Instituto Tecnológico y de Estudios Superiores de Monterrey

M5. Revision de avance 2

TC 2008 B570: Modelación de sistemas multiagentes con gráficas computacionales

Integrantes:

Gael Patricio Gaytan Botello **A00835334**

Gael Patricio Gaytan Botello Fortalezas
Soft: Dedicado, creativo, motivado

Fortalezas Hard: Desarrollo Web (HTML y CSS). Conocimientos de Python, C++ y Matlab.

Áreas de oportunidad: Mejorar conocimientos en estructura de datos, MySQL y Unity.

Expectativas del bloque: Durante este bloque espero aprender más sobre los sistemas multi-agentes. También espero mejorar mis habilidades con Unity. Estoy emocionado de participar en este bloque y trabajar junto con mi equipo en el reto.

Expectativas del equipo:

Estoy muy emocionado por formar parte de este bloque debido a que el contenido se ve muy interesante y retador. Espero mejorar nuestros conocimientos para poder solucionar el reto de la mejor forma posible. Soy alguien comprometido que siempre busca formas de mejorar, y daremos nuestro mejor esfuerzo a través de las diferentes etapas del reto.

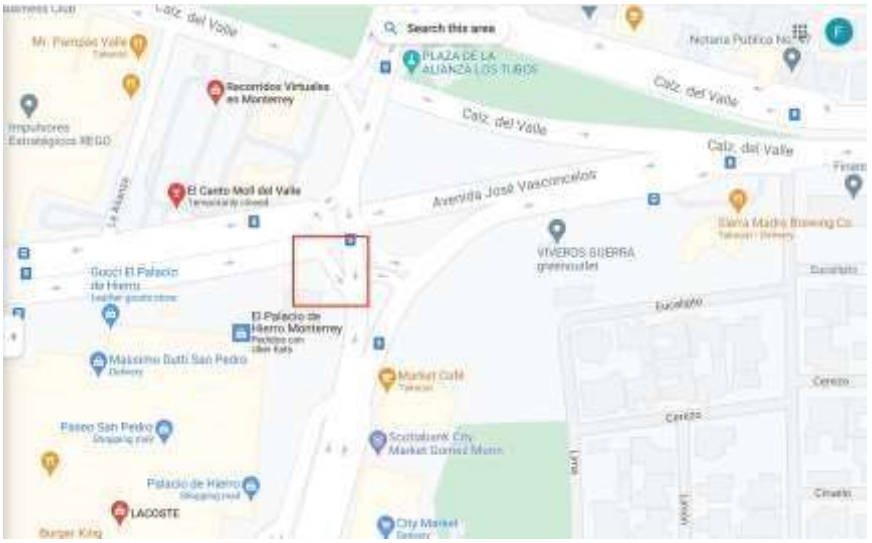
Creación de herramientas de trabajo colaborativo:

Se creó un repositorio en Github llamado Proyecto-Movilidad-Urbana y una carpeta compartida en Google Drive para colaborar en los archivos necesarios.

Propuesta formal del reto:

