UNIVERSIDAD NACIONAL DE INGENIERÍA
FACULTAD DE CIENCIAS

CC0F6 A Tópicos de Ciencia de la Computación III
Profesor: Gipsy Miguel Angel Arrunategui Angulo

TRABAJO FINAL 2022-2

Modelamiento basado en agentes

Autores:

George Bryan Vera Esquives
(20184159B)
Roberto Alexis Cerna Espiritu
(20184108I)
Brando Miguel Palacios Mogollon
(20180483J)
Kenjhy Javier Bazan Turin
(20180574E)

1 Introducción

Existe una red de Estaciones sensoriales que toman datos meteorológicos a través de sus sensores, los cuales pueden configurarse modularmente. Las estaciones están conectadas a la red y transmiten constantemente la data recopilada.

Se presenta la necesidad de desplegar estas estaciones en sitios que no se pueden conectar a la red, por lo que las estaciones guardarán la data en una memoria Micro SD para su posterior envío.

Muchas de estas estaciones desconectadas, se despliegan en sitios inaccesibles a donde no es posible llegar (por ejemplo: cerca al cráter de un volcán o en un incendio forestal etc), por lo que la única forma de recuperar la data es a través de un dron que sobrevuele a la estación.

2 Modelo

Se requiere simular la acción del dron en su misión de recuperar la información de las estaciones mediante un ABM (Modelo de Agentes inteligentes).

- Algunas de las restricciones a tener en cuenta son las siguientes:

1. La autonomía de vuelo es de 15' (duración de la carga de la batería del dron)
2. Un dron debe garantizar el viaje de ida hacia la(s) estación(es) y retorno su base
3. Un dron puede visitar más de 1 estación en una ruta programada
4. La velocidad de descarga de la información es de hasta 10Kbps
5. Las estaciones pueden guardar archivos de hasta 1MB por lo que quizá se necesite más de un viaje para descargar la información
6. El viento (a favor o en contra) debe considerarse
7. La Altitud de vuelo debe considerarse
8. El modelo debe mostrar las trayectorias, estaciones y otros de manera animada (tipo el modelo de Ants o similares)
9. El modelo debe parametrizarse para probar distintos escenarios y efectos.
10. El modelo debe graficar las principales variables involucradas

3 Metodología

3.1 Materiales

Para poder cumplir nuestro objetivo de realizar un modelo y el análisis del vuelo de un dron en una misión. Desarrollaremos el proyecto en el lenguaje de programación Python.

Así mismo haremos una llamada a la librería de modelado basado en agentes(mesa) perteneciente a este lenguaje.

Mesa permite a los usuarios crear rápidamente modelos basados en agentes utilizando componentes centrales integrados (como cuadrículas espaciales y programadores de agentes) o implementaciones personalizadas; visualizarlos usando una interfaz basada en navegador; y analice sus resultados utilizando las herramientas de análisis de datos de Python. Su objetivo es ser la contraparte basada en Python 3 de NetLogo, Repast o MASON.

```
import random
from mesa import Agent, Model
from mesa.time import BaseScheduler
from mesa.space import Grid,MultiGrid
from mesa.datacollection import DataCollector
from mesa.visualization.modules import CanvasGrid
from mesa.visualization.ModularVisualization import ModularServer
```

3.2 Implementación

Comenzamos definiendo las cantidades de estaciones y el tamaño de batería.

Luego definiendo 2 clases, una perteneciente a los agentes, los cuales son los usuarios que interactúan y la clase modelo que vendría a representar el entorno.

