Campus Monterrey

Propuesta Formal del Reto

Materia
Modelación de sistemas multiagentes con gráficas computacionales (Gpo 4)

Profesores
Edgar Covantes Osuna
Jorge Cruz Duarte

Integrantes

Facundo Vecchi A01283666
Nicolas Cardenas A01114959
Luis Omar Leyva A01570367
David Josué Marcial Quero A00828702

Septiembre 3, 2021

Indice

Link de github: https://github.com/josue-quero/Jupyter_Activities/blob/main/M3.ipynb

Historial de Versiones:

Version	Cambios	Responsables
1.0	Creación de documento con todos sus apartados e información relevante a la explicación del reto	Todo el equipo
1.1	Modificación del modelo y de la propuesta	Todo el equipo

Introduccion

“La movilidad urbana, se define como la habilidad de transportarse de un lugar a otro y es fundamental para el desarrollo económico y social y la calidad de vida de los habitantes de una ciudad. Desde hace un tiempo, asociar la movilidad con el uso del automóvil ha sido un signo distintivo de progreso. Sin embargo, esta asociación ya no es posible hoy. El crecimiento y uso indiscriminado del automóvil —que fomenta políticas públicas erróneamente asociadas con la movilidad sostenible— genera efectos negativos enormes en los niveles económico, ambiental y social en México.”

Nosotros debemos proponer una solución a este problema mediante el modelaje de un sistema que se pueda optimizar y que tenga en cuenta los diferentes agentes involucrados para poder mejorar la movilidad urbana en todos los aspectos.

Descripción del reto

El reto consiste en proponer una solución al problema de la movilidad urbana en México. En México el uso del automóvil ha incrementado significativamente en las últimas décadas y de igual manera se ha incrementado la contaminación y efectos negativos del uso de estos. Se debe de idear una solución la cual teniendo en cuenta los diferentes agentes involucrados, llegue a mejorar la movilidad urbana en todo sus aspectos.

Lineamientos

Para poder darnos una mejor idea de como modelar este sistema debemos tener primero en cuenta los diferentes lineamientos que se deben respetar para poder hacer el modelo lo más cercano a la realidad para que nuestras simulaciones puedan ser más precisas. Nuestro modelo debe poder simular una intersección controlada por señales de semáforos inteligentes, es decir,

mientras que no haya un vehículo cercano, el semáforo estará en luz amarilla. Al momento de que cualquier vehículo se acerque a la intersección, enviará un mensaje con el tiempo estimado de arribo y el semáforo le dará luz verde al semáforo más cercano y establecerá un programa de luces a partir de este punto para el resto de los vehículos, es decir, cuando un carro se acerque, el semáforo le dará luz verde a ese carro y le dará luz roja a los demás y le dará prioridad, en cuanto a la luz verde, al carro que haya llegado después del primero.

Para poder saber en qué orden llegaron los carros podemos implementar diferentes soluciones las cuales no darán diferentes resultados. Después de un profundo análisis concluimos que no podemos tener toda la información de GPS en el carro y pasarsela a un semáforo mediante alguna señal de media-larga distancia ya que no muchos carros cuentan con esas funciones y queremos implementar un sistema que se pueda aplicar donde sea y para todos. Otras posible solución era poner sensores en las calles ya sea de luz o peso pero esto tampoco se nos hizo muy práctico ya que tuvimos que dar mucho mantenimiento para que esté funcionando correctamente. Así que optamos por tener sensores *lidar* en los semáforos, estos sensores miden la distancia a la que está un objeto por lo cual si tenemos un sensor en cada semáforo apuntando a la dirección correcta podemos saber que está llegando un carro pero también qué tan rápido se está acercando y moverlo correctamente.

Para el funcionamiento de que muchos carros llegan alrededor del mismo tiempo podríamos implementar un tipo de “queue” que cuando el primer carro llega y se le pone verde y después va pasando otro carro por otra calle un poco después, que lo ponga en la “queue” el cual es como una lista de espera para indicar que el va después.

Modelado de Agentes

Identificación de los agentes involucrados:

- Vehículos
- Semáforos
- Banqueta
- Calles

Rol y funciones

Vehículos: Los atributos incluyen su velocidad actual, la cual es calculada (al menos en esta modelación) en los sensores del semáforo, la cual debe tener muy poco margen de error. También es relevante el modelo del carro para saber más o menos su tamaño, pero esto igualmente se calcula con los sensores del semáforo.

