Primera extensión: Transiciones probabilísticas

Modelación de sistemas multiagentes con gráficas computacionales

Omar Jiménez Armendáriz	A01732097
Francisco Rocha Juárez	A01730560
Alejandro Alfonso Ubeto Yañez	A01734977

1.1.- Versión de la extensión

Haz click **aquí** para acceder a nuestro repositorio de BitBucket.

Id de commit: **356c7ab**

1.2.- Documentación de cambios

Los cambios realizados durante esta extensión son los siguientes:

- Parámetro *Probability of spread* añadido a constructor de clase *Tree*.
- Adición de condición lógica: Si un árbol vecino está bien y además un número aleatorio multiplicado por 100 es menor a la probabilidad de esparcimiento, este se enciende en llamas.
- Adición de parámetro *Probability of spread* al constructor de la clase *Forest* con un 50 de valor por defecto.
- Creación de slider para controlar el parámetro *Probability of Spread* del modelo.

1.3.- Versión actual del código

```
from mesa import Agent, Model
from mesa.space import Grid
from mesa.time import RandomActivation

from mesa.visualization.modules import CanvasGrid
from mesa.visualization.ModularVisualization import ModularServer
```

```
from mesa.visualization.UserParam import UserSettableParameter

from mesa.datacollection import DataCollector
from mesa.visualization.modules import ChartModule

# Clase Agente: Arbol
class Tree(Agent):
    FINE = 0
    BURNING = 1
    BURNED_OUT = 2
    def __init__(self, model: Model, probability_of_spread):
        super().__init__(model.next_id(), model)
        self.condition = self.FINE
        self.probability_of_spread = probability_of_spread

    def step(self):
        if self.condition == self.BURNING:
            for neighbor in self.model.grid.neighbor_iter(self.pos, moore=False):
                if neighbor.condition == self.FINE and self.random.random() * 100 <
self.probability_of_spread:
                    neighbor.condition = self.BURNING
            self.condition = self.BURNED_OUT

# Clase Modelo: Forest
class Forest(Model):
    def __init__(self, height=50, width=50, density=0.90, probability_of_spread=50):
        super().__init__()
        self.schedule = RandomActivation(self)
        self.grid = Grid(height, width, torus=False)
        for _,x,y in self.grid.coord_iter():
            if self.random.random() < density:
                tree = Tree(self, probability_of_spread)
                if x == 0:
                    tree.condition = Tree.BURNING
                self.grid.place_agent(tree, (x,y))
                self.schedule.add(tree)

        # Recolector de información: procentaje de arboles quemados
        self.datacollector = DataCollector({"Percent burned": lambda m: self.count_type(m,
Tree.BURNED_OUT) / len(self.schedule.agents)})

    # Método estático para referenciar al modelo Forest dentro de la función lambda
    @staticmethod
    def count_type(model, condition):
        count = 0
        for tree in model.schedule.agents:
            if tree.condition == condition:
                count += 1
        return count

    def step(self):
        self.schedule.step()
        self.datacollector.collect(self)
```

```
        if self.count_type(self, Tree.BURNING) == 0:
            self.running = False

# Coloreado de agentes
def agent_portrayal(agent):
    if agent.condition == Tree.FINE:
        portrayal = {"Shape": "circle", "Filled": "true", "Color": "Green", "r": 0.75, "Layer": 0}
    elif agent.condition == Tree.BURNING:
        portrayal = {"Shape": "circle", "Filled": "true", "Color": "Red", "r": 0.75, "Layer": 0}
    elif agent.condition == Tree.BURNED_OUT:
        portrayal = {"Shape": "circle", "Filled": "true", "Color": "Gray", "r": 0.75, "Layer": 0}
    else:
        portrayal = {}

    return portrayal

grid = CanvasGrid(agent_portrayal, 50, 50, 450, 450)

# Creación de tabla que grafica datacollector
chart = ChartModule([{"Label": "Percent burned", "Color": "Black"}],
data_collector_name='datacollector')

# Implementación de Slider para manipular densidad de bosque y la probabilidad de
incendio
server = ModularServer(Forest,[grid, chart],"Forest",
                       {"density": UserSettableParameter(
                            "slider", "Tree density", 0.45, 0.01, 1.0, 0.01),
                        "probability_of_spread": UserSettableParameter(
                            "slider", "Spread probability", 50, 0, 100, 1),
                        "width":20,
                        "height":20
                        })

server.port = 8522 # The default
```

1.4.- Cambios observados

A continuación se muestran ejecuciones del modelo que demuestran la aún aleatoria naturaleza del modelo, sin embargo hacen evidente que ahora se tiene más control sobre el resultado gracias al nuevo parámetro *Probability of Spread*.

Esta variable representará una probabilidad que se comparará con un número aleatorio, si este es menor que la probabilidad de esparcimiento, entonces el árbol vecino se verá afectado y empezará a quemarse.

Visualmente la interfaz cuenta ahora con un input de tipo Slider que permite llevar a la variable en un rango del 0 al 100. Es así como se pueden apreciar los dos ejemplos: uno en el que la probabilidad de esparcimiento es del 50% y por ello solo llega a quemar un 10%

de los árboles totales, por otro lado está el ejemplo con una probabilidad del 100% y por ende se **llega a quemar el 99.9%** del bosque.