Posteriormente definimos 3 clases que se heredan de la clase agente

- **Clase Drone:** donde implementaremos el drone que se usará, donde le signaremos una posición y la cantidad de batería. Teniendo en cuenta que cada paso que realice el dron corresponde a un segundo.

```
class Drone(Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.pos = model.starting_pos
        self.battery = BATTERY_SIZE
        self.Type = 'Drone'
        self.currentCapacity = 0
        self.stationVisited = 0
    def step(self):
        print("Batería Actual:", self.battery)
        in_station = False
        if(len(self.model.grid[self.pos]) > 1):
            base = True
```

```
            station_obj = None

            for e in self.model.grid[self.pos]:
                if e.Type == "Station":
                    station_obj = e
                    base = False
            if(base):
                print("dron cargado")
                self.battery = BATTERY_SIZE
                if self.stationVisited == NUMBER_STATIONS:
                    self.model.schedule.remove(self)
                    return
            else:
                if station_obj.dataSize > 10:
                    print(f"descargando -
> capacidad del drone: {self.currentCapacity} kbps")
                    print(f"data en estación {station_obj.dataSize}")
                    self.currentCapacity = self.currentCapacity  + 10
                    station_obj.dataSize = station_obj.dataSize - 10
                    self.battery -= 1
                    in_station = True

                elif station_obj.dataSize > 0:
                    print(f"descargando -
> capacidad del drone: {self.currentCapacity} kbps")
                    print(f"data en estación {station_obj.dataSize}")


                    self.currentCapacity = self.currentCapacity + stat
ion_obj.dataSize
                    station_obj.dataSize = 0
                    self.stationVisited = self.stationVisited + 1
                    self.battery -= 1
                    station_obj.visit = True
                    in_station = False
                else:
                    in_station = False


        if self.battery > 0 and not in_station:
            self.move()
            self.battery -= 1
        elif self.battery > 0 and in_station:
            pass
        else:
            self.model.schedule.remove(self)

    def move(self):
        distances = []
```

```
        base_station_pos = (self.model.width//2, self.model.height//2)

        for element in self.model.grid.coord_iter():
            agent, posx, posy = element
            if len(agent) >0 and agent[0].Type == "Station" and agent[
0].visit == False:

                distance = abs(self.pos[0] - posx) + abs(self.pos[1] -
 posy)
                data = (distance, agent, (posx,posy))
                distances.append(data)


        if len(distances) == 0:
            distances.append((0, None, base_station_pos))

        distances.sort(key = lambda x: x[0])
        goal_posx = distances[0][2][0]
        goal_posy = distances[0][2][1]

        distance_to_base = abs(goal_posx - base_station_pos[0]) + abs(
goal_posy - base_station_pos[1])

        if self.battery < distances[0][0] + distance_to_base:
            #move to base
            goal_posx = base_station_pos[0]
            goal_posy = base_station_pos[1]
            if self.pos[0] !=  goal_posx:
                # + a la izq, - a la der
                if self.pos[0] - goal_posx < 0:
                    new_position = (self.pos[0] +1 , self.pos[1])
                else:
                    new_position = (self.pos[0] -1 , self.pos[1])
            else:
                # + a abajo, - a arriba
                if self.pos[1] - goal_posy < 0:
                    new_position = (self.pos[0], self.pos[1] +1)
                else:
                    new_position = (self.pos[0], self.pos[1] -1)
            self.model.grid.move_agent(self, new_position)
            return


        if self.pos[0] !=  goal_posx:
            # + a la izq, - a la der
            if self.pos[0] - goal_posx < 0:
                new_position = (self.pos[0] +1 , self.pos[1])
            else:
                new_position = (self.pos[0] -1 , self.pos[1])
        else:
```

```
            # + a abajo, - a arriba
            if self.pos[1] - goal_posy < 0:
                new_position = (self.pos[0], self.pos[1] +1)
            else:
                new_position = (self.pos[0], self.pos[1] -1)


        self.model.grid.move_agent(self, new_position)
```