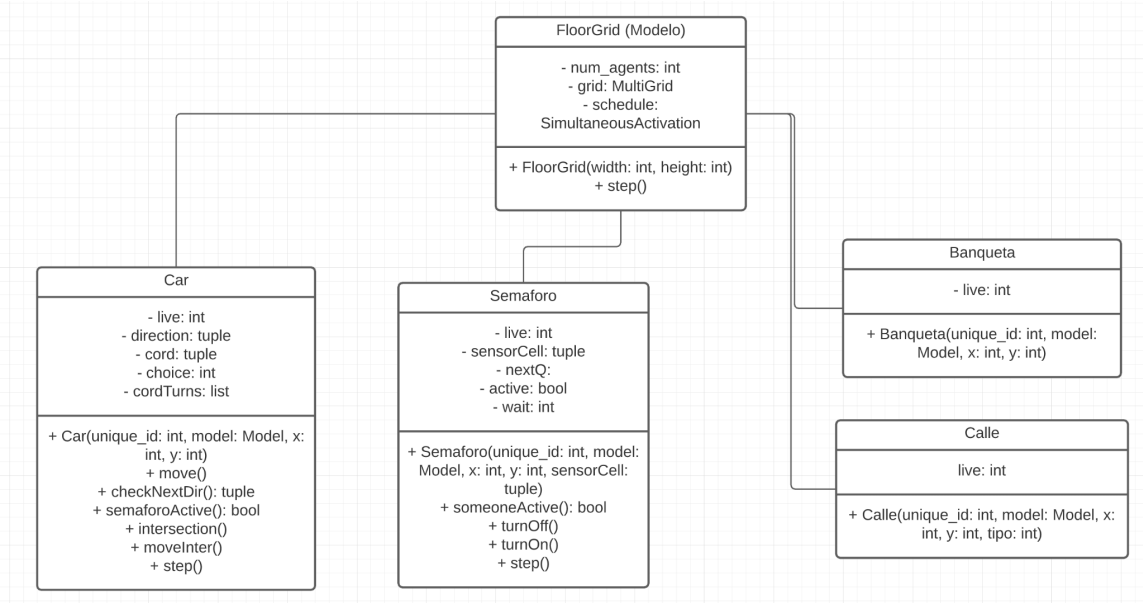
Semáforos: El agente correspondiente a un semáforo es uno de los más importantes, ya que es el que está encargado de optimizar y analizar toda la información que le está llegando desde

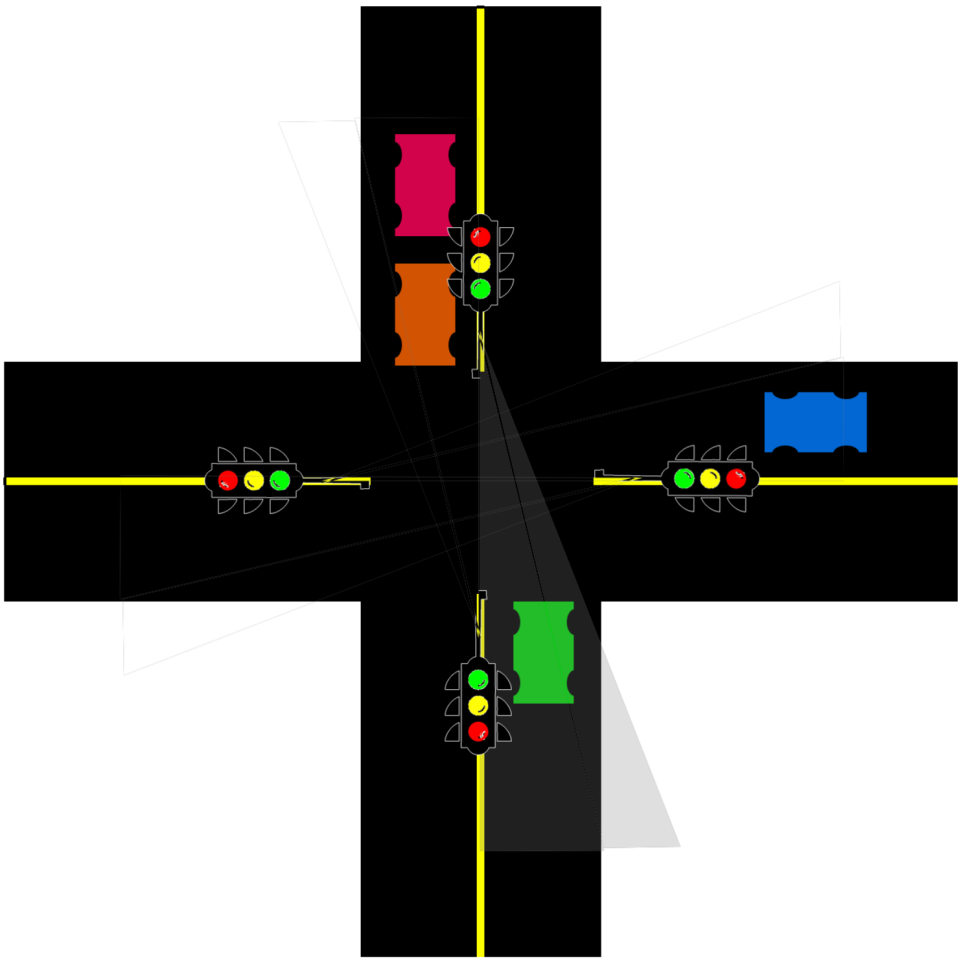
sensores sobre la posición y velocidad del carro. Un solo semáforo se comunica con los demás para coordinarse y darle paso a un solo carril a la vez, o dos en el caso de vueltas continuas. Asimismo, cambia su estado respecto a la información que le esté llegando. Finalmente, tiene como atributo, la calle en la que se encuentra y hacía cuál calle su luz verde permite llegar.

