Diagrama de Clases y Protocolos de Interacciones - Reto Movilidad Urbana

Jesús Sebastián Jaime Oviedo A01412442

Sergio Hiroshi Carrera Monnier A01197964

Juan Daniel Rodríguez Oropeza A01411625

Jackeline Conant Rubalcava A01280544

William Frank Monroy Mamani A00829796

Premysl Pilar A01760915

Monterrey, Nuevo León, México

Diciembre 03, 2022

Ing. César Raúl García Jacas

Ing. Sebastián Ulises Adán Saldívar

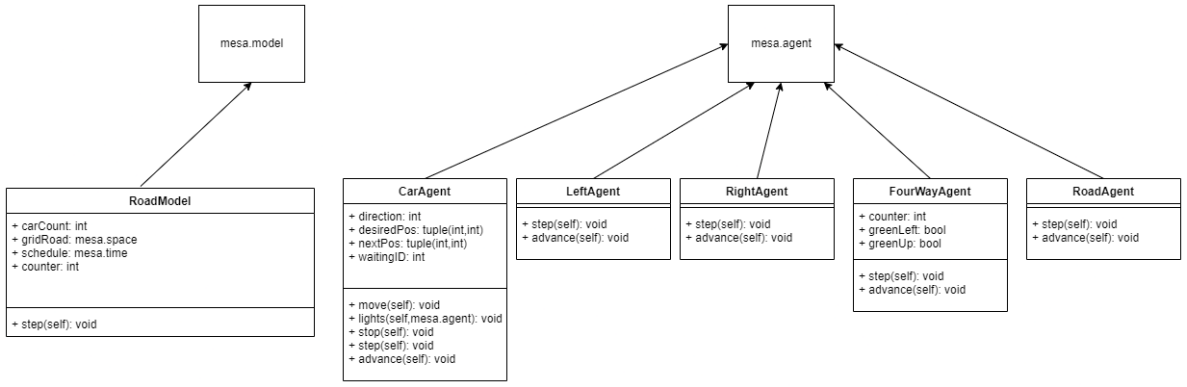
Instituto Tecnológico y de Estudios Superiores de Monterrey

Modelación de Sistemas Multiagentes con Gráficas Computacionales (Gpo 301)

Diagrama de clase - Planificación

Para la creación del diagrama de clase que se utilizó dentro del programa de movilidad urbana se crearon 6 clases que están divididas en los siguientes tipos de mesa: mesa.model(el modelo de la mesa) y mesa.agent (los agentes que hay dentro de la mesa). Dentro del modelo de la mesa se puede observar la clase RoadModel, cuenta con 4 variables, una entera contadora para los carros “CarAgent”, una mesa.space ‘gridRoad’ que son las coordenadas, una mesa.time ‘schedule’ y una variable entera contadora(count) y finalmente cuenta con un procedimiento llamado step.

Dentro de los agentes de la mesa, podemos ver 5 clases agentes llamados CarAgent, LeftAgent, RightAgent, FourWayAgent y RoadAgent. El primero y más importante es el CarAgent, este es el que controla al automóvil, a través de sus variables enteras de dirección(direction), posición deseada(desiredPos), siguiente posición(nextPos) y ID de espera(waitingID), puede hacer los procedimientos, de move, lights, stop, step y advance. Después está el LeftAgent, Right Agent y RoadAgent que tienen los procedimientos de step y advance, su utilidad es de saber la ubicación de avance y step, realmente no tienen efecto en nuestro CarAgent todavía. Como último está el FourWay Agent, que cuenta con variables entero contador(count) y booleanas para las luces greenLeft y greenUp, que tienen igualmente los procedimientos de step y advance.



Posteriormente podemos ver cómo se conectan los pasos o “steps” de la mesa de agentes “mesa.Agent” a lo que es el modelo de la mesa o el camino vehicular de nuestro proyecto.

