Mesa: Agent-based modeling in Python

Lecturer: Xin Lin

Peter.org3s@gmail.com

song@gea.mpg.de

bwang44@buffalo.edu

@ peter-kinger

@ SongshGeo

@wang-boyu

WeChat:Peter-org

整体结构

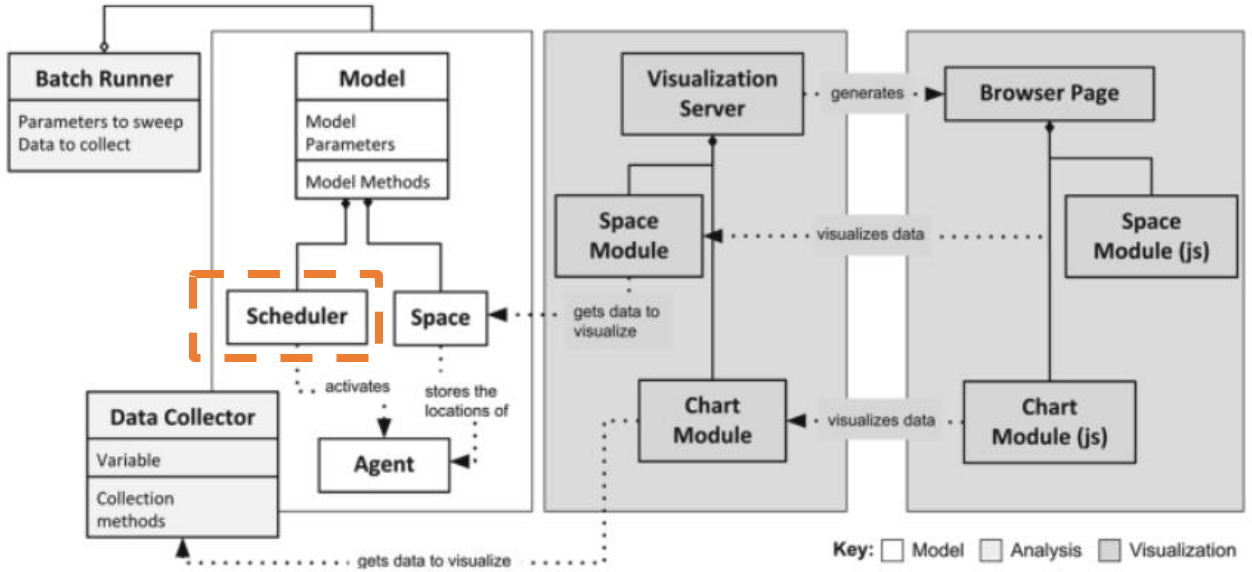


Fig. 1. Mesa model components: model, analysis and visualization.

构建模型的步骤：

1.定义 Agent 类: 定义其属性（如：年龄等）和方法（如：移动等）。
2.定义 Model 类:
3.运行/模拟: 创建模型的实例，并调用其 step() 方法多次来推进模拟。
4.数据收集与可视化 (可选): 使用 Mesa 的数据收集工具或可视化模块来观察和分析模拟结果。

Kazil J, Masad D, Crooks A. Utilizing python for agent-based modeling: The mesa framework[C]//International Conference on Social Computing, Behavioral-Cultural Modeling and Prediction and Behavior Representation in Modeling and Simulation. Cham: Springer International Publishing, 2020: 308-317.

Basic skeleton-1

学习的链接

- [Overview of the MESA library](https://mesa.readthedocs.io/stable/overview.html
- [Creating Your First Model](https://mesa.readthedocs.io/stable/tutorials/0_first_model.html
- [Adding Space](https://mesa.readthedocs.io/stable/tutorials/1_adding_space.html
- [Collecting Data](https://mesa.readthedocs.io/stable/tutorials/2_collecting_data.html
- [AgentSet](https://mesa.readthedocs.io/stable/tutorials/3_agentset.html
- [Basic Visualization](https://mesa.readthedocs.io/stable/tutorials/4_visualization_basic.html
- [Dynamic Agent Visualization](https://mesa.readthedocs.io/stable/tutorials/5_visualization_dynamic_agents.html)
- [Visualization using SpaceRenderer](https://mesa.readthedocs.io/stable/tutorials/6_visualization_rendering_with_space_renderer.html)
- [Property Layer Visualization](https://mesa.readthedocs.io/stable/tutorials/7_visualization_propertylayer_visualization.html)
- [Custom Visualization Components](https://mesa.readthedocs.io/stable/tutorials/8_visualization_custom.html):

