Tecnológico de Monterrey

Instituto Tecnológico y de Estudios Superiores de Monterrey

TC2008B.401 - Modelación de sistemas Multiagentes con gráficas computacionales(Grupo 601)

Evidencia 2: Actividad Integradora

Raymundo Iván Díaz Alejandre | A01735644

Daniela Lozada Bracamontes | A01736594

Erwin Porras Guerra | A01734881

Gerardo Deustúa Hernández | A01736455

Profesores: Luciano García Bañuelos e Iván Olmos Pineda

Domingo 15 de octubre del 2023

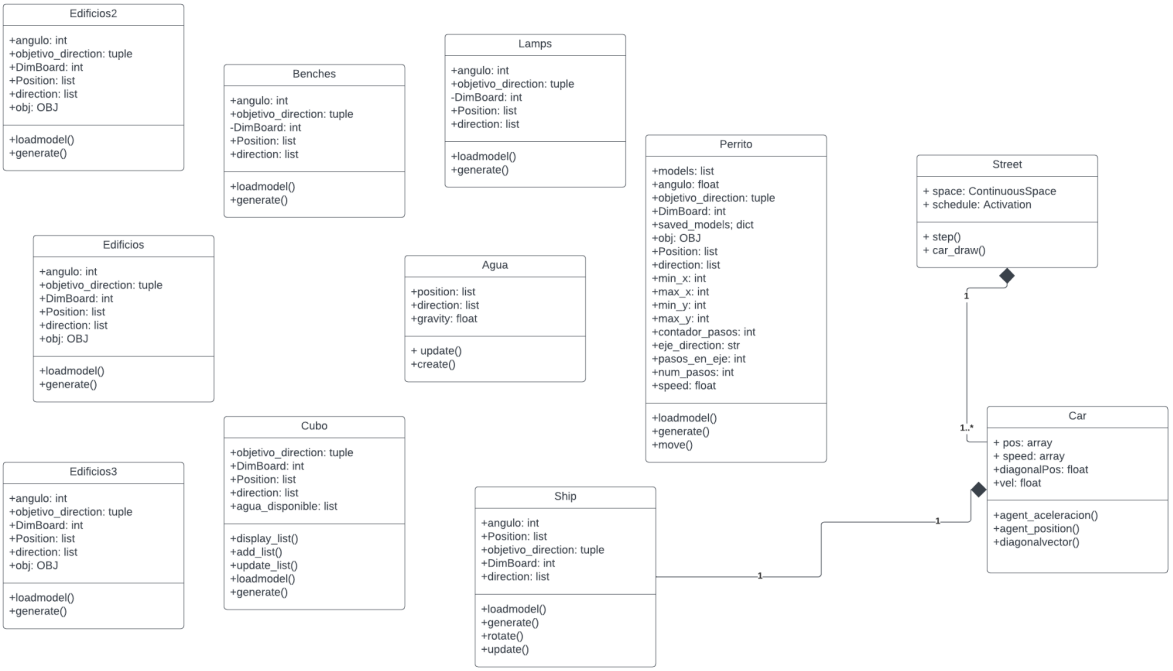
Descripción del reto a desarrollar:

La movilidad urbana en México es un desafío creciente que afecta a millones de personas en todo el país. Con una población en constante crecimiento, ciudades congestionadas y un sistema de transporte público a menudo inadecuado, el caos en las vías de tránsito se ha convertido en una realidad cotidiana para muchos mexicanos. El problema no solo implica la pérdida de tiempo y recursos, sino que también tiene un impacto significativo en la calidad del aire, la salud pública y el medio ambiente.

La complejidad de la movilidad urbana en México requiere soluciones innovadoras, y una de las respuestas prometedoras se encuentra en los sistemas de simulación de multiagentes. Estos sistemas permiten modelar y simular el comportamiento de múltiples agentes, como vehículos, peatones y servicios de transporte, en un entorno urbano virtual. Al utilizar datos en tiempo real, estas simulaciones pueden proporcionar información valiosa sobre patrones de tráfico, congestión y posibles mejoras en la infraestructura vial.

Identificación de agentes involucrados:

Diagrama de clases.



En el diagrama de clases se pueden observar las clases presentes en la elaboración del reto, donde podemos observar tanto como sus atributos y métodos. Se observa que solo 3 clases tienen relación entre ellas porque las otras clases distintas a ellas son las usadas para cargar los modelo 3D.

Las clases con relación son:

Street-Car.

Se observa una relación tipo composición ya que Street no puede existir si no existe la clase Car, ya que Street es como el espacio lógico de la simulación, no el plano con los distintos objetos de la ciudad como se podría llegar a pensar.

Street puede tener de una a muchas instancias de la clase Car, mientras que car solo 1 de la clase Street.