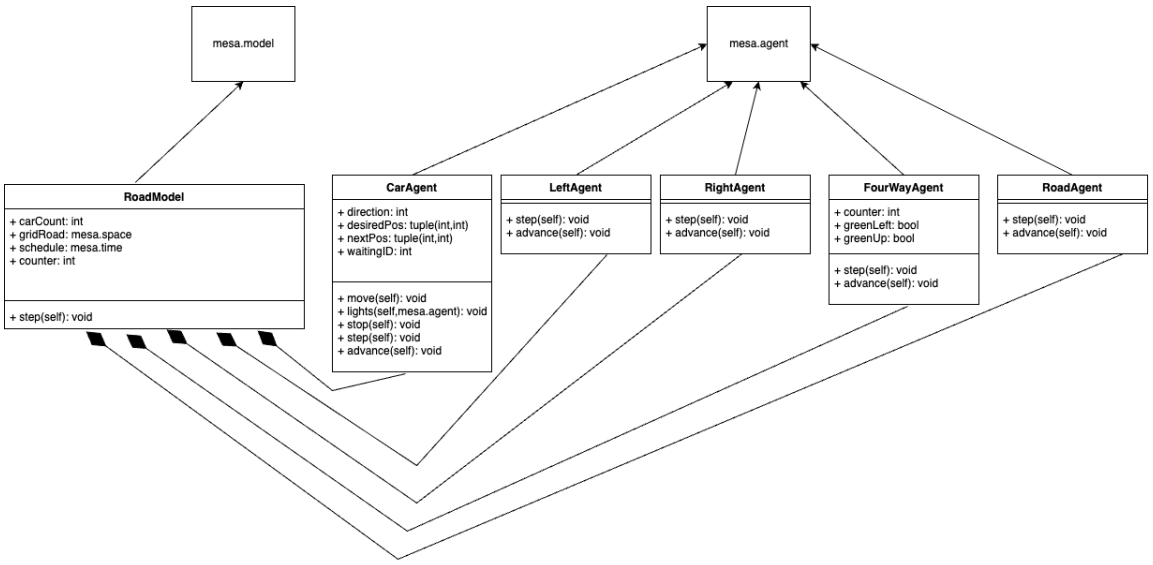


Diagrama de Protocolos / Interacciones - Mesa

Posteriormente a la creación del diagrama planificador de la Mesa se creó el diagrama de Interacciones o actividades donde se va a capturar el comportamiento interactivo que tiene el sistema de multiagentes que se va a implementar después dentro del Reto.

Inicialmente contamos con el modelo de la Mesa, donde empieza nuestro diagrama de interacciones, este a continuación crea lo que conocemos como una matriz, donde es necesario implementar coordenadas por donde van a pasar nuestros automóviles, para posteriormente moverse a la acción de dar un paso(step). Dentro de la acción paso(step) esta se puede crear igualmente una repetición o mover a lo que es el agente FourWayAgent donde se cambiarán las luces en un loop.

Después se moverá a la ramificación de una decisión “¿hay carros presentes?” con dos resultados, si y no. Si la decisión o respuesta es sí entonces se moverá al agente CarAgent donde se determinará la posición deseada del carro, pero si la respuesta es no entonces se moverá al agente RoadAgent, donde se determinará si hay un camino donde posicionar el carro, para después moverse al agente CarAgent y poner el carro dentro de la matriz creada.

Nuevamente se moverá una ramificación de decisión “Posición deseada en” con dos resultados, “Es carro” y “No es carro”. Si la respuesta es “Es carro” entonces se encontrará entre el modelo de la mesa y el carAgent determinando si otro carro se está moviendo, posteriormente se devolverá a la ramificación de una decisión “Posición deseada”. Si la respuesta es “No es carro” entonces se moverá a la siguiente ramificación de decisiones que es “Posición deseada en” con cuatro resultados “LeftAgent”, “RightAgent”, “Intersección” y “Carretera”. Si la respuesta es “LeftAgent” entonces se va a mover al agente LeftAgent y va a generar una acción de moverse y dar la vuelta a la izquierda. Si la respuesta es “RightAgent” entonces se va a mover al agente RightAgent y va a generar una acción de moverse y dar la vuelta a la derecha. Si la respuesta es “Carretera” entonces se va a mover al agente RoadAgent y va a generar una acción de moverse hacia delante.

Finalmente si la respuesta es “Intersección” se va a mover al agente “FourWayAgent” dónde está la ramificación de una decisión “Luz Verde”, donde las

respuestas son sí y no. Si la respuesta es sí se moverá hacia delante y si la respuesta es no va a detenerse.

