
TC2007B.401 | Integración de seguridad informática en redes y sistemas de software (Gpo 401)

*Campus Querétaro*

# M1. Actividad

***Presenta***

Carlos Adrián García Estrada

***Profesores***

*Pedro Oscar Pérez Murueta*
*Denisse Lizbeth Maldonado Flores*
*Alejandro Fernández Vilchis*

12 de noviembre del 2023

Link al Repositorio: https://github.com/SchroderCoder/TC2008B

**Planteamiento:**

Se nos da un escenario donde tenemos una matriz de cien celdas de largo y cien celdas de ancho. Se nos dice que tenemos agentes que cuando encuentran una celda sucia estos la “limpian” en un paso. La simulación consiste en saber el porcentaje de celdas sucias remanente.

**Solución:**
Se desarrolló un modelo con dos atributos principales: *dirty* y *dirty_cells* que indican respectivamente el valor de la celda en 1 o 0 (siendo 1 sucia y 0 limpia) y el número actual de celdas sucias en el modelo.

El agente tiene libertad de movimiento de radio 1 siendo 8 casillas en un sistema no toroidal, donde en los bordes se limita el número de espacios posibles a moverse. Se le añade una segunda restricción, el agente sólo puede moverse a celdas donde no haya un agente. El agente entonces se mueve a una celda vacía y analiza si esta tiene el atributo de *dirty* en 1 con el método *get_dirty_value* si resulta tenerlo el agente modifica el atributo a 0. Con un *DataCollector* recuperamos la información de la celda y señalamos las celdas vacías como blancas, las que actualmente tiene un agente como verde. El cálculo del porcentaje final se da a través del método *get_dirty_cells*. Se añadió una simulación para corroborar los resultados.

