# Sistemas Multiagente

## Simulación

## Mesa: Utilizing Python for Agent-Based Modeling

*Authors: Vicent Botti, Vicente Julián*

# Mesa. Introducción

- Framework de código abierto para crear modelos basados en agentes en Python
- ¿Mesa?
    1. Sonaba a Mason
    2. Evocaba las “mesas” de Santa Fe, donde se encuentra el Instituto de Santa Fe.
    3. Era un nombre corto que estaba disponible en el Python Package Index (PyPI).
- Github: https://github.com/projectmesa/mesa

# Mesa. Introducción

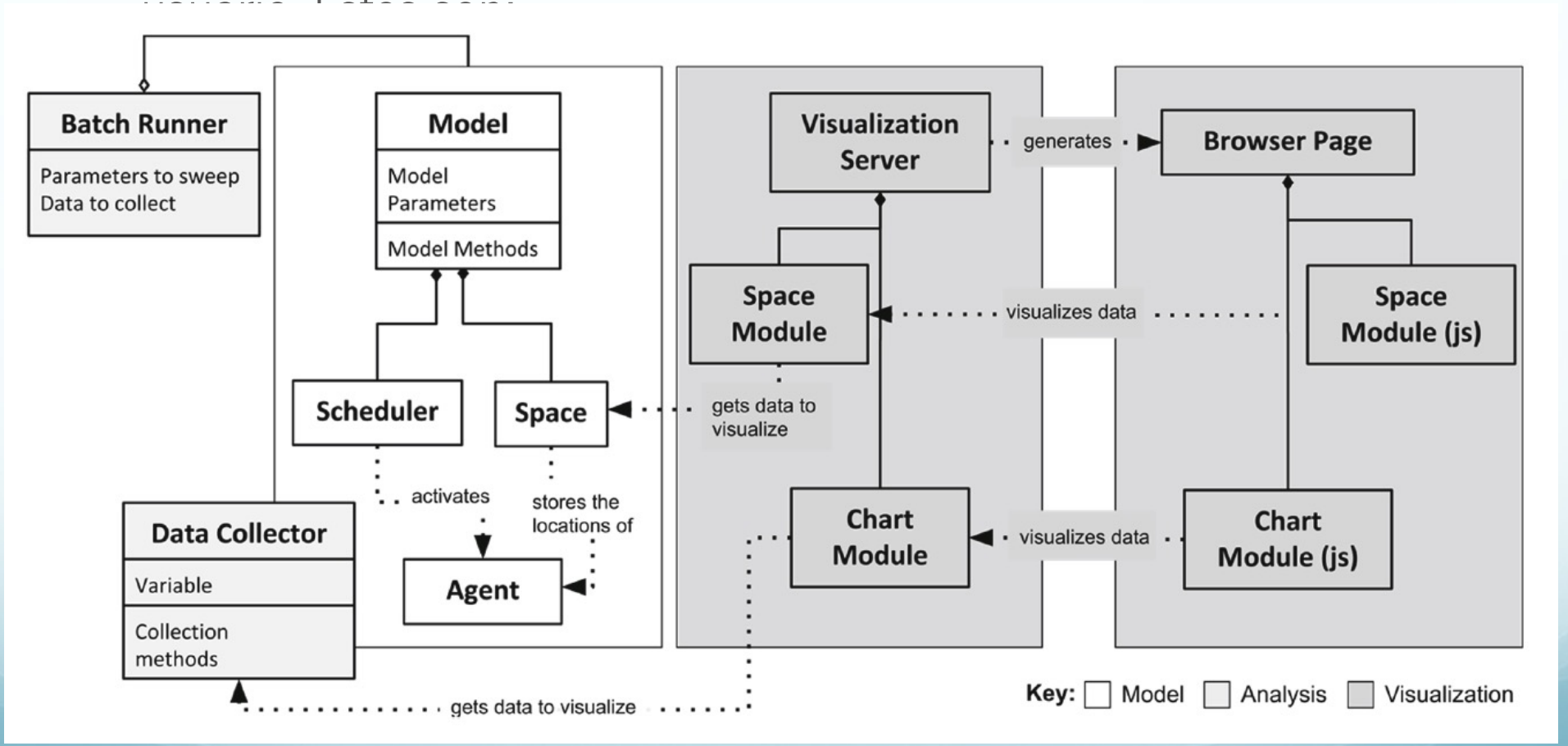
- Características:
  - Accesible: en la misma línea que NetLogo
  - Extensible: permite a los usuarios desarrollar y compartir sus propios componentes
  - No pretende ser una caja de herramientas, los componentes especializados deben ofrecerse como paquetes independientes, como Mesa-Geo
  - Se integra con herramientas populares de ciencia de datos como Jupyter notebooks y Pandas para facilitar el análisis de datos.