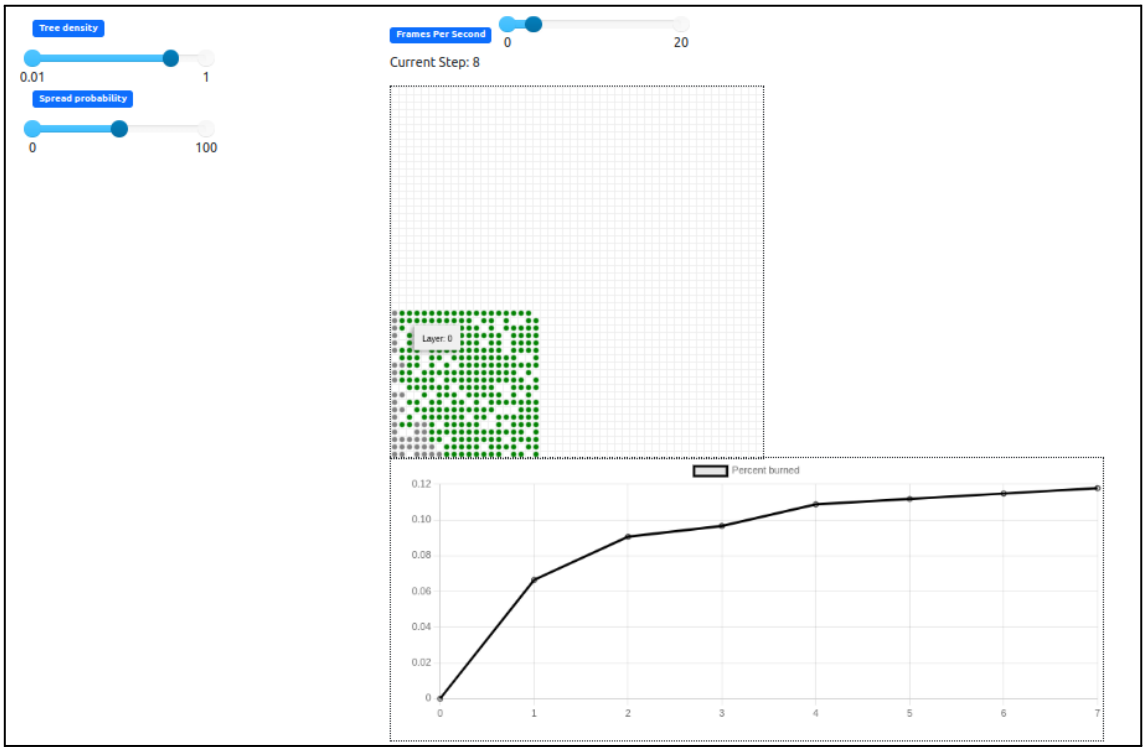

Imagen1.- Ejecución con spread probability de 50%

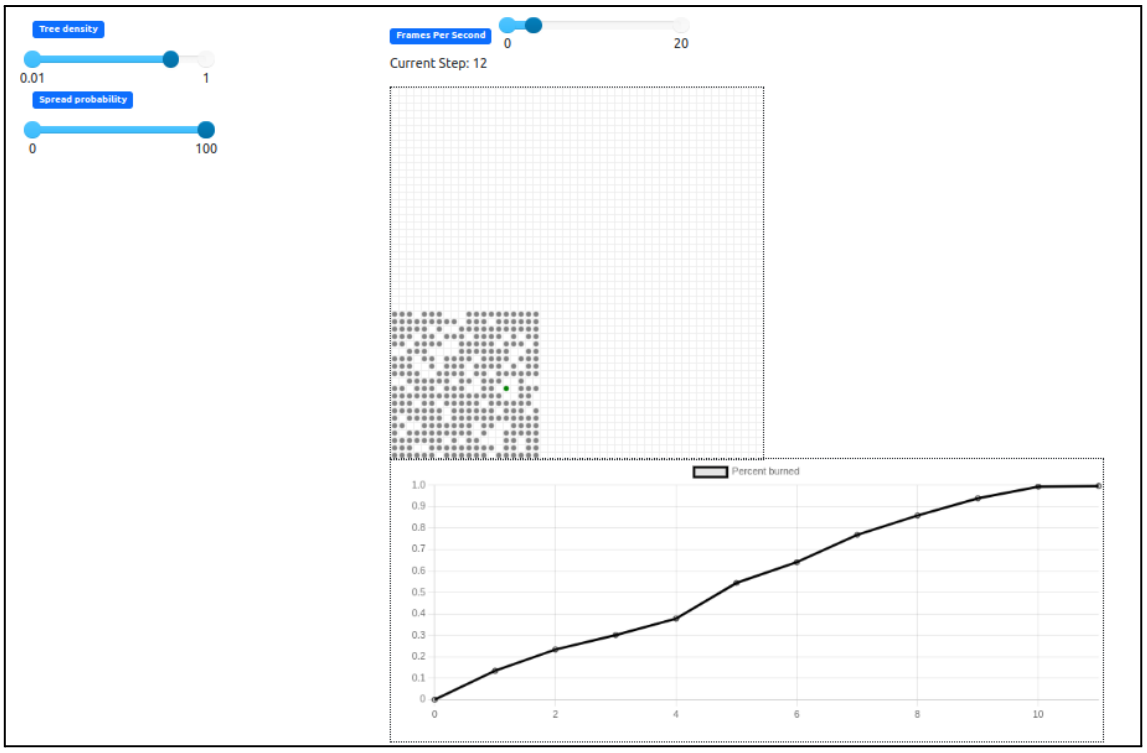

Imagen2.- Ejecución con spread probability de 100%