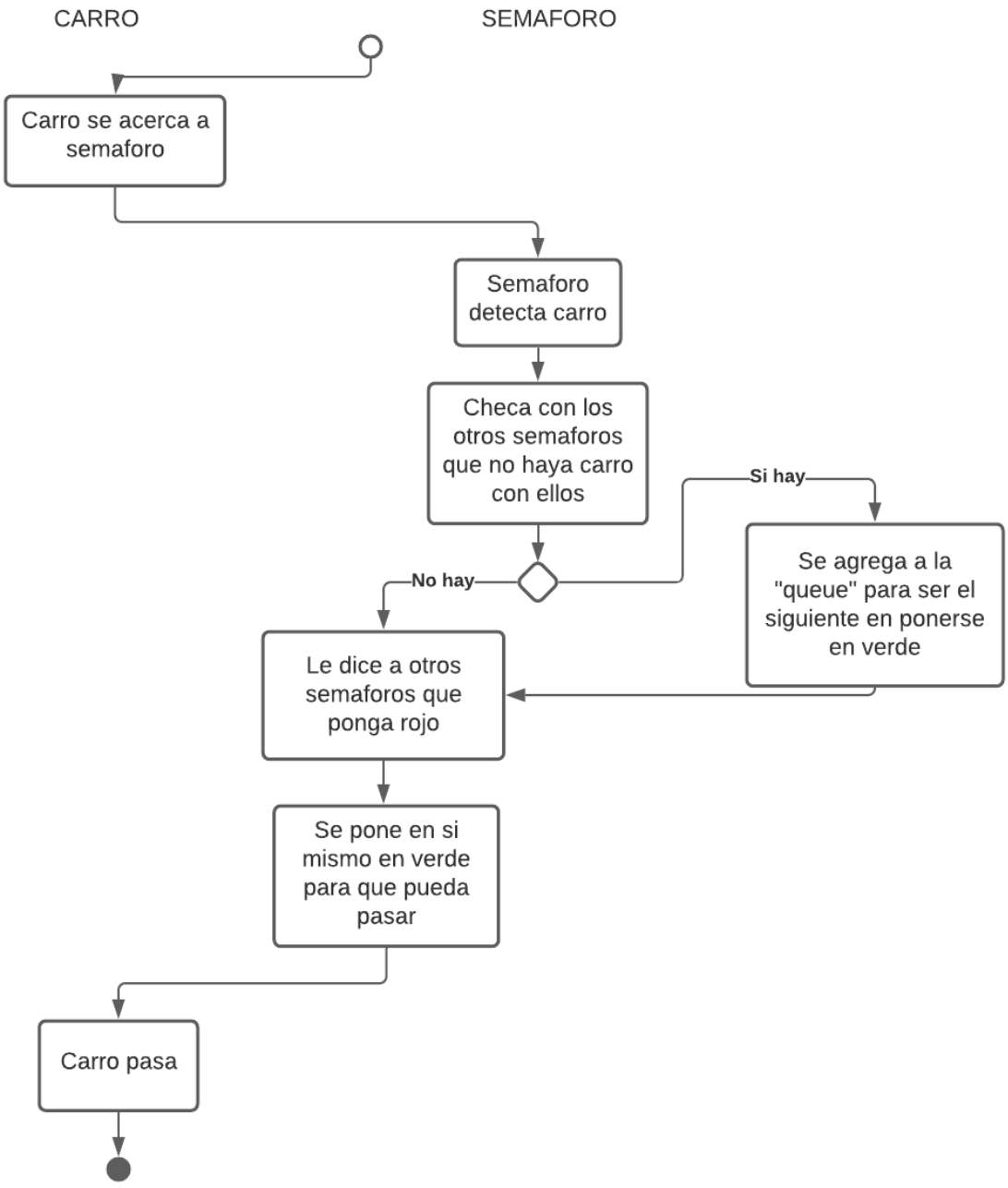
Calles: Para tener más precisión en el sistema, se tienen calles que le permiten al vehículo tener conciencia de las áreas por las que está transitando. Se puede tener una calle normal, una por la que no es posible moverse y otra por donde se tiene que parar al llegar al semáforo.