# Mesa. Arquitectura

- Hay tres componentes principales en Mesa desde la perspectiva del usuario. Estos son:
  - el modelo (Model, Agent, Scheduler y Space)
  - el análisis (Data Collector y Batch Runner)
  - la visualización (Visualization Server y Visualization Browser Page)
- Los componentes de Mesa están desacoplados para que puedan ser fácilmente sustituibles
  - y, en el caso del modelo, para que pueda utilizarse independientemente de los demás componentes.

# Mesa. Arquitectura

- Hay tres componentes principales en Mesa desde la perspectiva del



# Mesa. Modelo

- El modelo es el núcleo de Mesa.
- Model, es la clase principal para crear un modelo.
  - En esta clase, el usuario define: el estado inicial del modelo, lo que ocurre cuando el modelo se ejecuta, lo que ocurre en cada paso para el entorno y también el espacio que habitan los agentes.
- Agent, clase sobre la que se definen los agentes.
  - Los agentes son las entidades individuales que actúan en el modelo.

# Mesa. Modelo

```
class MoneyAgent(mesa.Agent):
    """An agent with fixed initial wealth."""

    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        print("Hi, I am agent " + str(self.unique_id) + ".")
        self.wealth = 1

    def step(self):
        if self.wealth == 0:
            return
        other_agent = self.random.choice(self.model.schedule.agents)
        other_agent.wealth += 1
        self.wealth -= 1

class MoneyModel(mesa.Model):
    """A model with some number of agents."""

    def __init__(self, N):
        self.num_agents = N
        self.schedule = mesa.time.RandomActivation(self)
        # Create agents
        for i in range(self.num_agents):
            a = MoneyAgent(i, self)
            self.schedule.add(a)

    def step(self):
        """Advance the model by one step."""
        self.schedule.step()
```

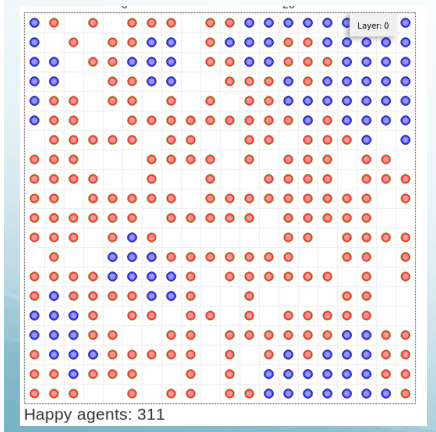
# Mesa. Modelo

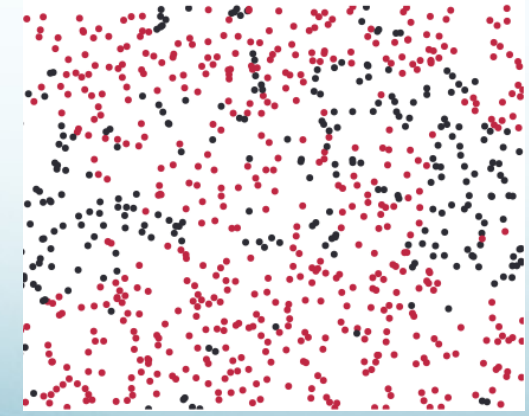
- Scheduler, ofrece una variedad de métodos para implementar la activación de agentes.
  - RandomActivation, se comporta de forma similar al planificador en NetLogo