Mesa建模一般文件夹安排

```
mesa_project/
├── run.py              # 主入口文件，运行模型与可视化
├── model.py              # 定义模型逻辑（Model 类）
├── agent.py              # 定义主体（Agent 类）
├── analysis.py
xxx...
```

具体的构建模型步骤：

```python
model = MyModel(seed=42)
for _ in range(100):
    model.step()
```

Mesa支持多种运行程序的方式:
在 run.py 里实例化你的模型并进行运行

Basic skeleton-2

```python
import mesa

class MyAgent(mesa.Agent):
    def __init__(self, model, age):
        super().__init__(model)
        self.age = age

    def step(self):
        self.age += 1
        print(f"Agent {self.unique_id} now is {self.age} years old")
        # Whatever else the agent does when activated

class MyModel(mesa.Model):
    def __init__(self, n_agents):
        super().__init__()
        self.grid = mesa.space.MultiGrid(10, 10, torus=True)
        for _ in range(n_agents):
            initial_age = self.random.randint(0, 80)
            a = MyAgent(self, initial_age)
            coords = (self.random.randrange(0, 10), self.random.randrange(0, 10))
            self.grid.place_agent(a, coords)
    self.datacollector
    def step(self):
        self.agents.shuffle_do("step")
```

Agent的属性:
比如年龄，身份，性别等，可以进一步拓展

Agent的行为:
定义随着时间变化，agent 会如何行动(agent与agent)，agent与环境互动

Model的空间:
Mesa 提供了多种agent存在且可以互动的空间类型：

- Discrete Spaces
  - Grid-based Spaces
  - Network Space:
- Continuous Space

Model 中如何初始化 space 和 agent:
定义如何创建 agent,如果有空间形式的话如何把agent放置上去

Model 中收集数据:
利用 datacollector 进行数据收集

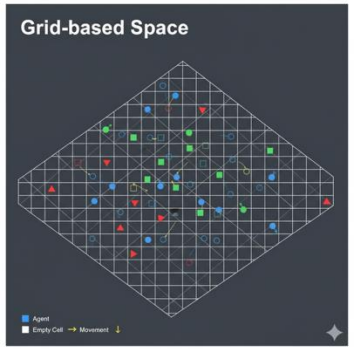
Model 中agent的激活方式:
Mesa 提供了多种方式灵活激活agent，具体使用AgentSet API:
model.agents.do("step")
model.agents.shuffle_do("step")
...

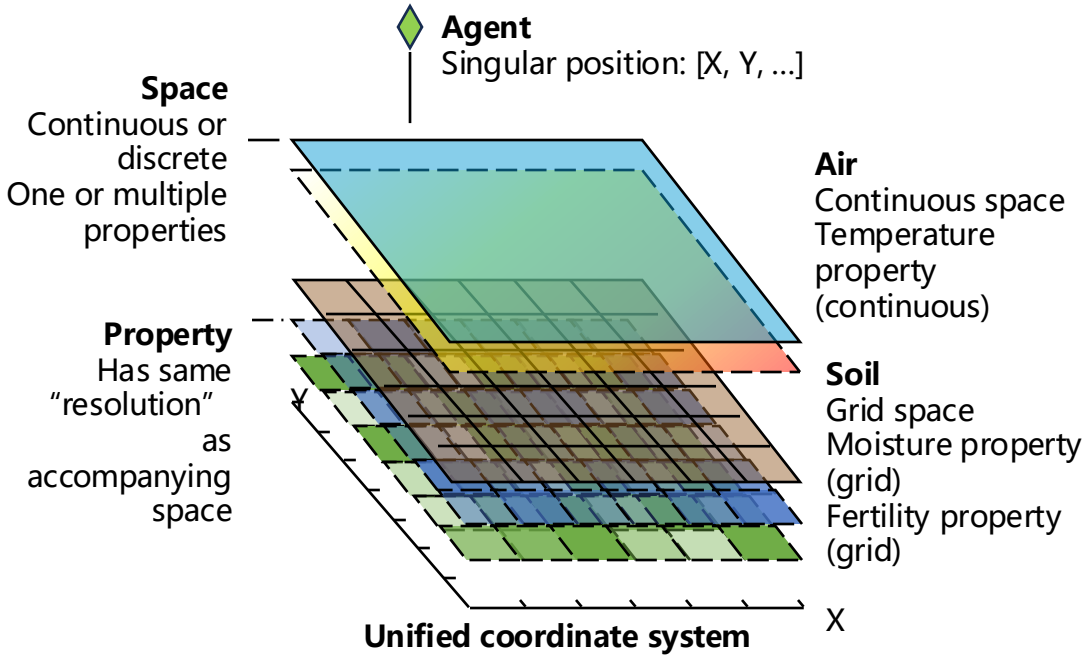
Tip:
Mesa 更新后将之前的schedule 替换为了 do 这样的操作，详见：
https://github.com/projectmesa/mesa/blob/11bbc52f28bb5a1d1220324f11bc34cc43ebc3b5/docs/migration_guide.md?plain=1