Car-Ship.

Se observa una relación tipo composición por qué Car necesita a Ship para existir, ya que la clase Car es la que se utiliza para dar los movimientos del vehículo mientras Ship es la carga y generador del modelo 3D de este. La relación es uno a uno.

Análisis de la solución desarrollada:

Se seleccionó simular una rotonda con un sistema multiagentes ya que en la ciudad existen varios lugares en donde se crea una redundancia al integrar semáforos en la rotonda, creando esencialmente lo que es una intersección glorificada. El propósito principal con este proyecto es crear una simulación de cómo deberían funcionar las rotondas de manera propia, donde no hay semáforos y los carros esperan hasta que puedan entrar a la rotonda.

La variable principal que se tomó en cuenta fue si el carro tenía una posición libre en la cual podría entrar a circular la rotonda, también se tomó en cuenta la velocidad del auto la cual aumenta o disminuye dependiendo de su cercanía a otro carro.

Estas variables eran fundamentales para determinar el comportamiento de los agentes, ya que para entrar a circular a la rotonda el carro necesitaba antes determinar si tenía un espacio libre para hacerlo, la velocidad también entra en

juego ya que esta cambia dependiendo de los demás agentes. Sin embargo una variable que no se tomó en cuenta fue una variable de destino, idealmente los carros deberían seleccionar un destino antes de entrar a la rotonda, pero en este caso se decidió que esta fuera una decisión al azar cada vez que se encuentra con la posibilidad de salir de la rotonda para hacer más fácil este problema.

El ambiente gráfico fue creado con la librería de OpenGL, la cual es una opción muy popular para crear una gran variedad de diseños gráficos para una cantidad inmensa de juegos y programas, esta librería es fácil de implementar en python y ofrece una gran variedad de flexibilidad al momento de la implementación, en cuanto a los modelos seleccionados y el diseño general del ambiente se escogieron arbitrariamente a los gustos de los integrantes.

Al final en la solución presentada, algunas de las ventajas que obtuvimos fueron que pudimos crear el comportamiento ideal de una rotonda, en donde los carros no requieren de ningún semáforo para ser dirigidos. Otra ventaja fue que tampoco se genera ningún tipo de colisión. Sin embargo sí existen algunas cuantas desventajas en la simulación presentada, ya que el comportamiento de los carros no es exactamente preciso ya que no se tomó en cuenta factores que en la vida real afectan a los conductores, como imprudencias o negligencias. Este modelo se basa en un comportamiento ideal donde siempre se da el paso, pero esto frecuentemente no es la realidad.

Para reducir estas desventajas y obtener una simulación más exacta se podrían tomar en cuenta estos factores de imprudencias y negligencias y también agregar otros factores como peatones.

Una reflexión sobre tu proceso de aprendizaje. Para ello, revisa el documento original que contiene tus expectativas al inicio del bloque y compáralo con las experiencias que viviste a lo largo de estas semanas.

En general el proceso de crear una simulación semi inteligente con un sistema de agentes fue algo bastante enriquecedor. Se vuelve evidente que la simulación no solo es una herramienta para comprender y modelar el mundo que nos rodea, sino

también una oportunidad para reflexionar sobre nuestro propio proceso de aprendizaje y resolución de problemas. La creación de simulaciones de multiagentes es un proceso desafiante pero apasionante que nos permite adentrarnos en la complejidad de los sistemas y aprender a través de la experimentación y la reflexión continua

Reporte del proyecto.

Como anteriormente se mencionó, la programación del proyecto está dividida en dos partes frontend que nos permite crear la interfaz 3D de la simulación y el backend donde controlamos la lógica del proyecto.

Mesa

La parte lógica del proyecto es controlada directamente por mesa, en esta se puede encontrar todo el funcionamiento de los autos y como estos reaccionan al ambiente en el que se encuentra. Posteriormente estas indicaciones o funciones son representadas dentro de openGL en una animación 3D.

Comenzamos con el archivo traffic2.py en el que mediante el lenguaje de programación python y la biblioteca de mesa podemos generar una programación de multiagentes.

Principalmente se tiene la clase Car, dentro de esta se determina el funcionamiento, las reglas y las características de cada agente.