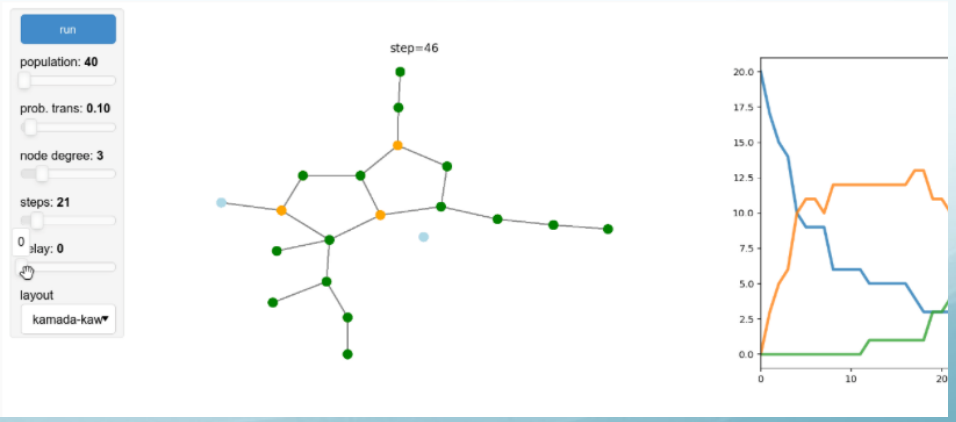
| Activation | Agent activation details |
|---|---|
| BaseScheduler | Agents are activated one at a time, in the order they were added to the scheduler (i.e., sequential activation) |
| RandomActivation | Agents are activated one at time, once per step, in random order. Reshuffled every time tick |
| SimultaneousActivation | Each agent's actions are queued based on the state of the model at the end of the previous step. Then all agents advance at the same time |
| StagedActivation | Allows agent activation to be divided into several stages instead of a single step. All agents execute one stage before moving on to the next. This scheduler tracks steps and time separately |

# Mesa. Modelo

- Space, representa el entorno
  - tiene tres posibles representaciones espaciales: continuo, cuadrícula y redes
  - permiten obtener información sobre la posición y ubicación del agente, mover el agente y obtener información sobre los vecinos del agente.





# Mesa. Modelo

```python
class MoneyAgent(mesa.Agent):
    """An agent with fixed initial wealth."""

    def __init__(self, unique_id, model):
        super().__init__(unique_id, model)
        self.wealth = 1

    def move(self):
        possible_steps = self.model.grid.get_neighborhood(
            self.pos, moore=True, include_center=False
        )
        new_position = self.random.choice(possible_steps)
        self.model.grid.move_agent(self, new_position)

    def give_money(self):
        cellmates = self.model.grid.get_cell_list_contents([self.pos])
        if len(cellmates) > 1:
            other_agent = self.random.choice(cellmates)
            other_agent.wealth += 1
            self.wealth -= 1

    def step(self):
        self.move()
        if self.wealth > 0:
            self.give_money()
```