Space (空间): 定义了代理在其中移动和交互的结构

Mesa 中 Space 分类:

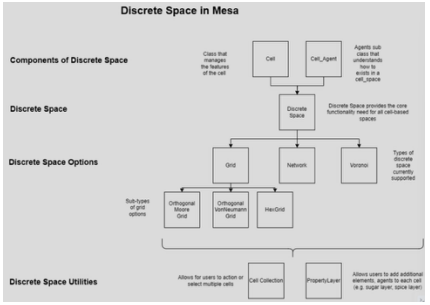
- Discrete Spaces
  - Grid-based Spaces
  - Network Space
  - Voronoi Space
  - Property Layers
- Continuous Space





API中 Space 分类:

- PropertyLayer: A data layer that can be added to Grids to store cell properties*
- SingleGrid: a Grid which strictly enforces one agent per cell.*
- MultiGrid: a Grid where each cell can contain a set of agents.*
- HexGrid
- ContinuousSpace
- NetworkGrid
- ...



- https://mesa.readthedocs.io/latest/tutorials/1_adding_space.html
- https://mesa.readthedocs.io/stable/apis/space.html

Agent：模拟的独立个体，具有属性和行为，可以与环境发生交互

Mesa 使用基于集合的方法AgentSet，使用户能够高效直观地管理其代理。大多数情况下，用户不会显式调用 AgentSet

Mesa 中 Agent 属性（基本组成）：

- model (Model): A reference to the model instance.
- unique_id (int): A unique identifier for this agent.
- pos (Position): A reference to the position where this agent is located.

Mesa 中 Agent 分类：

取决于不同的建模问题，在不同的空间中采用不同的agent：

- 无空间
  - agent
- 离散空间
  - CellAgent
  - FixedAgent
  - Grid2DMovingAgent
- 连续空间
  - agent

Mesa 中管理（ AgentSet ）：

Mesa 3.0 统一使用 AgentSet进行管理，类似于高级的集合数组

“此类扩展了 MutableSet 和 Sequence，通过顺序保留和串行作提供类似集合的功能。”

功能

- Select
- Shuffle
- Do
- shuffle_do
- Add
- …

- https://mesa.readthedocs.io/latest/tutorials/3_agentset.html
- https://mesa.readthedocs.io/stable/apis/agent.html
- **https://mesa.readthedocs.io/stable/apis/discrete_space.html#mesa.discrete_space.__init__.CellAgent**
- **https://mesa.readthedocs.io/stable/apis/agent.html#mesa.agent.AgentSet**

DataCollector: 数据收集器

- DataCollector 旨在提供一种简单、标准的方式来收集 Mesa 模型生成的数据。它收集四种类型的数据：模型级数据、代理级数据、代理类型级数据和表格。

Datacollector 主要收集的数据类型：

- model-level data
- agent-level data
- agent-type-level
- Tables data

DataCollector 步骤1：

基本设置（在 `__init__` 里）

```python
class MoneyModel(mesa.Model):
    # 其他过程省略了 ....

    def __init__(self, n, width, height, seed=None):
    # Instantiate DataCollector
        self.datacollector = mesa.DataCollector(
            # 收集模型级别的数据
            model_reporters={"Gini": compute_gini},
            # 收集agent级别的数据
            agent_reporters={"Wealth": "wealth"}
        )
```