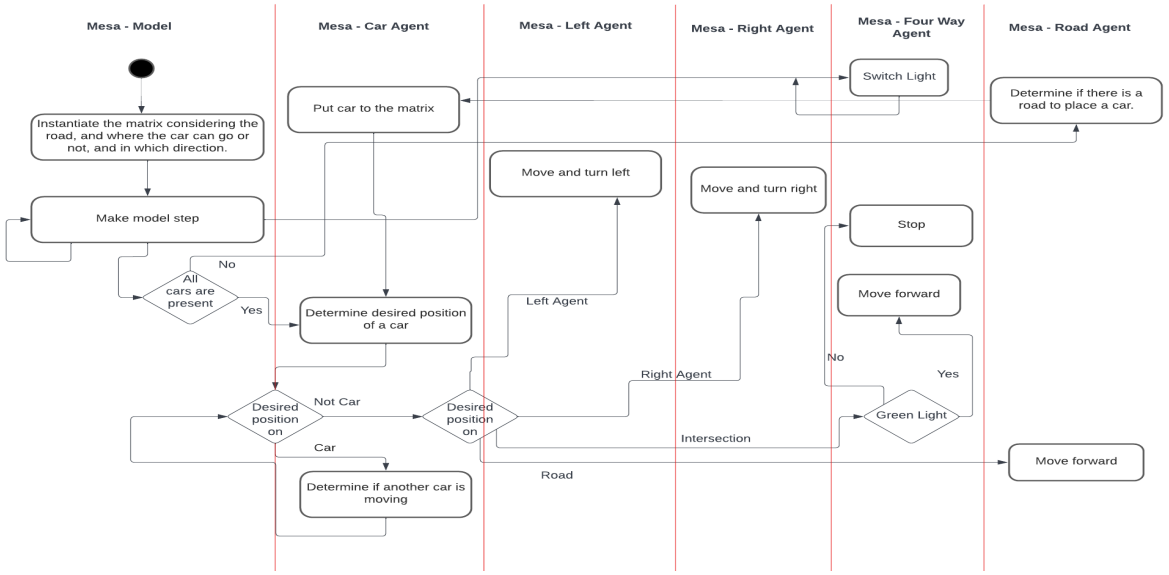


Diagrama de clase - Unity

Para la creación del diagrama de clase que se utilizó dentro del programa de movilidad urbana de Unity se crearon 4 clases, Singleton, LlamarApi, Waypoints y SemaphoreScript(El script del semáforo).

Dentro de la clase Singleton existe una variable pública static T llamada instance, dos procedimientos públicos, Instance y Awake() y dos procedimientos privados, SetupInstance() y RemoveDuplicates().

Después está la clase llamada API que hereda a la clase Singleton y manda información a las clases Waypoints y el Semaphore Script. Donde hay una variable privada string llamada uri, 5 variables públicas, una raíz variables Guardar, un GameObject semáforo, un GameObject[] carros, una variable float que es el timer y una variable entera llamada tempora. Esta clase cuenta con 4 procedimientos, Start(), Update(), mesa() y CreateGetRequest() y está conectada a 3 clases, Agent Root y Semaphore.

Posteriormente está la clase Waypoints que recibe datos de la clase LlamarApi, donde hay 4 variables públicas, una lista(list<GameObject>) waypoints, una float de rapidez(speed), una variable entera index y una variable entera carl. Después tiene 3 variables privadas, un boot del semaforo(semaphore), un GameObject de la luz(light) y la llamada del api LlamarApi. Como último posee 2 procedimientos, uno Start() y FixedUpdate().

Finalmente está la clase SemaphoreScript que recibe datos de la clase LlamarApi, donde hay 7 variables públicas, un GameObject red_light, un GameObject yellow_light, un GameObject green_light, un GameObject red_light_2, un GameObject yellow_light_2, un GameObject green_light_2 y la llamada del api LlamarApi. Como último tiene 3 procedimientos, uno que es Start(), ChangeLights() y ChangeLights_2().

