




# Mesa 3: Agent-based modeling with Python in 2025

Ewout ter Hoeven <sup>1</sup>, Jan Kwakkel <sup>1</sup>, Vincent Hess <sup>2</sup>, Thomas Pike <sup>3</sup>,  
Boyu Wang <sup>4</sup>, rht <sup>5</sup>, and Jackie Kazil <sup>3</sup>

<sup>1</sup> Delft University of Technology (Faculty of Technology, Policy and Management), the Netherlands <sup>2</sup> Independent Researcher, Germany <sup>3</sup> George Mason University (Department of Computational Social Science), United States <sup>4</sup> University at Buffalo (Department of Geography), United States <sup>5</sup> Independent Researcher

DOI: [10.21105/joss.07668](https://doi.org/10.21105/joss.07668)

## Software

- [Review](#) 
- [Repository](#) 
- [Archive](#) 

Editor: [George K. Thiruvathukal](#)

## Reviewers:

- [@martibosch](#)
- [@jofmi](#)

Submitted: 30 December 2024

Published: 28 March 2025

## License

Authors of papers retain copyright and release the work under a Creative Commons Attribution 4.0 International License ([CC BY 4.0](#)).

## Summary

Mesa is an open-source Python framework for agent-based modeling (ABM) that enables researchers to create, analyze, and visualize agent-based simulations. Mesa provides a comprehensive set of tools and abstractions for modeling complex systems, with capabilities spanning from basic agent management to sophisticated representation of spaces where agents interact. First released in 2014 and published in Masad et al. (2015) (with updates published in Kazil et al. (2020)), this paper highlights advancements and presents Mesa in its current version (3.1.5) as of 2025.

## Statement of need

Agent-based models (ABMs) are composed of autonomous, heterogeneous agents interacting locally with other agents. These interactions give rise to emergent phenomena. The aggregate dynamics of a system under study emerge from these local interactions (Joshua M. Epstein, 1999; Joshua M. Epstein & Axtell, 1996). This type of modeling quickly grew more sophisticated, requiring frameworks to execute them. This led to the establishment of NetLogo in 1999 and MASON in 2003.

NetLogo is the most widely adopted tool and the first to make ABMs accessible, but it only allows for small models. MASON is Java-based, allowing for advancements in scalability and speed above NetLogo, but MASON is difficult for non-programmers. Both of these tools did not serve models over HTTP, which allows for hosting models on the web, nor did they take advantage of the rich scientific Python ecosystem. In response to these needs, Mesa was created with the goal of accessibility – targeting both beginner and advanced programmers. The major release of Mesa 3 provides advanced usability and stabilized functionality. These features include enhanced management of agents, data collection advancements, an improved visualization framework, and making it easier for researchers to create and analyze complex simulations.

## Applications

Since its creation in 2014, Mesa has been applied to modeling a wide range of phenomena from economics and sociology to ecology and epidemiology and has been cited in more than 500 papers and 800 authors. Mesa has been applied across diverse domains, including:

- Infrastructure resilience and post-disaster recovery planning (Sun & Zhang, 2020)
- Market modeling, including renewable energy auctions and consumer behavior (Anatolitis & Welisch, 2017)

- Transportation optimization, such as combined truck-drone delivery routing ([Leon-Blanco et al., 2022](#))
- Recommender systems analysis examining consumer-business value tradeoffs over time ([Ghanem et al., 2022](#))
- Climate adaptation modeling examining household-level behavioral responses to environmental shocks ([Taberna et al., 2023](#))
- SEIR modeling of Sars-CoV-2 (Covid-19) ([Pham et al., 2021](#))
- Management of edge computing resources ([Souza et al., 2023](#))

These applications showcase Mesa's versatility in modeling complex systems with autonomous interacting agents, whether representing individual consumers, infrastructure components, buildings, or vehicles.