DataCollector 步骤2：

数据收集过程（在 model 的 step 中）

```python
class MoneyModel(mesa.Model):
    # 其他过程省略了 ....

    def step(self):
        # Collect data each step
        self.datacollector.collect(self)
```

DataCollector 步骤3：

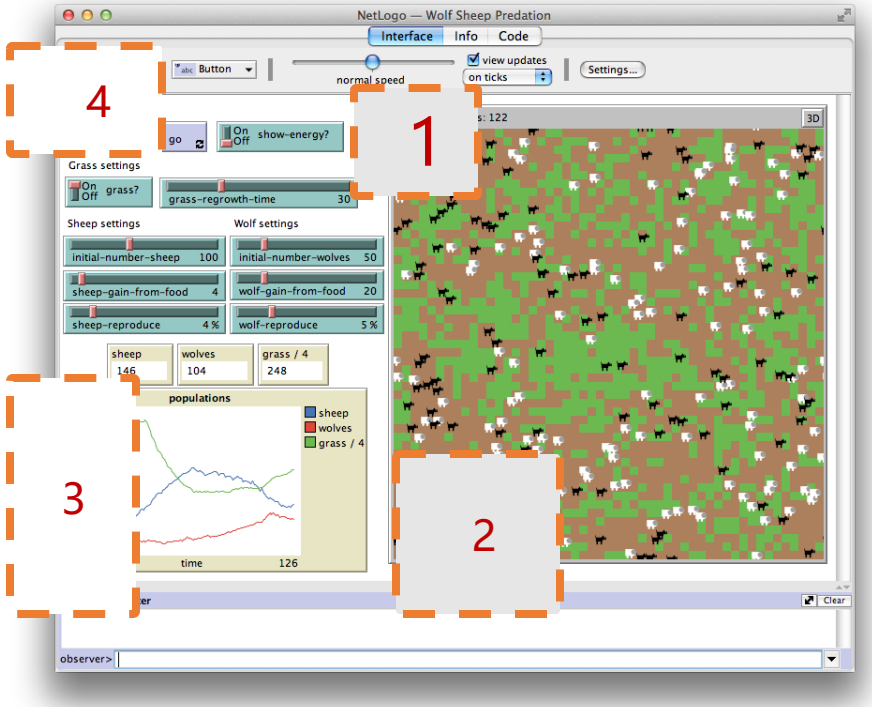
数据获取和分析过程（运行结果以后收集，有一个 pandas DataFrame）

```python
gini = model.datacollector.get_model_vars_dataframe()
```

https://mesa.readthedocs.io/stable/tutorials/2_collecting_data.html
https://mesa.readthedocs.io/stable/apis/datacollection.html

Analysis & Visualization

- Mesa 的可视化工具（尤其是新版本中）主要围绕 SolaraViz 展开，它提供了一个基于 Web 的交互式仪表板



1. 定义 agent 描绘器 (agent_portrayal)

使用网格来对 agent 返回结果进行可视化的时候，需要收集对应的数据并指定如何将数据与可视化对应起来，使用 `AgentPortrayalStyle` 对象来完成，

2. 定义空间组件（SpaceRenderer 组件）

mesa 中使用 SpaceRenderer 组件 来进行定义空间的可视化部分，你可以选择多种后端渲染方式，比如 `matplotlib` 和 `altair`

3. 定义绘图组件 (make_plot_component)

mesa 中组件是支持放在不同的位置，需要利用 `make_plot_component` 函数来定义，传入指定的变量，并指定页面（也就是在 `page=xx` 里面指定）