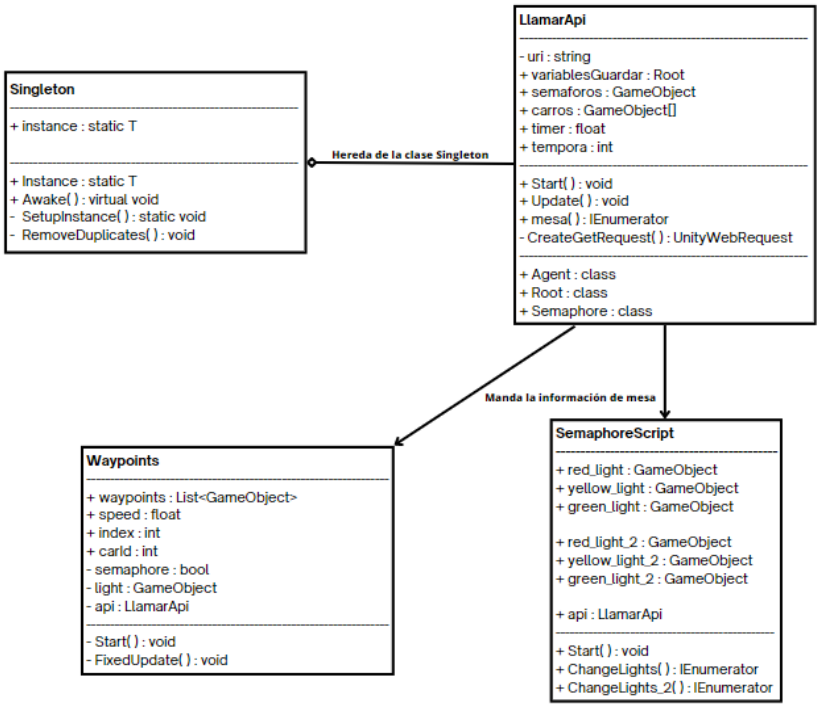


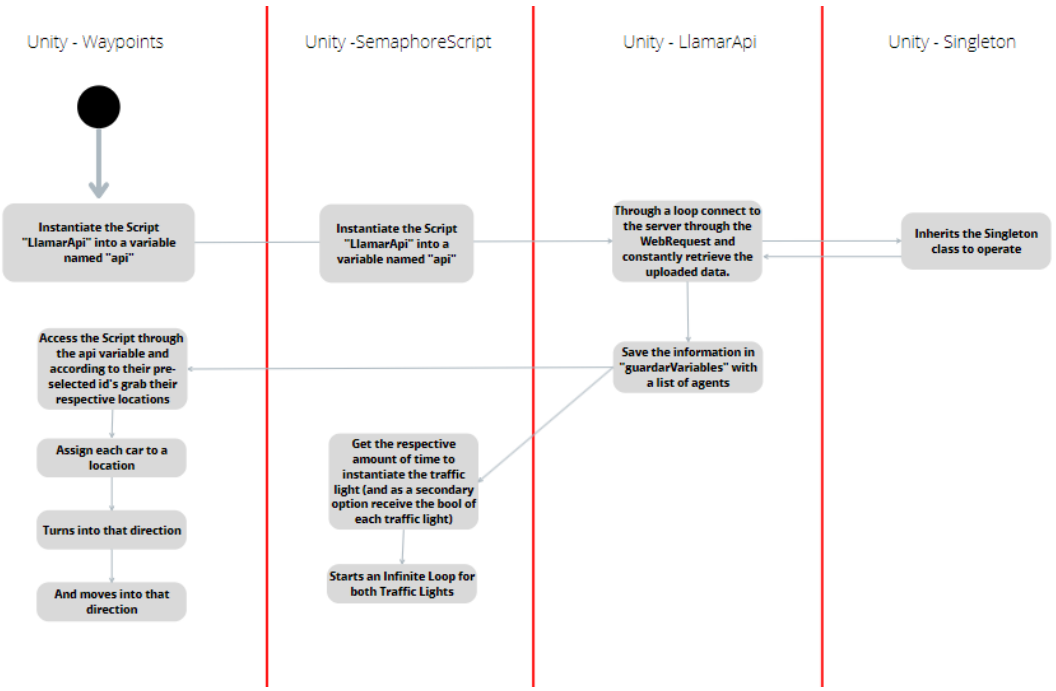
Diagrama de Protocolos / Interacciones

Posteriormente a la creación del diagrama de movilidad vehicular de Unity se creó el diagrama de Interacciones o actividades donde se va a capturar el comportamiento interactivo que se va a implementar dentro del modelo gráfico.

Inicialmente contamos con el modelo de Unity, donde empieza nuestro diagrama de interacciones dentro del canal Waypoints, donde se moverá a la acción de instanciar el código(Script) “LlamarApi” a las variables con el nombre de “api”, posteriormente se moverá al al Canal Semaphore Script para volver a hacer la acción de instanciar el código(Script) “LlamarApi” a las variables con el nombre de “api”, después nos iremos al canal “LlamarApi” donde vamos a incrementar la acción loop donde vamos conectar el servidor dentro de Webrequest y constantemente recoger la información que fue subida. Posteriormente a través del loop se va a poder llamar al canal Singleton donde se va heredar el Singleton de la clase operate.

Dentro del Canal “LlamarApi”, después del loop vamos a poder salvar la información en la variable guardar variables con la lista de agentes, dependiendo de la información vamos a poder movernos al “SemaphoreScript” donde se va a implementar la acción de conseguir la cantidad respectiva de tiempo Instanciada a la luz del tráfico(Y como opción secundaria recibir la luz de tráfico de cada booleano), cómo último dentro de canal vamos a empezar un Loop infinito para todas las luces de tráfico.

Como último dependiendo de la información se va a mover a el canal “WayPoints” dónde se va a tener acceso al script o código a través de la variable “api” y de acuerdo con el ip seleccionado previamente tomar sus respectivas localizaciones. Para después moverse a la acción de asignar a cada localización un carro, posteriormente moverse a la acción voltear a esa dirección y finalmente terminar en la acción de moverse hacia esa dirección.


Unity - Waypoints
Unity -SemaphoreScript
Unity - LlamarApi
Unity - Singleton
Instantiate the Script "LlamarApi" into a variable named "api"
Instantiate the Script "LlamarApi" into a variable named "api"
Through a loop connect to the server through the WebRequest and constantly retrieve the uploaded data.
Inherits the Singleton class to operate
Access the Script through the api variable and according to their pre-selected id's grab their respective locations
Save the information in "guardarVariables" with a list of agents
Assign each car to a location
Get the respective amount of time to instantiate the traffic light (and as a secondary option receive the bool of each traffic light)
Turns into that direction
Starts an Infinite Loop for both Traffic Lights
And moves into that direction