The framework is particularly suited for:

- Models with heterogeneous agent populations
- Systems requiring sophisticated spatial interactions
- Interactive exploration of parameter spaces
- Teaching and learning agent-based modeling

## Core capabilities

Mesa is a Python-based framework for ABM that provides a comprehensive set of tools for creating, running, and analyzing ABMs. Mesa integrates with the wider scientific Python ecosystem with libraries such as [NumPy](#), [pandas](#), [Matplotlib](#), [NetworkX](#), and more. The backend of the framework is written in Python, while the front-end uses a Python implementation of React. The modular architecture is comprised of three main components:

1. Core ABM components (*i.e.*, agents, spaces, agent activation, control over random numbers) to build models
2. Data collection and support for model experimentation
3. Visualization systems

This decoupled design allows selective use of components while enabling extension and customization.

Mesa follows a two-track development model where new features are first released as experimental before being stabilized. Experimental features are clearly marked as such and may have their APIs change between releases. They are graduated to stable status once their APIs and implementations are proven through community testing and feedback. Stable features follow semantic versioning.

## Core ABM components

### Model

The central class in Mesa is the Model. To build a model, the user instantiates a model object, creates a space within it, and populates the space with agent instances. Since ABMs are typically stochastic simulations, Mesa includes a random number generator and, for reproducibility purposes, allows the user to pass a seed.

```
class SimpleModel(mesa.Model):
    def __init__(self, n_agents=10, seed=42):
        super().__init__(seed=seed) # Initialize Mesa model with random seed

        SimpleAgent.create_agents(self, n_agents, energy=100)
```

```
def step(self):  
    self.agents.shuffle_do("step") # Activate all agents in random order
```

## Agents

Central to ABMs are the autonomous heterogeneous agents. Mesa provides a variety of base agent classes which the user can subclass. In its most basic implementation, an agent subclass specifies the `__init__` and `step` method. Any subclass of the basic mesa agent subclass registers itself with the specified model instance, and via `agent.remove` it will remove itself from the model. It is strongly encouraged to rely on `remove`, and even extend it if needed to ensure agents are fully removed from the simulation. Sometimes an agent subclass is referred to as a “type” of agent.

```
class SimpleAgent(mesa.Agent):  
    def __init__(self, model, energy):  
        super().__init__(model) # Initialize Mesa agent  
        self.energy = energy  
  
    def step(self):  
        self.energy -= 1  
        if self.energy <= 0:  
            self.remove()
```

## Agent management

One significant advancement of Mesa 3 is expanded functionality around agent management. The new `AgentSet` class provides methods that allow users to filter, group, and analyze agents, making it easier to express complex model logic.

When agents are created, they automatically register with the model via `model.register_agent(self)`. This registration automatically adds the agent to an `AgentSet` that’s accessible through the model’s `agents` property. Additional `AgentSet` instances for each agent type are maintained and available through `model.agents_by_type`. These collections are automatically updated when agents are added or removed from the model.

```
# Select wealthy agents and calculate average wealth  
wealthy = model.agents.select(lambda a: a.wealth > 1000)  
avg_wealth = wealthy.agg("wealth", func=np.mean)  
  
# Group agents by type and apply behaviors  
grouped = model.agents.groupby("species")  
for species, agents in grouped:  
    agents.shuffle_do("reproduce")
```

## Spaces

Mesa 3 provides both discrete (cell-based) and continuous space implementations. In discrete spaces, an agent occupies a cell. Mesa implements discrete spaces using a doubly-linked structure where each cell maintains connections to its neighbors. The framework includes several discrete space variants with a consistent API:

- Grid-based: `OrthogonalMooreGrid`, `OrthogonalVonNeumanGrid`, and `HexGrid`
- Network-based: `Network` for graph-based topologies
- Voronoi-based: `VoronoiMesh` for irregular tessellations (where space is divided into cells based on proximity to seed points)

Example grid creation:

```
grid = OrthogonalVonNeumannGrid((width, height), torus=False, random=model.random)
```

In Mesa 3, specialized agent classes for spatial interactions in discrete spaces were added:

- `FixedAgent`: Is assigned to a cell, can access this cell, but cannot move to another cell.
- `CellAgent`: Can move between cells
- `Grid2DMovingAgent`: Extends `CellAgent` with directional movement methods