Descripción del Reto a Desarrollar

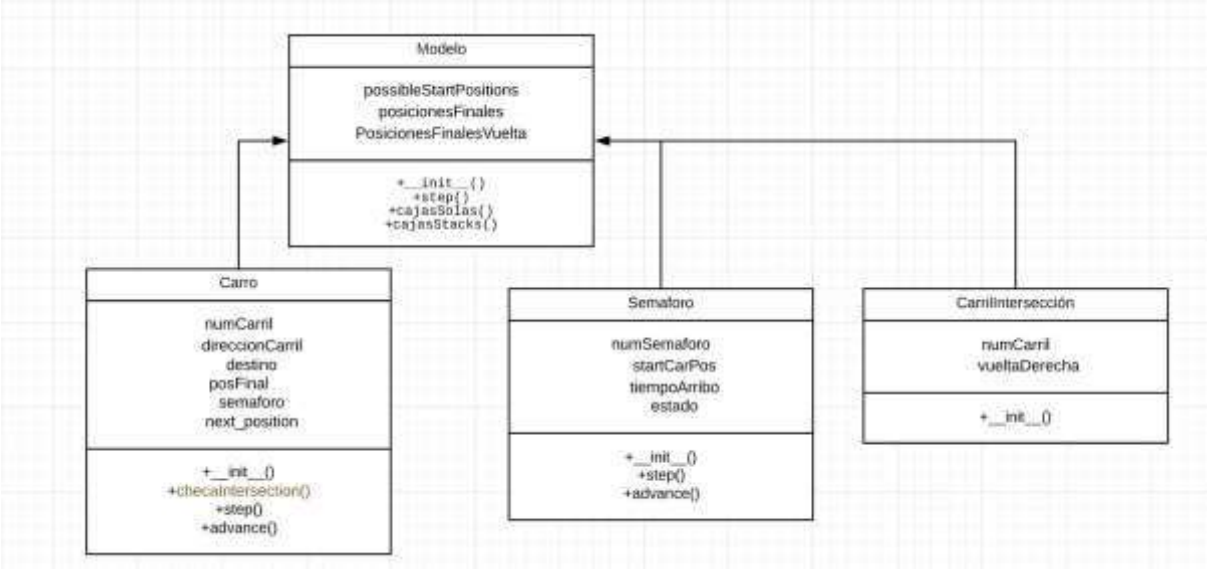
Con el objetivo de reducir la congestión vehicular y mejorar la movilidad urbana en México, desarrollaremos una sistema multi-agente que indique cuando haya muchos carros formados para cruzar una intersección. De esta forma, un semáforo podrá determinar el momento y duración de la luz verde. Así evitando posibles accidentes durante el cruce. Esta simulación gráfica será basada en la actual intersección que conecta a la avenida Vasconcelos con la avenida Gómez Morín, situada en el municipio de San Pedro Garza Garcia. Se seleccionó esta intersección porque actualmente genera mucho tráfico vehicular durante hora pico debido a accidentes durante el cruce.