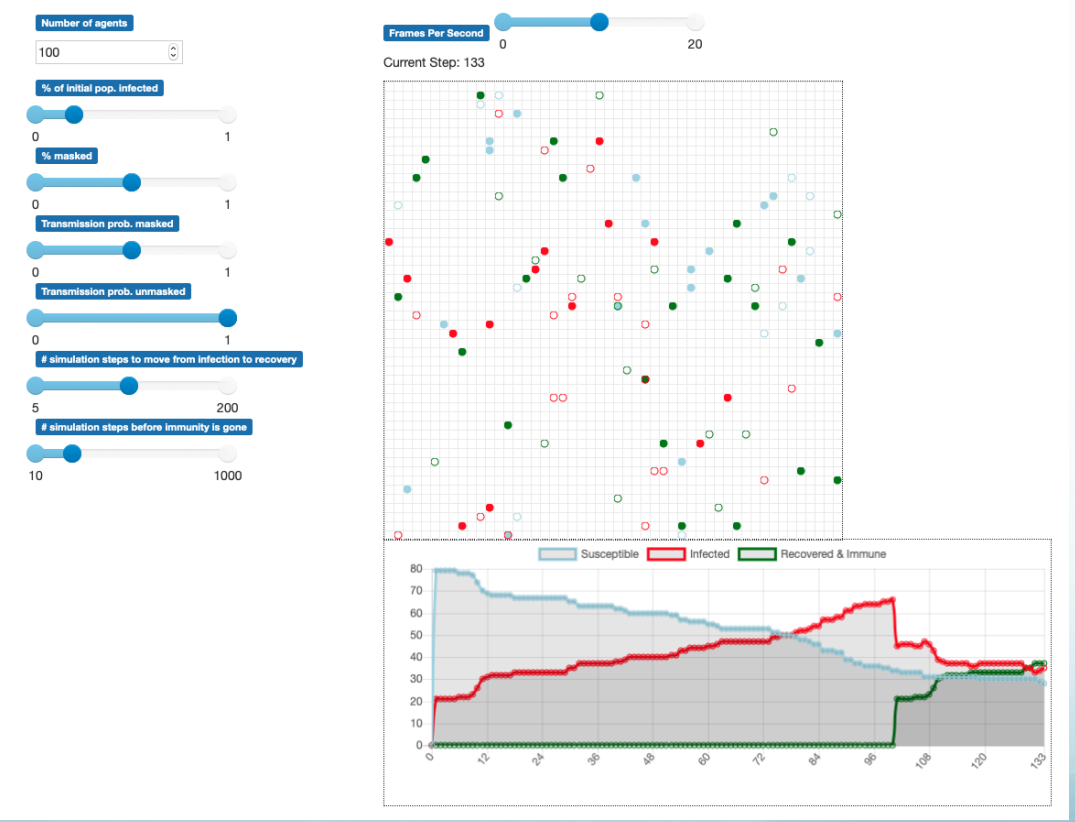
```python
class MoneyModel(mesa.Model):
    """A model with some number of agents."""

    def __init__(self, N, width, height):
        self.num_agents = N
        self.grid = mesa.space.MultiGrid(width, height, True)
        self.schedule = mesa.time.RandomActivation(self)
        # Create agents
        for i in range(self.num_agents):
            a = MoneyAgent(i, self)
            self.schedule.add(a)
            # Add the agent to a random grid cell
            x = self.random.randrange(self.grid.width)
            y = self.random.randrange(self.grid.height)
            self.grid.place_agent(a, (x, y))

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

# Mesa. Análisis

- Módulo encargado de recopilar datos y realizar un análisis.
- DataCollector,  registra, almacena y datos del modelo y de los agentes, así como los datos que no están cubiertos por las abstracciones de datos del modelo o de los agentes.
  - facilita la exportación de datos a formatos de datos comunes como Pandas DataFrames, JSON o CSV.
- BatchRunner, permite generar un conjunto de ejecuciones para todas las combinaciones posibles de valores que el usuario ha indicado.

# Mesa. Visualización

- Mesa también proporciona una visualización basada en navegador (opcional)



# Mesa. Instalación y ejecución

- Instalación `pip install mesa`
- Ejecución del modelo del ejemplo
  - Ejecución de 10 agentes en 10 steps

```
model = MoneyModel(10)
for i in range(10):
    model.step()
```

  - 100 ejecuciones del modelo con 10 steps, guardando los valores finales de riqueza

```
all_wealth = []
# This runs the model 100 times, each model executing 10 steps.
for j in range(100):
    # Run the model
    model = MoneyModel(10)
    for i in range(10):
        model.step()

    # Store the results
    for agent in model.schedule.agents:
        all_wealth.append(agent.wealth)

plt.hist(all_wealth, bins=range(max(all_wealth) + 1))
```

# Mesa. ejemplo

- https://github.com/projectmesa/mesa-schelling-example

- run.py: Lanza un servidor de visualización de modelos.
- schelling.py: Contiene la clase agente, y la clase modelo general.
- server.py: Define clases para visualizar el modelo en el navegador a través del servidor modular de Mesa, e instanciar un servidor de visualización.
- analysis.ipybn: Notebook que muestra cómo ejecutar experimentos y barridos de parámetros en el modelo

# Ejemplos

- Modelo SFM
  - Modificarlo para evitar bloqueos
- Modelo SIR
  - Modificarlo para convertirlo en SEIR