- **Clase Station:** donde implementamos las estaciones que visitará el drone y le asignamos un type para poder identificar si la estación ya fue visitada o no.

```
class Station(Agent):
    def __init__(self, unique_id, model, dataSize):
        super().__init__(unique_id, model)
        self.pos = self.generate_valid_pos(model)
        #self.possible_points = random.sample([(x,y) for x in range(mo
del.width) for y in range(model.height)], 3)
        self.dataSize = dataSize
        self.Type = 'Station'
        self.visit = False

    def generate_valid_pos(self, model):
        pos = random.sample([(x,y) for x in range(model.width) for y i
n range(model.height)], 1)[0]
        while pos in model.occupied_positions:
            pos = random.sample([(x,y) for x in range(model.width) for
 y in range(model.height)], 1)[0]
        model.occupied_positions.add(pos)
        return pos
```

- **Clase BaseStation:** donde implementamos el lugar donde parte el dron

```
class BaseStation(Agent):
    def __init__(self, unique_id, model, pos):
        super().__init__(unique_id, model)
        self.pos = pos
        self.possible_points = random.sample([(x,y) for x in range(mod
el.width) for y in range(model.height)], 1)
        self.Type = 'BaseStation'
        model.occupied_positions.add(pos)
```

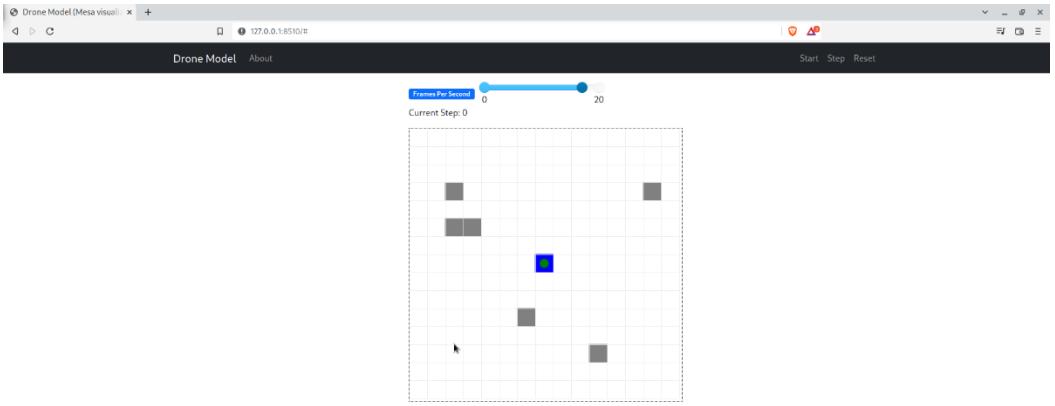
Análogamente definimos la clase que hereda el modelo como DroneModel donde vamos a definir la dimensión del espacio y las posiciones de las estaciones, teniendo en cuenta que usamos un tamaño de 6 estaciones para la implementación y la cantidad de drones que se usará(en este caso1).

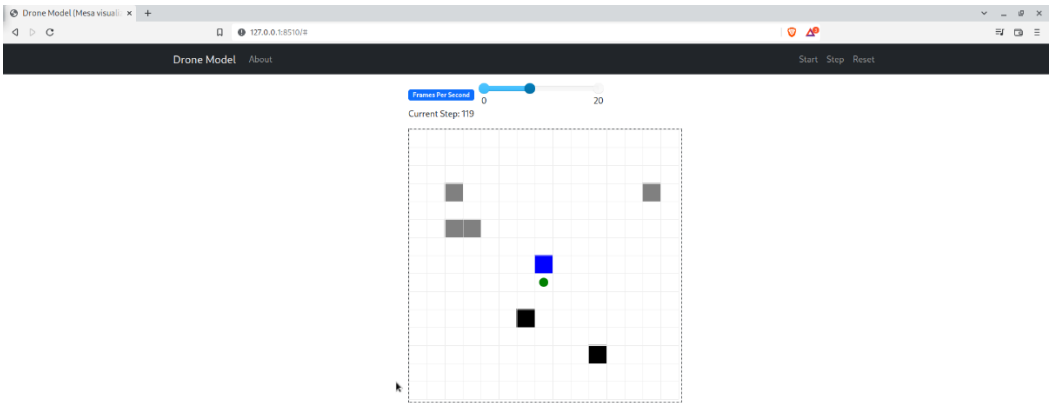
```
class DroneModel(Model):
    def __init__(self, N, width, height):
        self.width = width
        self.height = height
        self.num_drones = N
```