Los parámetros que se necesitan para su funcionamiento es su posición inicial(self,pos), su velocidad (self.vel) y su dirección(self.speed), declarados dentro de la función init.

```
class Car(Agent):

    #No importa la direcccion a la que empiecen, cuan
    #-----------------------------------------------
    def __init__(self, model: Model, pos, speed):
        super().__init__(model.next_id(), model)
        self.pos = pos
        self.speed = speed
        self.diagonalPos = 90
        self.vel = 1
```

Posteriormente dentro de la función step dentro de la clase Car, se encuentran las condiciones necesarias para cambiar la dirección del carro si se encuentra en un punto crítico. Esto es útil al momento de dar giros cuando el camino lo requiere, o a escoger entre dos posibles rutas.

```
    direccionAbajo = [0,0.5]
    direccionArriba = [ 0,-1]

    direccionDerecha = [0.5, 0]
    direccionIzquierda = [-1, 0]
    d_y = self.speed
```

```
#ESTOS SON DECISIONES PARA CONTROLAR LA NUEVA DIRECCION DEL CARRO

        #RUTA B
        randomB = random.choice([direccionAbajo, direccionDerecha])
        if (x == 5) and (y == 10):
            self.diagonalPos = 180 # 270 Grados hasta 6, 14
            # self.diagonalPos = np.array(self.diagonalvector(270))
        elif (x == 7) and (y == 10):
            self.diagonalPos = 270
        elif (x == 6) and (y == 14):
            self.diagonalPos = 315 # 315 Grados hasta 10, 18
            # self.diagonalPos = np.array(self.diagonalvector(315))
        elif (x == 7) and (y == 14):
            self.diagonalPos = 315
            self.speed = np.array(direccionAbajo)
        elif (x == 7) and (y == 16):
            self.speed = np.array(direccionDerecha)
        elif (x == 8) and (y == 16):
            self.speed = np.array(direccionAbajo)
        elif (x == 8) and (y == 17):
            self.speed = np.array(direccionDerecha)
```

Dado que la función step se actualiza constantemente, se puede controlar su futura posición y si esta es posible, para eso se creó una función que ayuda a verificar si la nueva posición está disponible o está ocupada por un agente. En el primer caso el agente podrá seguir avanzando, pero en el segundo caso se deberá detener hasta que esta nueva posición esté disponible.

```
#-----------------------------------------------------------------------
#NUEVAS POSICIONES
    new_pos = self.pos + np.array([1,1]) * self.speed
   # print("New_pos", new_pos)
    if not self.agent_position(new_pos[0], new_pos[1]):
        self.model.space.move_agent(self, new_pos)
#.....................................................................

def agent_position(self, x, y):
        agents = self.model.space.get_neighbors((x, y), include_center =True, radius=0)
        agentbool =any(isinstance(agent, Car) for agent in agents)
        # print("ENFRENTE", agentbool)
        return agentbool
```

Es posible controlar la velocidad a la que el agente va considerando si tiene vecinos cercanos o el camino se encuentra disponible para que avance. Esto se logró con una función similar a la anterior cambiando unos parámetros como el radio en el que considera vecinos y se irá actualizando con una nueva variable que interactúa con la velocidad actual.

```
    d,v = self.speed
    d= d * self.vel
    v= v * self.vel
    self.speed = [d,v]
#.....................................................................
#ACELERACION Y FRENO DE LOS COCHES CONSIDERANDO SUS VECINOS

    if self.agent_aceleracion(new_pos[0], new_pos[1]) == True:
        # print("desacelera porque hay coches")
        self.vel = self.vel - 0.05
    else:
        # print("acelera porque no hay coches")
        self.vel = self.vel + 0.001

def agent_aceleracion(self, x, y):
        agents = self.model.space.get_neighbors((x, y), include_center =False, radius=1)
        agentbool =any(isinstance(agent, Car) for agent in agents)
        # print("ACELERACION", agentbool)
        return agentbool
```

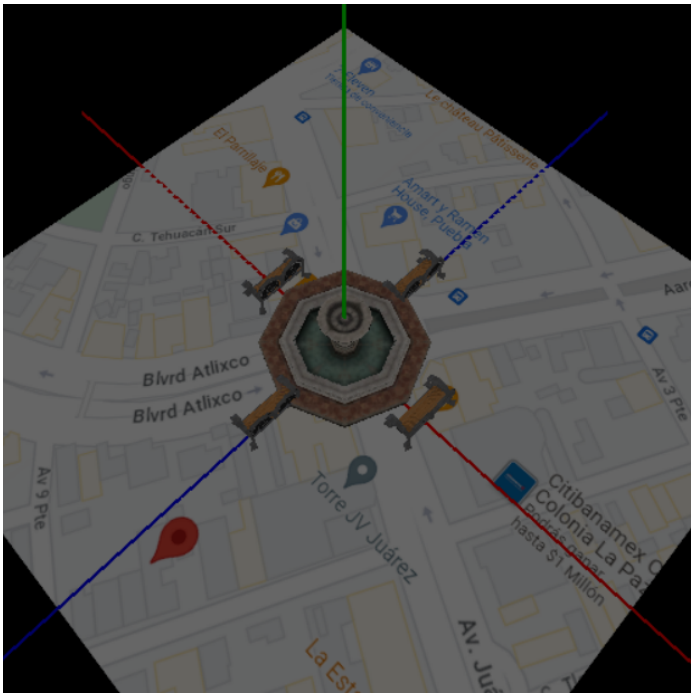
Finalmente se creó la clase Street que será llamada por el backend.py para interactuar con openGL. Dentro de esta se creó el tamaño del ambiente en continuousSpace que nos permitirá generar carros del otro lado del plano cuando excedan las dimensiones (Un ejemplo de ello es pacman). Existe una variable que permite controlar la cantidad de carros que se generan de cada lado y 4 fors para controlar la creación de carros de cada lado.

```
class Street(Model):
    def __init__(self):
        super().__init__()
        self.space = ContinuousSpace(25, 25, True)
        self.schedule = RandomActivation(self)

#CANTIDAD DE CARROS
        ncarros = 2
#Generación de carros lado A YAAA ESTA
        a = 10

        for px in np.random.choice(1, ncarros, replace=True):
            #               Posicion Inicial    direccion inicial
            car = Car(self, np.array([a, px]), np.array([0.0, 1.0]))
            self.space.place_agent(car, car.pos)
            self.schedule.add(car)
# # #Generación de carros lado B
        b = 14
```

