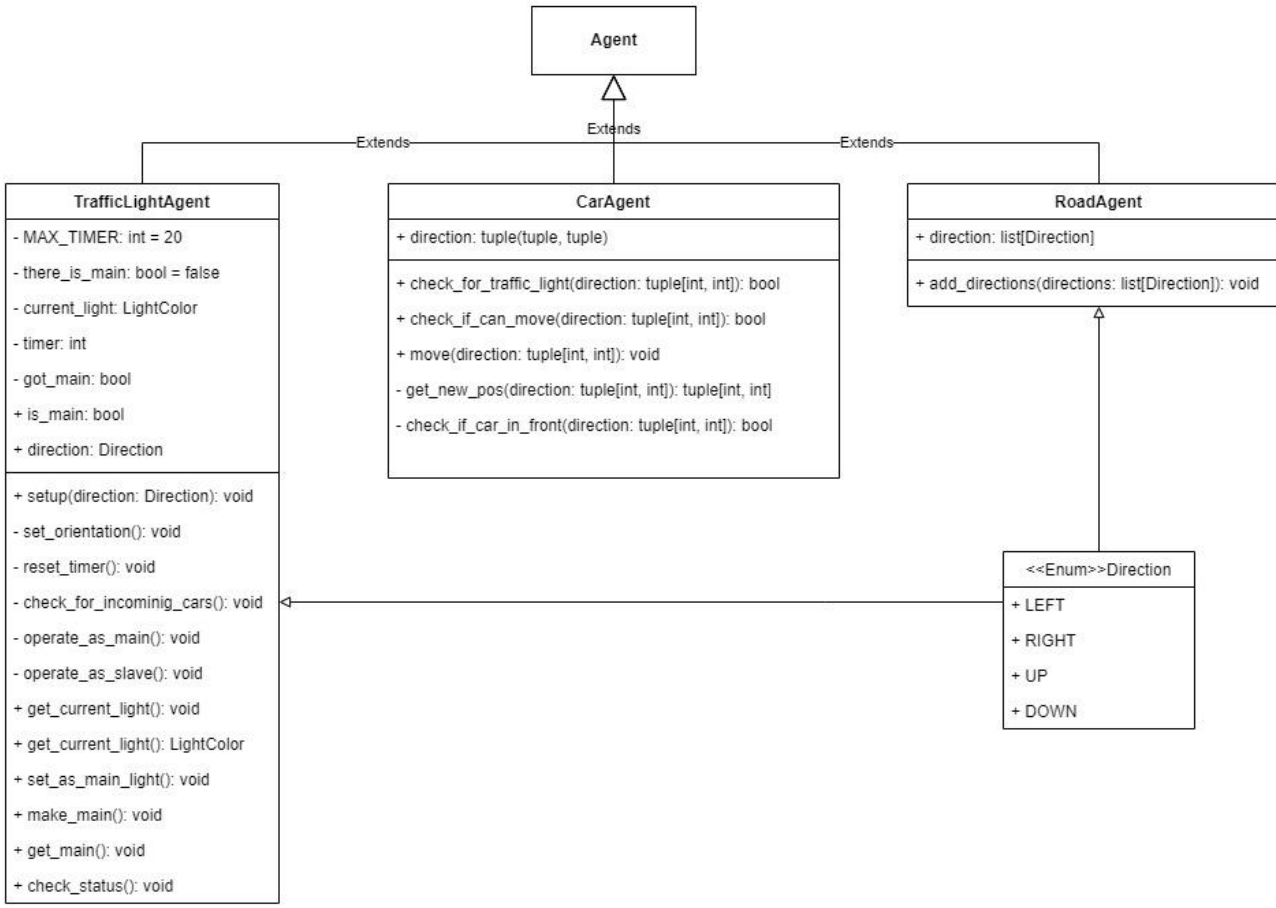
Estrategia cooperativa:
Para poder diseñar este proyecto, tuvimos que hacer divisiones de trabajos, unos en unity, otros en el backend y otros en el server, y muchas veces luego todos nos concentramos en una parte y luego otra vez todos iban a sus respectivas áreas y así es como también todos sabemos como funciona el codigo ya que todos ayudamos en cada parte.