**Código:**

```
# Importamos las clases que se requieren para manejar los agentes
(Agent) y su entorno (Model).
# Cada modelo puede contener múltiples agentes.
from mesa import Agent, Model

# Debido a que necesitamos que existe un solo agente por celda,
elegimos ''SingleGrid''.
from mesa.space import MultiGrid

#Importamos Random Activation ya que queremos que los agentes
tomen decisiones 1 a la vez para evitar conflictos de choque
from mesa.time import RandomActivation

# Haremos uso de ''DataCollector'' para obtener información de
cada paso de la simulación.
from mesa.datacollection import DataCollector
```

```
#Importamos random para obtener coordenadas al azar para setear
nuestros dirty cells
import random

# matplotlib lo usaremos crear una animación de cada uno de los
pasos del modelo.
%matplotlib inline
import matplotlib
import matplotlib.pyplot as plt
import matplotlib.animation as animation
plt.rcParams["animation.html"] = "jshtml"
matplotlib.rcParams['animation.embed_limit'] = 2**128

# Importamos los siguientes paquetes para el mejor manejo de
valores numéricos.
import numpy as np
import pandas as pd

# Definimos otros paquetes que vamos a usar para medir el tiempo
de ejecución de nuestro algoritmo.
import time
import datetime
```

```
class CleaningAgent(Agent):
    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.type = 3
    def step(self):

        #Obtenemos las posibles celdas a las que el agente se
puede mover
        possible_steps = self.model.grid.get_neighborhood(
            self.pos,
            moore=True,
            include_center=False)

        #Validamos que de las posibles celdas, no haya ya un
agente ahí y
        # también validamos que si haya celdas vacías
        empty_cells = [cell for cell in possible_steps if
self.model.grid.is_cell_empty(cell)]
```

```
        if (empty_cells) :
            #Si el dirty value es 1 significa que la celda está
sucia, por lo que
            #seteamos el valor a 0 a través del agente
            # Y nos movemos a una nueva celda para realizar el
mismo proceso

            current_dirty_value =
self.model.get_dirty_value(self.pos[0],self.pos[1] )
            if current_dirty_value == 1:
                self.model.clean(self.pos[0],self.pos[1])
                self.model.update_cells()
                chosen_cell = self.random.choice(empty_cells)
                self.model.grid.move_agent(self, chosen_cell)
            else:
                chosen_cell = self.random.choice(empty_cells)
                self.model.grid.move_agent(self, chosen_cell)
```

```
#En get_grid obtenemos los contenidos de nuestra celda y seteamos
valores numéricos
#para después situar los colores en el grid

def get_grid(model):
    grid = np.zeros((model.grid.width, model.grid.height),
dtype=int)

    for (content, (x, y)) in model.grid.coord_iter():

        if content:
            grid[x][y] = 1   # Cleaning agent

        elif model.get_dirty_value(x, y) == 1:
            grid[x][y] = 2  # Dirty cell (brown)
        else:
            grid[x][y] = 0  # Empty cell
    return grid
```

```
class CleaningModel(Model):
    def __init__ (self, width, height, num_agents, dirty_percent
= 0.9):
        self.num_agents = num_agents
        self.grid = MultiGrid(width, height, torus = False)
        self.schedule = RandomActivation(self)
```

```
        self.datacollector =
DataCollector(model_reporters={"Grid": get_grid})

        #inicializamos valores, llenamos la matriz de valores en
0 para dirty
        self.dirty = np.zeros((width, height))

        #calculamos el valor de celdas sucias a través del largo
y ancho multiplicado
        #por el porcentaje de celdas sucias que queremos
        self.dirty_cells = int (width*height* dirty_percent)

        #añadimos una id única a los agentes
        id = 0

        for i in range(1, (num_agents + 1)):
                cleaner = CleaningAgent(id, self)
                self.grid.place_agent(cleaner, (1, 1))
                self.schedule.add(cleaner)
                id = id + 1

        #Elegimos coordenadas al azar y validamos que no haya un
agente ahí
        #en esa coordenada seteamos el valor de dirty a 1
        for i in range(1, self.dirty_cells+1):
            x, y = random.randrange(self.grid.width),
random.randrange(self.grid.height)
            if self.grid.is_cell_empty((x, y)):
                self.dirty[x][y] = 1

    def get_dirty_value(self,x,y):
        return self.dirty[x][y]

    def clean(self, x,y ):
        self.dirty[x][y] = 0

    #Cada vez que seteamos el valor a 0 de dirty desde el agente
también
    #reducimos el valor de dirty cells para hacer el calculo de
procentaje

    def update_cells(self):
        self.dirty_cells = self.dirty_cells -1
```

```
    def get_dirty_cells(self):
        return self.dirty_cells

    def step(self):
      if self.get_dirty_cells() == 0:
            self.running = False
      else:
        self.datacollector.collect(self)
        self.schedule.step()
```

```
WIDTH = 100
HEIGHT = 100
DIRTY_PERCENTAGE = 0.9
NUM_AGENTS = 2
# Definimos el número máximo de steps a correr
MAX_STEPS = 100

# Registramos el tiempo de inicio y ejecutamos la simulación
start_time = time.time()

model = CleaningModel(WIDTH, HEIGHT, NUM_AGENTS,
DIRTY_PERCENTAGE)

for i in range(MAX_STEPS):
    model.step()

#Hacemos calculo de celdas sucias al dividirlo entre el el tamaño
de la matriz y
#multiplicamos por 100

print("celdas sucias: ",model.get_dirty_cells())

print("Porcentaje de celdas sucias:
{:.2f}%".format(model.get_dirty_cells() / (WIDTH * HEIGHT) *
100))

# Imprimimos el tiempo que le tomó correr al modelo.

print('Tiempo de ejecución:',
str(datetime.timedelta(seconds=(time.time() - start_time))))

#Recolectamos los datos del grid
```

```
all_grid = model.datacollector.get_model_vars_dataframe()
fig, axs = plt.subplots(figsize=(7, 7))
axs.set_xticks([])
axs.set_yticks([])

# Definimos el color para la distinguir agentes, y celdas sucias
#el numero que seteamos anteriormente corresponde al valor del
array
#de colores que queremos que se muestre
cmap = plt.cm.colors.ListedColormap(['white', 'green', 'brown'])

patch = plt.imshow(all_grid.iloc[0][0], cmap=cmap, vmin=0,
vmax=2)

def animate(i):
    patch.set_data(all_grid.iloc[i][0])
    patch.set_clim(vmin=0, vmax=2)  #establecemos los limites del
color

anim = animation.FuncAnimation(fig, animate, frames=MAX_STEPS)
anim
```