Banqueta: La banqueta sirve para delimitar el movimiento del vehículo, por lo cual un determinado vehículo no podrá salir de estos límites. Adicionalmente, sirve también para la visualización de la simulación, ya que permite una distinción más clara de la intersección y sus elementos.

Diagramas


FloorGrid (Modelo)
- num_agents: int
- grid: MultiGrid
- schedule: SimultaneousActivation
+ FloorGrid(width: int, height: int)
+ step()
Car
- live: int
- direction: tuple
- cord: tuple
- choice: int
- cordTurns: list
+ Car(unique_id: int, model: Model, x: int, y: int)
+ move()
+ checkNextDir(): tuple
+ semaforoActive(): bool
+ intersection()
+ moveInter()
+ step()
Semaforo
- live: int
- sensorCell: tuple
- nextQ:
- active: bool
- wait: int
+ Semaforo(unique_id: int, model: Model, x: int, y: int, sensorCell: tuple)
+ someoneActive(): bool
+ turnOff()
+ turnOn()
+ step()
Banqueta
- live: int
+ Banqueta(unique_id: int, model: Model, x: int, y: int)
Calle
live: int
+ Calle(unique_id: int, model: Model, x: int, y: int, tipo: int)





Ambiente

En nuestro tipo de intersección se podrían tener dos luces verdes prendidas al mismo tiempo pero por simplicidad solo tendremos uno verde al mismo tiempo, ya que si quisiéramos implementar

dos al mismo tiempo tuviéramos que tener los semáforos con flecha verde al igual que la luz verde.

Cuando un carro se acerca al semáforo, el semáforo checa si no hay ningún otro semáforo prendido, si no hay carros el se activa poniéndose a la *queue.* El semáforo se activa por cierto tiempo de segundos, en nuestro caso 7 pasos de la simulación, pero si sigue detectando carro en su celda (significando que hay mas de un carro), aumenta su duracion para que puedan pasar todos los carros lo cual es mas eficiente que andar cambiando para solo un carro y que se vayan exactamente en el orden que llegaron; ya que se pierde mucho tiempo cambiando de color porque tenemos que verificar que todos esten en rojo antes de darle luz verde a otro semaforo. Seria impractico que se cambiara el semaforo con cada carro ya que debe haber un tiempo de amarillo en el cambio de semaforo para que no esten circulando en el medio carros cuando hay cambio de sentido y es por esto que pasamos a todos de un carril bajo cierto tiempo y despues se hace cambio al otro carril y si no hay carros en otros carril podemos cambiar el estado al previo.

Con la manera que planteamos la solución, nuestros carros y conductores toman mínima parte en la manera que optimizamos la intersección ya que el semáforo está encargado de recolectar y analizar la información. Esto es beneficioso ya que nosotros podemos estar dándole mantenimiento a los semáforos cuando a los vehículos no podríamos estar monitoreando sus GPS o transmisores de señal.

La comunicación entre estos agentes sería mínima ya que el semáforo mismo es el que recauda la información del carro y su velocidad y posicionamiento. La semántica del acto de habla funcionaria algo así: performativo = “me acerco al semáforo” acto de habla = “¡Puede pasar!”. La comunicación que tomaría lugar sería entre semáforo y semáforo ya que hay múltiples semáforos en cada intersección y tienen que trabajar en conjunto ya que no pueden dar luz verde más de uno a la vez ni tener todos luz roja al mismo tiempo.

<u>Ejemplo de diálogo KQML / KIF:</u>
Semáforo 1 → Semáforo 2 : (ask-if(tu luz verde)))
Semáforo 2 → Semáforo 1 :(reply true)
Semáforo 1 → : (queue For Green)