All discrete spaces support `PropertyLayers` - efficient numpy-based arrays for storing cell-level properties. This newly added feature allows for agents to interact with spatial properties of the cell more easily:

```
grid.create_property_layer("elevation", default_value=10)
high_ground = grid.elevation.select_cells(Lambda x: x > 50)
```

For models where agents need to move continuously through space rather than between discrete locations, `ContinuousSpace` allows agents to occupy any coordinate within defined boundaries:

```
space = ContinuousSpace(x_max, y_max, torus=True)
space.move_agent(agent, (new_x, new_y))
```

### Time advancement

Mesa supports two primary approaches to advancing time in simulations: incremental-time progression (tick-based) and next-event time progression.

Typically, ABMs represent time in discrete steps (often called “ticks”). For each tick, the model’s step method is called, and agents are activated to take their designated actions. The most frequently implemented approach is shown below, which runs a model for 100 ticks:

```
model = Model(seed=42)

for _ in range(100):
    model.step()
```

Before Mesa 3, all agents were activated within the step method of the model using predefined schedulers. However, the newly added `AgentSet` class provides a more flexible way to activate agents. These changes include the removal of the `Scheduler` API and its previously available fixed patterns.

```
model.agents.do("step") # Sequential activation

model.agents.shuffle_do("step") # Random activation

# Multi-stage activation:
for stage in ["move", "eat", "reproduce"]:
    model.agents.do(stage)

# Activation by agent subclass:
for klass in model.agent_types:
    model.agents_by_type[klass].do("step")
```

Mesa also includes experimental support for next-event time progression through the `DiscreteEventSimulator`. This experimental feature allows scheduling events at arbitrary timestamps rather than fixed ticks, enabling both pure discrete event-based models and hybrid approaches. The latter hybrid approach combines traditional ABM time steps with the flexibility and potential performance benefits of event scheduling. While currently experimental, this capability is being actively developed and tested:

```
# Pure event-based scheduling (experimental)
simulator = DiscreteEventSimulator()
model = Model(seed=42, simulator=simulator)
simulator.schedule_event_relative(some_function, 3.1415)
```

```
# Hybrid incremental time and next-event time progression (experimental)
model = Model(seed=42, simulator=ABMSimulator())
model.simulator.schedule_event_next_tick(some_function)
```

## Visualization

Mesa's visualization module, [SolaraViz](#), allows for interactive browser-based model exploration. Advancements with Mesa 3 update the visualization from harder-to-maintain custom code to [Solara](#), a standardized library. Usage of the visualization module can be seen below:

```
visualization = SolaraViz(
    model=model,
    components=[
        make_space_component(wolf_sheep_portrayal), # Grid visualization
        make_plot_component(["Wolves", "Sheep", "Grass"]), # Population plot
        lambda m: f"Step {m.steps}: {len(m.agents)} agents" # Text display
    ],
    model_params=model_params
)
```



Figure 1: A screenshot of the WolfSheep Model in Mesa

Key features include:

- Interactive model controls
- Real-time data visualization
- Customizable agent and space portrayal
- Support for multiple visualization types, including grids, networks, and charts

## Experimentation and analysis

### Data collection

Mesa's DataCollector enables systematic data gathering during simulations:

```
collector = DataCollector(
    model_reporters={"population": lambda m: len(m.agents)},
    agent_reporters={"wealth": "wealth"},
```

```
agenttype_reporters={
    Predator: {"kills": "kills_count"},
    Prey: {"distance_fled": "flight_distance"}
}
```

The collected data integrates seamlessly with pandas for analysis:

```
model_data = collector.get_model_vars_dataframe()
agent_data = collector.get_agent_vars_dataframe()
```

### Parameter sweeps

Mesa supports systematic parameter exploration:

```
parameters = {
    "num_agents": range(10, 100, 10),
    "growth_rate": [0.1, 0.2, 0.3]
}
results = mesa.batch_run(MyModel, parameters, iterations=5, max_steps=10)
```

## Community and ecosystem

Mesa has grown into a complete ecosystem with [extensions](#) including:

- [Mesa-Geo](#) for geospatial modeling ([Wang et al., 2022](#))
- [Mesa-Frames](#) for high-performance simulations
- A rich collection of community-contributed extensions, [example models](#), and tutorials

## Conclusions

Mesa 3 introduces significant advancements to the Python ABM framework, enhancing the core toolkit with greater control, interactivity, and speed for researchers. These notable improvements, paired with its foundational integration with the scientific Python ecosystem, modular architecture, and active community, make it an indispensable tool for researchers across disciplines working in Python who want to create and analyze agent-based models.