**Resultados:**

Caso 1: 1 Agente, 90% de celdas sucias:

100 pasos:

```
celdas sucias:  8970
Porcentaje de celdas sucias: 89.70%
Tiempo de ejecución: 0:00:01.924450
```

1000 pasos:

```
celdas sucias:  8738
Porcentaje de celdas sucias: 87.38%
Tiempo de ejecución: 0:00:17.310935
```

10000 pasos:

```
celdas sucias:  7557
Porcentaje de celdas sucias: 75.57%
Tiempo de ejecución: 0:03:23.673359
```

Caso 2: 2 Agentes, 90% 100 de celdas sucias:

100 pasos:

```
celdas sucias:  8942
Porcentaje de celdas sucias: 89.42%
Tiempo de ejecución: 0:00:02.541251
```

1000 pasos:

```
celdas sucias:  8544
Porcentaje de celdas sucias: 85.44%
Tiempo de ejecución: 0:00:20.355225
```

10000 pasos:

```
celdas sucias:  5932
Porcentaje de celdas sucias: 59.32%
Tiempo de ejecución: 0:03:42.287550
```

Número de aspiradoras óptima:

El número máximo de aspiradoras es 10, para hacer el cálculo se hicieron simulaciones con distintos números de agentes y distintos números de pasos, tomando en cuenta que las opciones de movimiento de los agentes son más limitadas al haber más de ellos.

100 pasos:

| Número de agentes | Porcentaje Limpio |
|---|---|
| 1 | 89.6 |
| 2 | 89.5 |
| 5 | 89.3 |
| 7 | 88.9 |
| 10 | 88.6 |

1000 pasos:

| Número de agentes | Porcentaje Limpio |
|---|---|
| 1 | 89.6 |
| 2 | 87.8 |
| 5 | 82.3 |

| | |
|---|---|
| 7 | 81.56 |
| 10 | 79.1 |

Análisis:

El número de agentes limpiando la matriz no es el valor significativo para indicar el éxito del número de celdas limpias, viendo que el porcentaje de celdas es estable en número de agentes al tener 100 pasos, pero reducido significativamente al tener 1000 pasos, podemos concluir que lo mejor es tener de 3-5 agentes con una cantidad de pasos mayores a 10000. Podemos comprobarlo con el resultado de la simulación de 10000 pasos, donde tan solo 2 agentes pudieron limpiar el 41% de la matriz. La cantidad de agentes tiene un potencial en grandes cantidades de pasos.

Pasos para limpiar todo el espacio:

El comportamiento de los agentes se limita a un rango muy pequeño y tomando en cuenta que la elección de posición es arbitraria. La obtención de un resultado recae en que la simulación pare. Esto está validado dentro del código a través de la función step, donde si el porcentaje es 0 la simulación se detiene. El cálculo para determinar el número de pasos requiere de un poder de computación bastante significativo.

Se realizaron tres simulaciones con pasos de 10000, 20000 y 30000 con uno y dos agentes respectivamente. Ninguna de estas cantidades alcanzó un nivel cercano al 0% siendo 30000 pasos predeciblemente la más cercana con 36.4% de celdas sucias. Podemos estimar que la limpieza completa se llevaría a cabo en el rango de 50000-100000 pasos con 2 agentes. Realizar esta comprobación requiere de una capacidad computacional que no poseo.