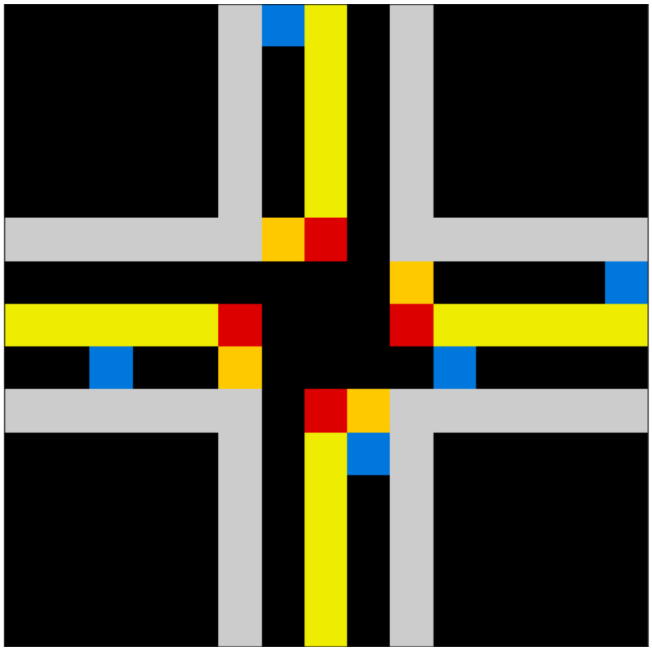
Para que nuestros semáforos puedan funcionar correctamente debemos tener un sistema de comunicación entre ellos de “Informar” y “Solicitar” cada vez que hay nueva información, se informa que tiene carro a los demás semáforos y se solicita cambiar el estado; es importante solicitar ya que se checa con los otros semáforos a ver quien está en *queue* para evitar los accidentes y respetar los turnos.

Como ya lo sabemos, una intersección busca el bien común de no generar accidentes, es decir, que el semáforo ayude a guiar a los conductores a pasar por dicha intersección de manera organizada y justa. Esto lo hace un ambiente cooperativo entre agentes, entre el semáforo y conductores pero también por conductores entre sí. Significa que son agentes benevolentes. En este sistema los semáforos comparten las tareas y los conductores/vehículos comparten los resultados: un cruce de intersección sin accidentes.

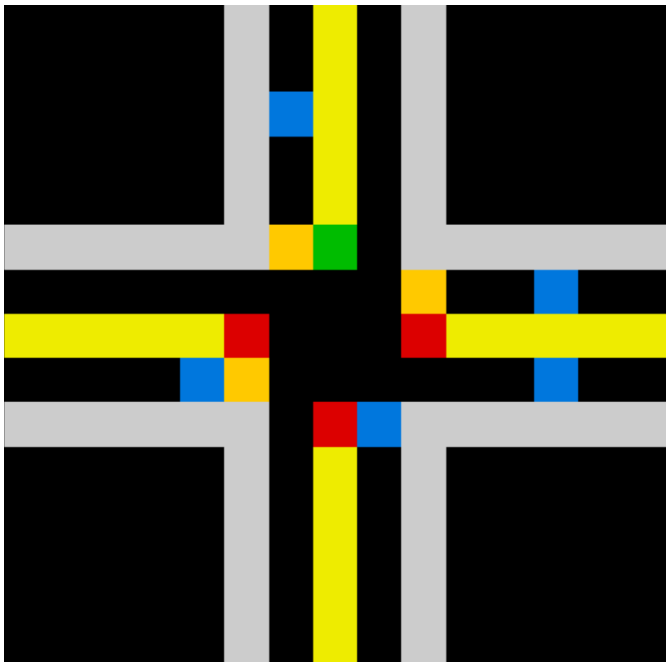
Simulacion

Indice de colores para la simulación

- Banqueta : Blanco
- Naranja : STOP sign
- Rojo : Semáforo en rojo
- Verde : Semáforo en verde
- Amarillo : Línea de distinccion de carril
- Azul : Carro
- Negro : Calle