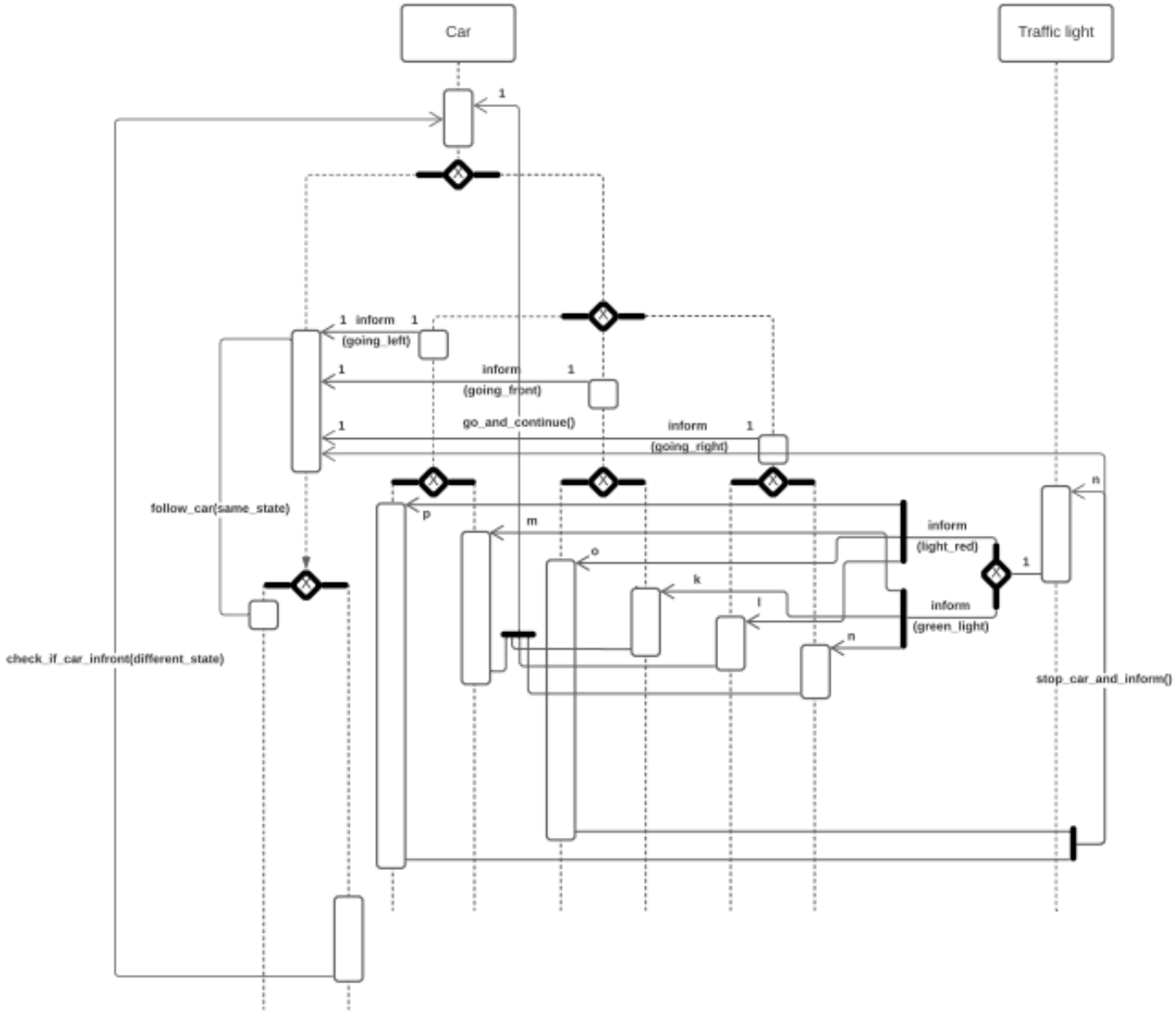
Diagramas de clase e interacción entre agentes:
En el diagrama podrá ver que a calle no lo pusimos como un agente y es porque es más como un agente pasivo, ya que en realidad no hace nada solo está ahí para que el coche sepa a dónde puede ir.

City Model



Agent Model




Car
Traffic light
1
1 inform 1
(going_left)
1
inform 1
(going_front)
1
go_and_continue()
inform 1
(going_right)
follow_car(same_state)
p
m
o
k
l
n
inform
(light_red)
1
n
inform
(green_light)
check_if_car_infront(different_state)
stop_car_and_inform()