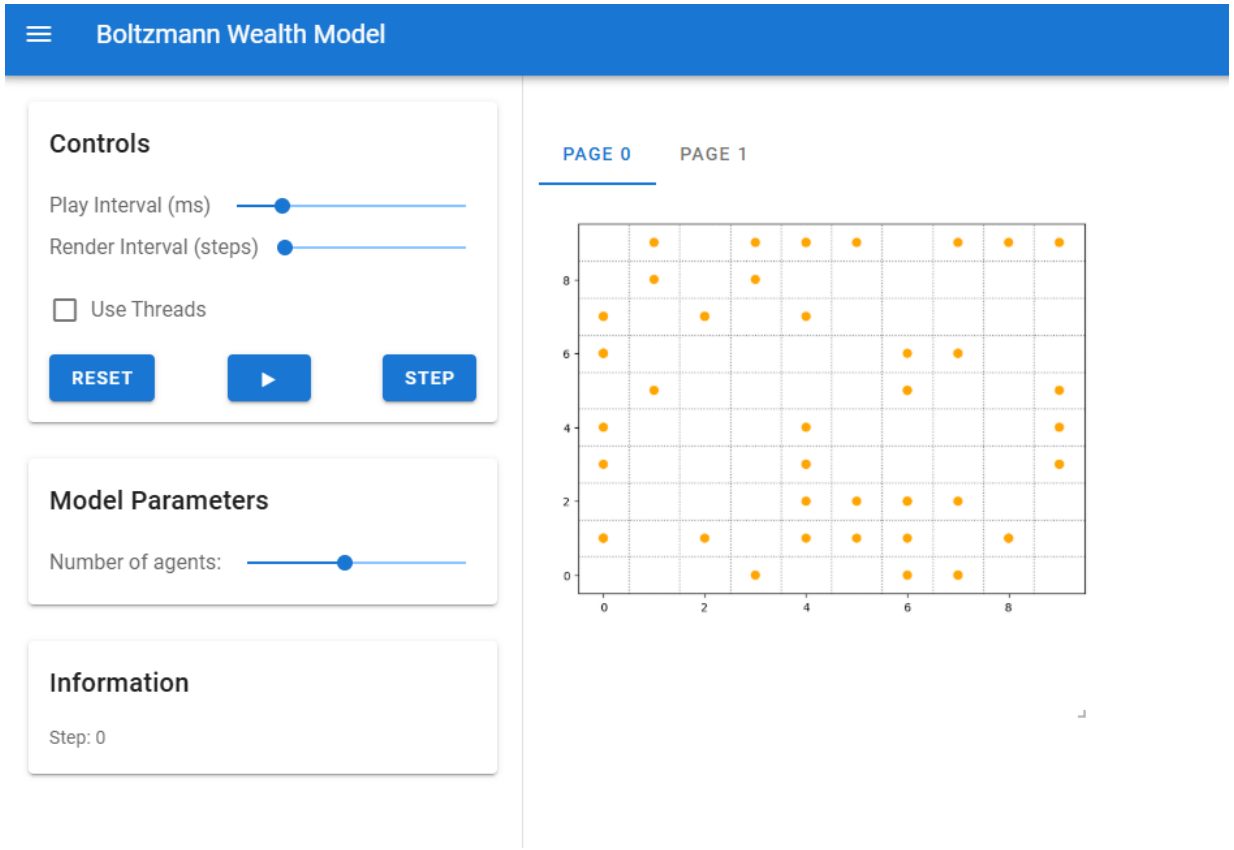
4. 参数和控制（model_params）

最开始模拟书写的 `run.py` 中使用的是固定参数，Mesa 支持类似于 Netlogo 中自定义调节参数来查看模拟结果，需要将所需要控制的参数统一打包并定义组件更改。

https://mesa.readthedocs.io/stable/tutorials/4_visualization_basic.html
https://mesa.readthedocs.io/stable/apis/visualization.html
动态的过程：https://mesa.readthedocs.io/stable/tutorials/5_visualization_dynamic_agents.html#