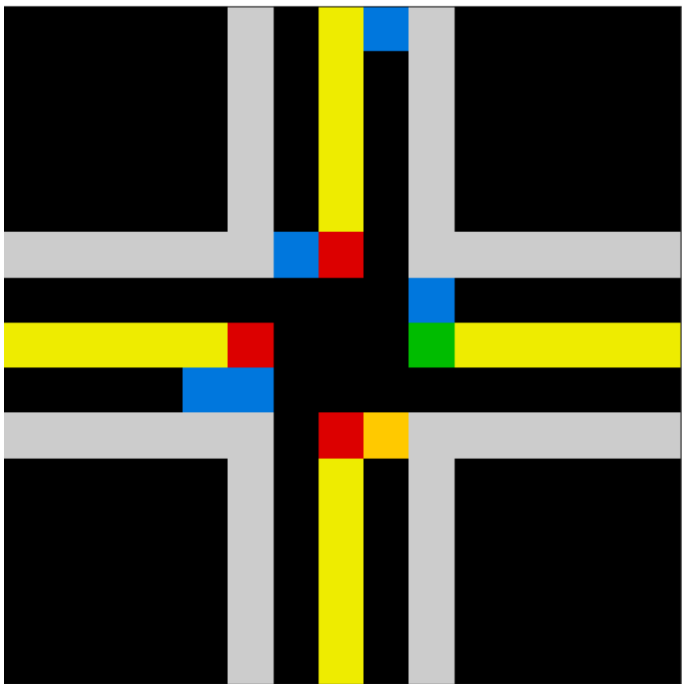


Posición de arranque para nuestra simulación

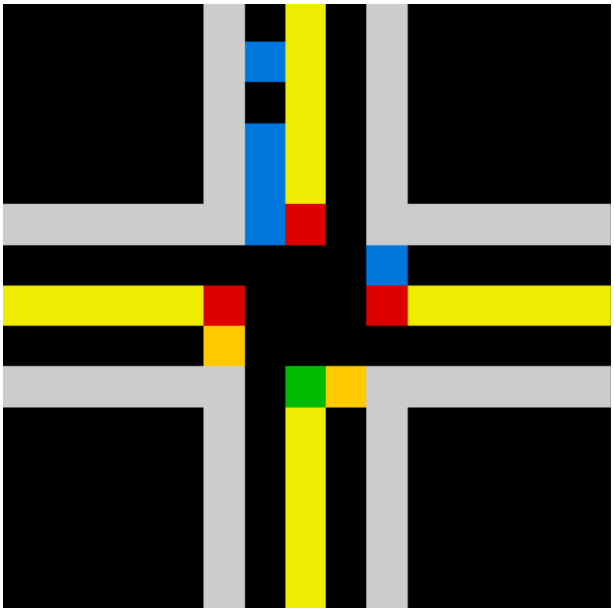


Un carro de acerca a un stop-sign y el semáforo prende verde

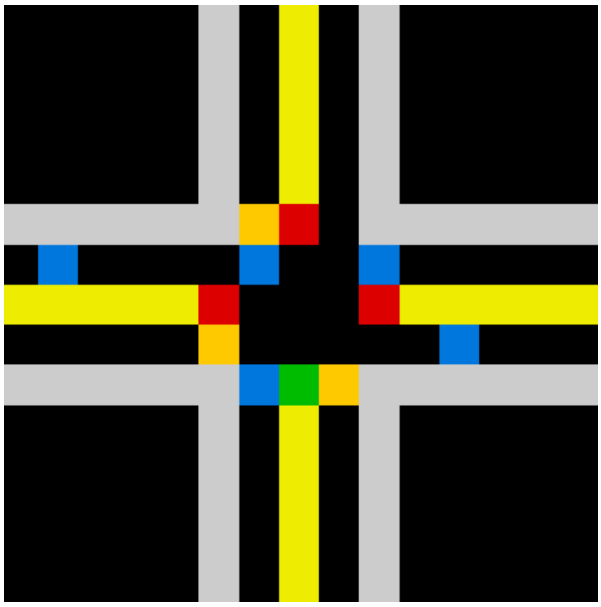
A partir de este punto el carro puede hacer 4 opciones: ir recto, girar a la derecha o girar a la izquierda, o dar vuelta en sentido de U. Cada vez que el carro pasa de un carril a otro a través del espacio toroidal es cuando escoge qué es lo que va a hacer la próxima vez que pase por un STOP sign.



Ya que pasa el primer carro, se apaga el semáforo activo y prende el semáforo que sigue (en nuestro caso sería el de la derecha ya que los del carril de la izquierda fueron los que llegaron después del primero)



En un caso como este donde tenemos 4 carros en un carril buscando moverse en la intersección, el programa o más bien el semáforo activo sensa que siguen pasando carros por el sensor por lo cual extiende el tiempo por el cual está activo.



Los cuatros carros toman caminos diferentes, cabe recalcar que toman diferentes tiempos en cruzar la intersección dependiendo de qué tipo de vuelta quieren dar, al igual que la vida real.