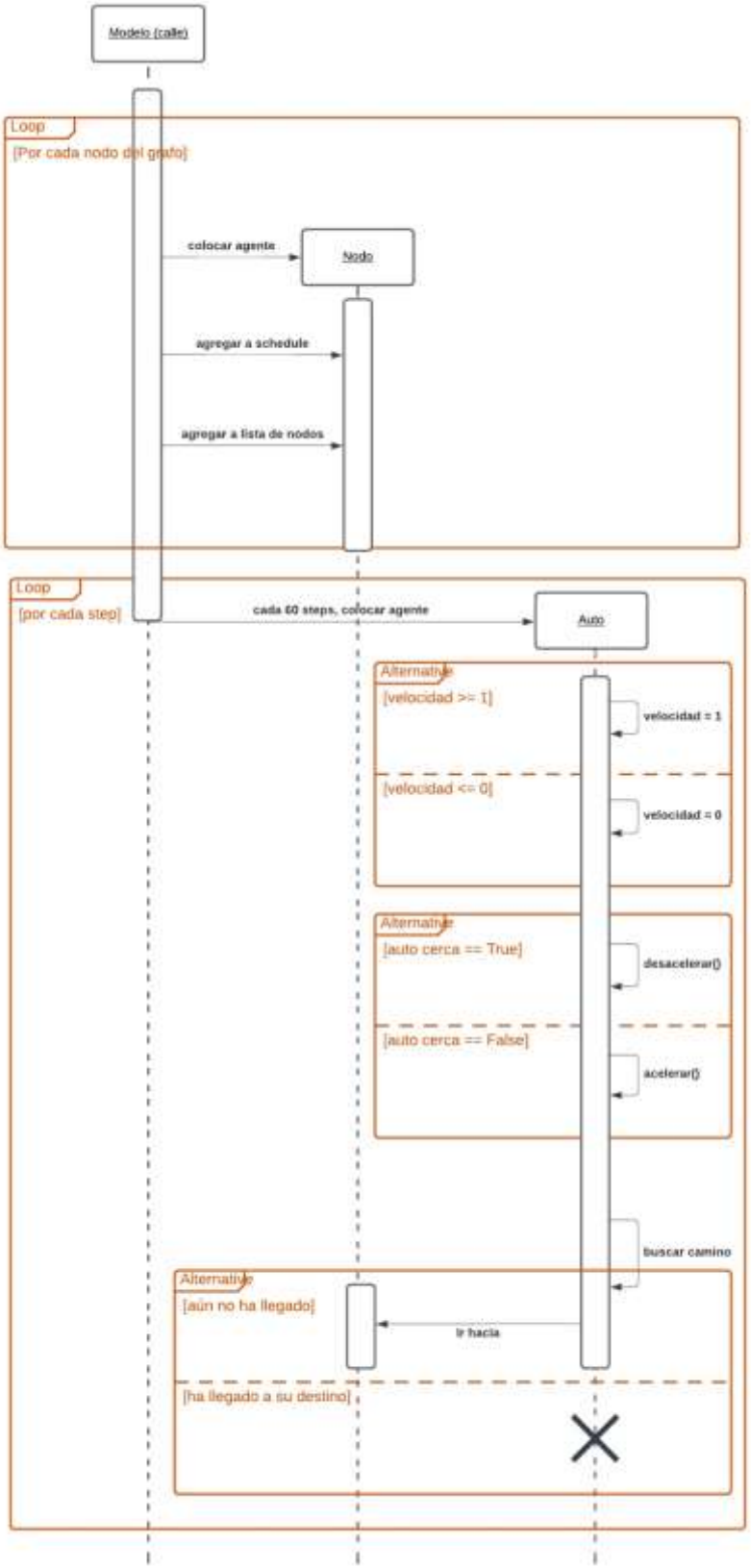
Agentes Involucrados

Agentes involucrados: automóvil y semáforo.

Diagramas de Clases UML para los Agentes Involucrados



Protocolos de interacción:



Implementación de agentes:

Explicación: el avance hasta ahora de la simulación de tráfico es el siguiente: a nivel de comportamiento de agentes y de la lógica del problema, ya existe un modelado de tráfico en la rotonda seleccionada, donde los autos saben cómo moverse por las calles, obedecen el sentido de circulación y, partiendo de una ubicación inicial aleatoria, se dirigen hacia un destino aleatorio siguiendo un recorrido trazado a partir del mapeo visible en la **imagen 3**. Es necesario trabajar en la detección de colisiones de los autos ya que puede provocar errores en la simulación y en dotar de una apariencia más natural los desplazamientos de los vehículos.