## Acknowledgements

The advancements leading to Mesa 3 were developed by seven maintainers (the authors) and an active community with over 140 [contributors](#). We would especially like to thank [David Masad](#) for his foundational work on Mesa.

## References

- Anatolitis, V., & Welisch, M. (2017). Putting renewable energy auctions into action—an agent-based model of onshore wind power auctions in Germany. *Energy Policy*, 110, 394–402. <https://doi.org/10.1016/j.enpol.2017.08.024>
- Epstein, Joshua M. (1999). Agent-based computational models and generative social science. *Complexity*, 4(5), 41–60.
- Epstein, Joshua M., & Axtell, R. (1996). *Growing artificial societies: Social science from the bottom up*. Brookings Institution Press; MIT Press. ISBN: 9780262550253

- Ghanem, N., Leitner, S., & Jannach, D. (2022). Balancing consumer and business value of recommender systems: A simulation-based analysis. *Electronic Commerce Research and Applications*, 55, 101195. <https://doi.org/10.1016/j.elerap.2022.101195>
- Kazil, J., Masad, D., & Crooks, A. (2020). Utilizing Python for agent-based modeling: The Mesa framework. *Social, Cultural, and Behavioral Modeling: 13th International Conference, SBP-BRiMS 2020*, 308–317. [https://doi.org/10.1007/978-3-030-61255-9\\_30](https://doi.org/10.1007/978-3-030-61255-9_30)
- Leon-Blanco, J. M., Gonzalez-R, P. L., Andrade-Pineda, J. L., Canca, D., & Calle, M. (2022). A multi-agent approach to the truck multi-drone routing problem. *Expert Systems with Applications*, 195, 116604. <https://doi.org/10.1016/j.eswa.2022.116604>
- Masad, D., Kazil, J. L., & others. (2015). Mesa: An agent-based modeling framework. *SciPy*, 51–58. <https://doi.org/10.25080/Majora-7b98e3ed-009>
- Pham, T. M., Tahir, H., Wijgert, J. H. H. M. van de, Van der Roest, B. R., Ellerbroek, P., Bonten, M. J. M., Bootsma, M. C. J., & Kretzschmar, M. E. (2021). Interventions to control nosocomial transmission of SARS-CoV-2: A modelling study. *BMC Medicine*, 19, 211. <https://doi.org/10.1186/s12916-021-02060-y>
- Souza, P. S., Ferreto, T., & Calheiros, R. N. (2023). EdgeSimPy: Python-based modeling and simulation of edge computing resource management policies. *Future Generation Computer Systems*, 148, 446–459. <https://doi.org/10.1016/j.future.2023.06.013>
- Sun, J., & Zhang, Z. (2020). A post-disaster resource allocation framework for improving resilience of interdependent infrastructure networks. *Transportation Research Part D: Transport and Environment*, 85, 102455. <https://doi.org/10.1016/j.trd.2020.102455>
- Taberna, A., Filatova, T., Hadjimichael, A., & Noll, B. (2023). Uncertainty in boundedly rational household adaptation to environmental shocks. *Proceedings of the National Academy of Sciences*, 120(44), e2215675120. <https://doi.org/10.1073/pnas.2215675120>
- Wang, B., Hess, V., & Crooks, A. (2022). Mesa-geo: A GIS extension for the Mesa agent-based modeling framework in Python. *Proceedings of the 5th ACM SIGSPATIAL International Workshop on GeoSpatial Simulation*, 1–10. <https://doi.org/10.1145/3557989.3566157>