Link Collab:
https://colab.research.google.com/drive/1oSBkaU7zc08Bacqq4jCwsIfqityCepI-?usp=sharing

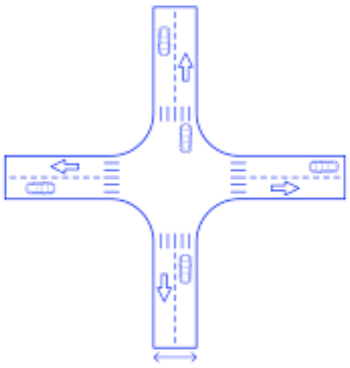
Graficas

(LO DE UNITY AQUI)

Limitaciones y Mejoras

Como ya se mencionó, en una simulación ya más avanzada donde todos los carros cuentan con sensores y puedan transmitir esa información, solo en ese entonces, podríamos tener un sistema donde la calle tenga sensores complementarios y los carros están mandando señales a los semáforos; en este sistema futurista tendríamos que poner en pie un sistema de comunicación eficiente pero por ahora con los datos que junta el semáforo son suficientes, también al momento de simular gráficamente podríamos poner los semáforos peatonales para indicar cuando pueden moverse los peatones pero en nuestro modelos no tendrán lugar ya que va más allá de lo que vamos a analizar. En un futuro también podríamos tener una sola máquina que registre todos los viajes de todos los conductores mediante Google Maps o algo parecido y organice la mejor manera de moverlos alrededor, y sincronizar los carros que para ese entonces serían completamente automáticos y ni siquiera tener la necesidad de implementar semáforos. El semáforo no puede saber en qué dirección se va a mover el carro al menos que se implemente algún sistema de reconocimiento de imagen con cámaras el cual lea la direccional del carro. Asimismo, podríamos agregarle a la simulación una luz naranja o amarilla para avisar que ya se va a acabar el semáforo, la implementación sería fácil pero optamos por no hacerlo ya que no brinda beneficio al momento de simularlo ya que no interactuara con ningún agente.

Cabe mencionar que nuestro modelo a programar sería basado en uno parecido al que se muestra en la parte inferior:



Cruce T Controlado

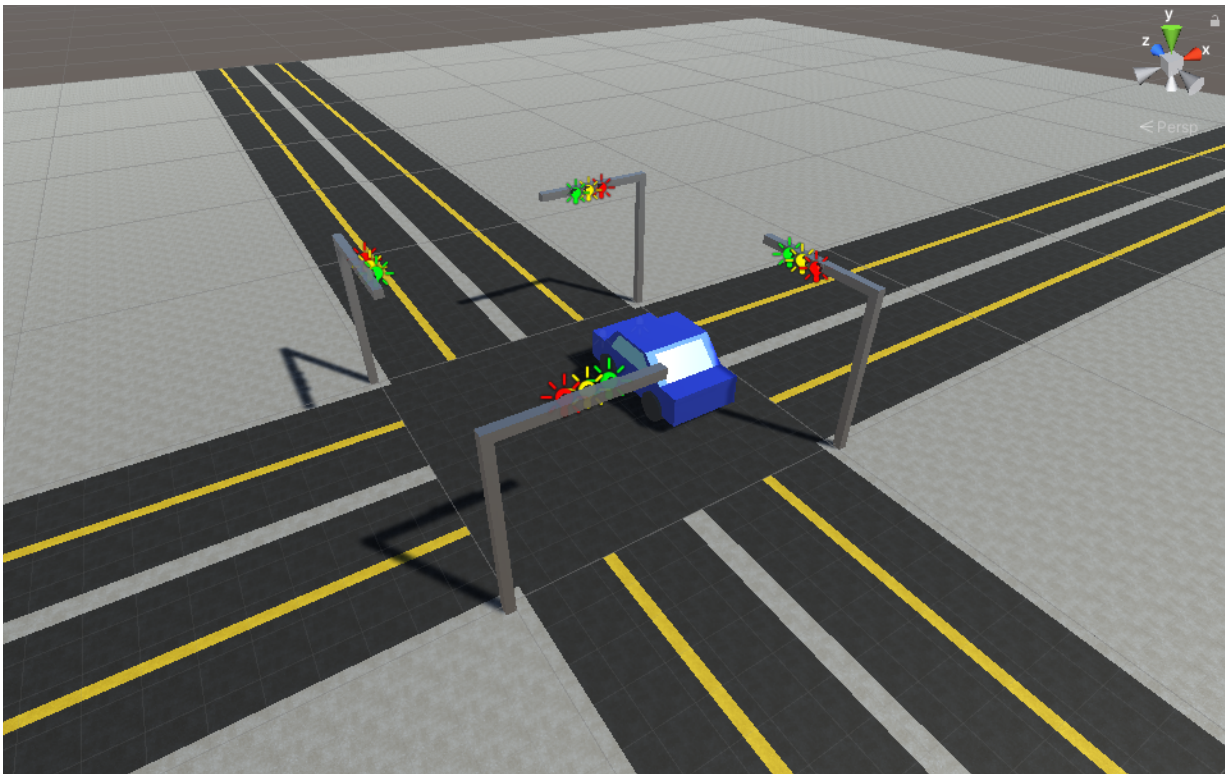
El cual es como un caso base pero para diferentes intersección es posible que haya modificaciones en el código, ya que en intersecciones de tres diferentes sentidos o múltiples carriles se podría optimizar mejor ya que se pueden tener dos verdes al mismo tiempo pero nuestro programa debe ser funcional en casi cualquier intersección que comparta las

características de la nuestra pero existen muchas variaciones y tipos (controlado, no controlado, glorieta, *Y-fork,* entre otras) por lo tanto nuestro programa no puede considerar todas. Nuestra intersección asume que no hay cruces peatonales. En nuestra interacción que simulamos no puedes dar la vuelta a la derecha sin semáforo ya que cuando lo simulamos así había muchos "choques".