código

```
#importacion de librerias y modulos especificos

import numpy as np



from mesa import Agent, Model

from mesa.space import ContinuousSpace

from mesa.time import RandomActivation
from mesa.visualization.ModularVisualization import

ModularServer



#importacion de modulo continuo simple pre definido

from SimpleContinuousModule import SimpleCanvas



#definicion del agente auto
class Car(Agent):

    #constructor del agente auto

    def __init__(self, model: Model, pos, speed, inicial,

final):

        super().__init__(model.next_id(), model)
```

```
        #propiedades basicas de posicion y velocidad

        self.pos = pos
        self.speed = speed

        #variables de control para seguimiento de rutas

        self.contador = 0;

        self.contadorNodos = 1;

        #variables de control para algoritmo aStar

        self.openList = []

        self.closedList = []
        #seleccion de color aleatorio

        self.color = self.random.choice(["Brown", "Blue",
"Red", "Yellow", "Green"])

        #creacion de una ruta con aStar

        self.aStar(inicial, final)

        """

        print(self.color) for nodo
        in self.closedList:

            print(nodo.unique_id, nodo.pos)

        """



    #step del agente auto
```

```
    def step(self):

        #deteccion de auto delantero

        car_ahead = self.car_ahead()



        #control de aceleracion y desaceleracion

        new_speed = self.accelerate() if car_ahead == None else

self.decelerate(car_ahead)

        #control de limites de velocidad

        if new_speed >= 1.0:

            new_speed = 1.0

        elif new_speed <= 0.0:

            new_speed = 0.0



        #asignacion de velocidad
        self.speed = np.array([new_speed, new_speed])



        #accion cuando el agente llega a su destino (es

eliminado)

        if self.contadorNodos >= len(self.closedList):
```

```
            self.model.schedule.remove(self)

            self.model.space.remove_agent(self)
            return



        #seguimiento de ruta creada

        if self.model.space.get_distance(self.pos,

self.closedList[self.contadorNodos].pos) > 0.1:

            if self.pos[0] >= 0 and self.pos[0] <

self.model.width and self.pos[1] >= 0 and self.pos[1] <
self.model.height:

                self.pos +=

(self.closedList[self.contadorNodos].pos - self.pos) *

self.speed * self.contador

                #print(self.pos)

                self.model.space.move_agent(self, self.pos)

                self.contador += 0.1
        else:

            self.contadorNodos += 1

            self.contador = 0



    #funcion auxiliar de aStar para encontrar las adyacencias

de un nodo

    def encontrar_adyacentes(self, actual):
        indice_actual = actual.unique_id - 1
```

```
        for i in range(len(self.model.matrix[indice_actual])):
            if self.model.matrix[indice_actual][i] == 1:
                if self.model.nodos[i] in self.closedList:
                    continue

                self.openList.append(self.model.nodos[i])

    #funcion auxiliar de aStar para encontrar el nodo adyacente
con la menor distancia hacia el objetivo
    def encontrar_menor(self, final):
        menor = 180000000
        for nodo in self.openList:
            comparar = nodo.model.space.get_distance(nodo.pos,
final.pos)
            if comparar < menor:
                nodoMenor = nodo
                menor = comparar
        return nodoMenor
```

```
    #funcion aStar para pathfinding en grafos
    def aStar(self, inicial, final):

        actual = inicial

        self.openList.append(inicial)

        while (len(self.openList) > 0 and actual != final):

            actual = self.encontrar_menor(final)

            self.openList.clear()

            self.encontrar_adyacentes(actual)
            self.closedList.append(actual)



    #funcion de deteccion de auto delantero

    def car_ahead(self):

        for neighbor in

self.model.space.get_neighbors(self.pos, 1, False):

            if type(neighbor) == Car:
                if self.model.space.get_distance(self.pos,

neighbor.pos) < 0.1:

                #if neighbor.pos[0] > self.pos[0] or

neighbor.pos[1] > self.pos[1]:

                    return neighbor
```

```
        return None

    #funcion para acelerar

    def accelerate(self):

        return self.speed[0] + 0.05



    #funcion para desacelerar

    def decelerate(self, car_ahead):

        return car_ahead.speed[0] - 0.1

#definicion del agente nodo.

#Sirve como representacion grafica del grafo y punto a seguir

para los autos

class Nodo(Agent):

    def __init__(self, model: Model, pos):

        super().__init__(model.next_id(), model)
        self.pos = pos



    def step(self):

        pass
```

```
#definicion del modelo de calle

class Street(Model):
    #constructor del modelo calle

    def __init__(self):

        super().__init__()

        #asigna el ancho y alto de la escena

        self.width = 1800

        self.height = 1200

        #asigna el espacio de la simulacion como un continuous
space

        self.space = ContinuousSpace(self.width, self.height,

False)

        #asigna el schedule del modelo aleatorizado

        self.schedule = RandomActivation(self)

        #lista que contendra los nodos del grafo

        self.nodos = []

        #grafo representado como matriz de adyacencias

        self.matrix = [

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

```
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],
            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,

0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0],

            [0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
```

Actividades pendientes

Se creó una especie de calendario, dividido en las cuatro semanas que restan del bloque. Cada semana tiene un objetivo de enfoque. Toda la semana será dedicada a la implementación de dicho objetivo en el reto. En la segunda y tercera columna se encuentra el rango de fechas para los trabajos de cada semana. Finalmente en la última columna se incluyen las horas diarias de trabajo que le estaremos dedicando al reto. Este número va incrementando ya que al aproximarse la fecha de entrega, el trabajo será más riguroso y demandante. Sin embargo es nuestra responsabilidad cumplir con una entrega de calidad.

Objetivo	**Fecha de inicio de trabajo**	**Fecha de conclusión**	**Horas dedicadas por día**
3.1 Comunicación entre agentes	Lunes 12 de Febrero	Domingo 18 de febrero	1/2 hora de trabajo al día durante la semana
5.1 Toma de decisiones multiagente	Lunes 19 de Febrero	Domingo 25 de Febrero	3/4 de hora de trabajo al día durante la semana
5.2 Aprendizaje multiagente	Lunes 26 de feberero	Domingo 3 de marzo	1 hora de trabajo al día durante la semana
Integración del reto	Lunes 11 de marzo	Viernes 17 de marzo	2+ horas de trabajo al día durante la semana