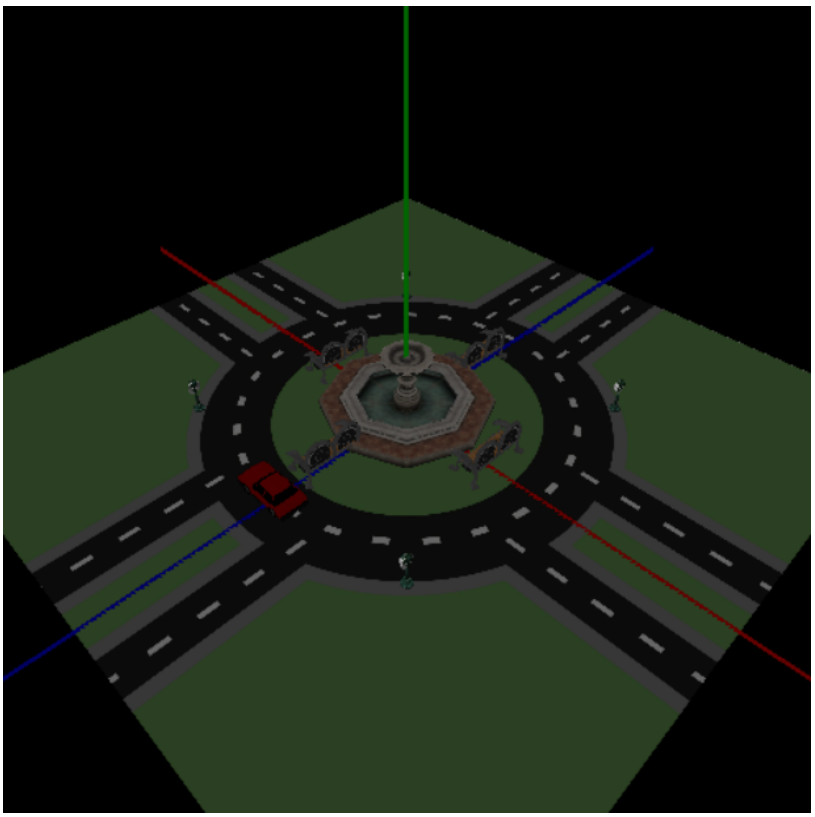
OpenGL

Para la implementación de OpenGL primero se realizó una simulación muy básica donde solo se dibujaba un plano gris y el agente que sería representado por un cubo multicolor, el fin de esta implementación fue más que nada para realizar la conexión básica entre Mesa y la simulación de OpenGL.

Para la segunda versión de la simulación se decidió trabajar la parte gráfica y la parte lógica por separado, por lo que en está versión del entorno gráfico se eliminó el dibujado del agente y se agregó que el plano de la simulación ya contará con una textura de la calle que sirvió como idea para la simulación final. A la par se agregaron objetos 3D como decoración a la escena como lo es una fuente y bancas que rodean está fuente.



La siguiente serie de modificaciones importantes realizadas en la simulación fue que se cambió el fondo del plano a una imagen que simula la carretera por donde circularán los coches y que está hecha a la medida para la parte lógica de mesa. También se agregaron más elementos decorativos así como una versión preliminar del modelo 3D del agente.



Por último, se agregaron edificios decorativos en las áreas verdes así como luces que son emitidas desde las farolas, y también una luz que ilumine el plano desde arriba, igual se arregló la dirección de algunos objetos. Por último, se cambió el diseño que tomarán los agentes en la simulación, y en este caso se cambiaron por naves de Star Wars.