Producto Final

Al conectar nuestra simulación con los modelos 3D en Unity, llegamos a la solución que buscábamos la cual contiene todo lo que requiere para solucionar nuestra problemática. La conexión de nuestra representación gráfica, Unity, y nuestra simulación, Mesa/Python fue hecha mediante un servidor local el cual mandaba objetos de tipo JSON a Unity para que pudiera actualizar la posición de los carros y las luces de los semáforos de acuerdo a cómo se movían en la simulación. Este paso fue el más complicado durante todo el desarrollo del reto ya que consiste en conectar todas las partes realizadas anteriormente y está conexión fue complicada de programar y adaptar para que luciera correctamente y pudiéramos visualizar la simulación de una manera precisa que pudiera servir para solucionar la problemática.

El producto final luce de la siguiente manera:



Conclusion

Modelar un sistema como este para resolver un problema tan relevante como este, tiene que tomar en cuenta muchos factores y tener los agentes correctamente identificados para también saber cómo se comunican entre ellos . Y sólo tras este análisis podremos llevarlo a una etapa más cercana a la realidad: la simulación, en un futuro nuestro equipo modelará el sistema que se delineó hoy en un motor gráfico para poder visualizar las interacciones entre estos agentes y poder optimizarlo de manera más eficiente.

Plan de trabajo

Actividades:

- ☑ ~~P1. Arranque de Proyecto~~
    - Fecha límite: 13/08/21
    - Responsables: Todos los miembros del equipo.
    - Esfuerzo:
- ☑ ~~P2. Revision de avance~~
    - Fecha límite: 03/09/21
    - Responsables: Todos los miembros del equipo.
    - Entregables: Modelado del reto y avance del código en Python.
    - Tiempo estimado: 4 horas.
    - Tiempo real: más de 4 horas.
- ☐ P3. Entrega Final
    - Fecha límite: 10/09/21
    - Responsables: Todos los miembros del equipo.
    - Entregables: Modelado del reto, código en Python, simulación gráfica en Unity e instrucciones de instalación del proyecto.
    - Tiempo estimado: 3 días.
    - Tiempo real: 3 días.

Aprendizaje adquirido

P1
Como equipo aprendimos sobre las fortalezas y áreas de oportunidad de cada uno de los miembros del equipo lo cual nos ayudará a delegar las responsabilidades y actividades del proyecto.

P2

Al haber hecho diferentes prácticas como la actividad integradora, contamos con más experiencia tanto como para modelar los agentes pero también para programarlos en un ambiente como este. También obtuvimos experiencia con las actividades en Unity la cual nos servirá para el siguiente avance del proyecto.

P3

Ya con los modelos 3D de Unity creados para el proyecto y con la simulación de Python completa, aprendimos como hacer la conexión entre estas dos partes la cuales hemos estado trabajando durante una gran cantidad de tiempo. En está fase juntamos todo lo realizado al igual que los documentos que se nos pidieron para realizar la entrega final. Ya contamos con la experiencia de todas las herramientas con las cuales trabajamos (Mesa,Python,Unity) pero creemos que el aprendizaje principal durante esta fase fue como realizar la conexión de la simulación a la representación gráfica mediante un servidor local.

Referencias

Presentación Modelación de sistemas multiagentes con gráficas computacionales (TC2008B) - Comunicación entre agentes - Dr. Edgar Covantes Osuna

(2021). Retrieved 18 August 2021, from https://experiencia21.tec.mx/courses/182839/pages/reto

How to Keep Safe at 8 Popular Types of Road Intersections. (2021). Retrieved 18 August 2021, from https://driving-tests.org/beginner-drivers/crossing-paths-keeping-yourself-and-others-safe-at-intersections/

(2021). Retrieved 18 August 2021, from https://www.ontario.ca/document/official-mto-drivers-handbook/driving-through-intersections
Transport for NSW - Intersections (2021). Retrieved 18 August 2021, from https://roads-waterways.transport.nsw.gov.au/roads/safety-rules/stopping-turning/intersections.html