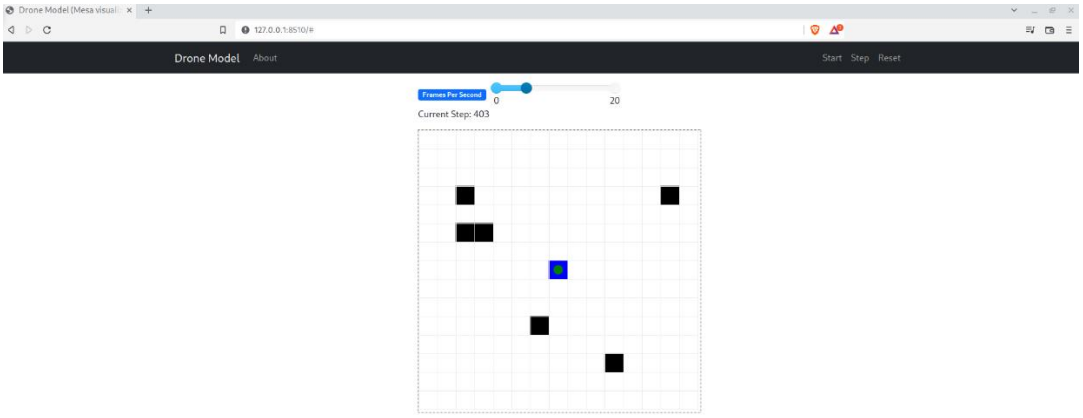
```
        self.grid = MultiGrid(width, height, torus=False)
        self.schedule = BaseScheduler(self)
        self.starting_pos = (width // 2, height // 2)
        self.occupied_positions = set()

        for i in range(self.num_drones):
            d = Drone(i, self)
            self.schedule.add(d)
            self.grid.place_agent(d, self.starting_pos)

            s = BaseStation(random.randrange(100000), self, self.starting_po
s)
            self.schedule.add(s)
            self.grid.place_agent(s, self.starting_pos)

        for i in range(NUMBER_STATIONS):
            #pos = random.sample([(x,y) for x in range(width) for y in range
(height)], 1)[0]
            s = Station(random.randrange(100000), self, random.randrange(102
4))
            self.schedule.add(s)
            self.grid.place_agent(s, s.pos)

        #self.datacollector = DataCollector(
        #    agent_reporters={"Battery": lambda x: x.battery}
        #)

    def step(self):
        print(self.schedule)
        #self.datacollector.collect(self)
        self.schedule.step()
def drone_portrayal(agent):
    portrayal = {}
    if agent.Type == "Drone":
        portrayal["Layer"] = 1
        portrayal["Shape"]="circle"
        portrayal["Filled"] = "true"
        portrayal["r"] = 0.5
        if agent.battery > 0:
            portrayal["Color"] = "green"
        else:
            portrayal["Color"] = "red"
    if agent.Type == "Station":
        portrayal["Layer"] = 0
        portrayal["Shape"]="rect"
        portrayal["Filled"] = "true"
        portrayal["w"] = 1
        portrayal["h"] = 1
        if agent.visit:
            portrayal["Color"] = "black"
```

```
        else:
            portrayal["Color"] = "gray"

    if agent.Type == "BaseStation":
        portrayal["Layer"] = 0
        portrayal["Shape"]="rect"
        portrayal["Filled"] = "true"
        portrayal["w"] = 1
        portrayal["h"] = 1
        portrayal["Color"] = "blue"
    return portrayal
```

4 Resultados







5 Conclusiones

1. Se logro la implementación del modelo y el análisis de vuelo de un dron e una misión.