Analysis & Visualization 启动操作

Solarviz 可以选择jupyternotebook 或 terminal 启动



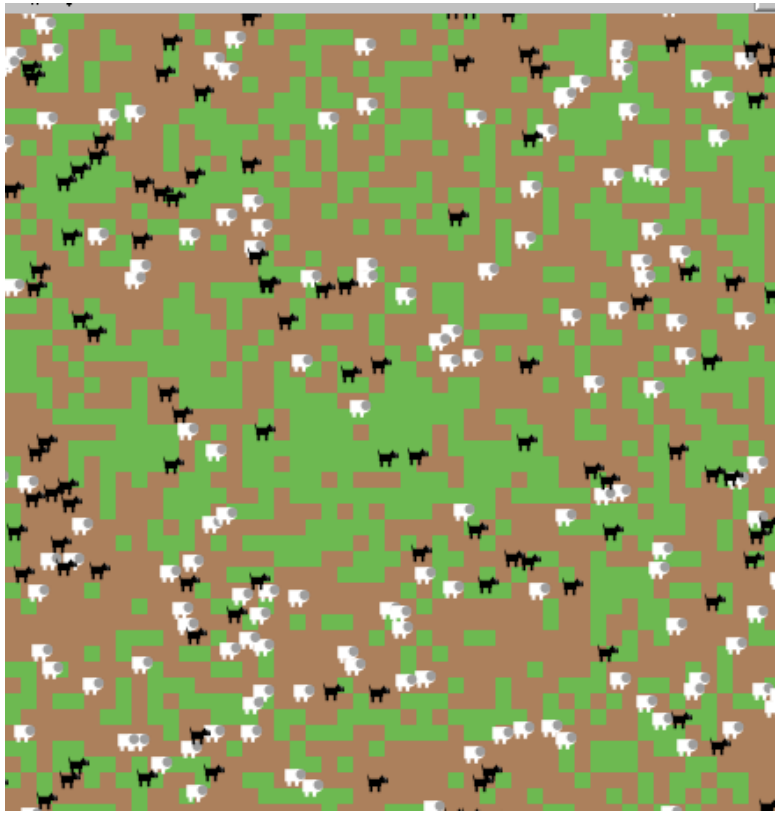

```python
# --- 可视化 ---
renderer = SpaceRenderer(model=money_model, backend="matplotlib").render(
    agent_portrayal=agent_portrayal # 调用的可视化显示部分，如何对应上色
)


GiniPlot = make_plot_component("Gini", page=1)

page = SolaraViz( # 指定页面上放置什么内容
    money_model, # 指定模型
    renderer, # 空间可视化部分（默认 page 0 ）
    components=[GiniPlot], # 绘制什么变量
    model_params=model_params,
    name="Boltzmann Wealth Model", # 页面的标题
)
# This is required to render the visualization in the Jupyter notebook
page
```

- **Jupyternotebook 中启动**
- **Terminal: solara run app.py**

https://mesa.readthedocs.io/stable/tutorials/4_visualization_basic.html
https://mesa.readthedocs.io/stable/apis/visualization.html

Wolf-Sheep-Model

- Wolf-Sheep Predation Model 使用 Mesa 模拟了在一个网格环境中，狼（捕食者）和羊（被捕食者）的种群动态。通过简单的规则，我们可以观察到它们的数量如何随着时间和空间互动而波动，甚至出现周期性的涨落。



这是一个简单的生态模型，包含三种主体：狼、羊和草。狼和羊在网格中随机游荡。它们移动都会消耗能量，并通过进食补充能量。羊吃草，如果狼和羊出现在同一个网格单元中，狼就会吃羊。

如果狼和羊有足够的能量，它们就会繁殖，产生新的狼或羊（在这个简化的模型中，繁殖只需要一个亲本）。每个单元格中的草以恒定的速度再生。如果任何狼或羊耗尽能量，它们就会死亡。

```
wolf_sheep/
├── model.py      # 模型定义
├── agent.py      # Agent 定义
└── run.py        # 运行和可视化脚本
```

https://mesa.readthedocs.io/stable/examples/advanced/wolf_sheep.html

Wolf-Sheep-Model Time示意

Wolf-Sheep-Model

流程梳理：

1. 初始化阶段：
   - 创建模型和模拟器
   - 在地图上放置狼、羊和草
   - 如果草未完全生长，调度初始生长事件
2. 模拟循环：
   - 每个时间步：
     - 狼和羊移动、交互（固定时间步进）
     - 羊吃草（如果草已完全生长）
     - 狼捕食羊（如果在同一位置）
     - 模拟器处理已调度的事件：
     - 检查是否有到期的草生长事件
   - 事件驱动：
     - - 草被吃掉时，触发 `fully_grown` setter
     - - setter 调度一个重新生长事件
     - - 事件在指定时间后执行，草重新生长

Model:

- 初始化空间（网格）
- 创建智能体
- 定义时间步行为
- 收集数据

run：

- **SpaceRenderer：空间可视化**
- **SolaraViz：交互式可视化界面**
- **DataCollector：数据收集和图表**

Agent

- Animal（基类）：

- 能量管理
- 移动行为
- 繁殖逻辑
- 死亡判断

- Sheep（羊）：

- 吃草行为
- 避狼移动

- Wolf（狼）：

- 捕食行为
- 追羊移动

- GrassPatch（草地）：

- 生长状态管理
- 自动再生调度

https://mesa.readthedocs.io/stable/examples/advanced/wolf_sheep.html