Mesa Overview

Mesa is a modular framework for building, analyzing and visualizing agent-based models.

Agent-based models are computer simulations involving multiple entities (the agents) acting and interacting with one another based on their programmed behavior. Agents can be used to represent living cells, animals, individual humans, even entire organizations or abstract entities. Sometimes, we may have an understanding of how the individual components of a system behave, and want to see what system-level behaviors and effects emerge from their interaction. Other times, we may have a good idea of how the system overall behaves, and want to figure out what individual behaviors explain it. Or we may want to see how to get agents to cooperate or compete most effectively. Or we may just want to build a cool toy with colorful little dots moving around.

Mesa Modules

Mesa is modular, meaning that its modeling, analysis and visualization components are kept separate but intended to work together. The modules are grouped into three categories:

1. **Modeling:** Modules used to build the models themselves: a model and agent classes, a scheduler to determine the sequence in which the agents act, and space for them to move around on.
2. **Analysis:** Tools to collect data generated from your model, or to run it multiple times with different parameter values.
3. **Visualization:** Classes to create and launch an interactive model visualization, using a server with a JavaScript interface.

Modeling modules

Most models consist of one class to represent the model itself; one class (or more) for agents; a scheduler to handle time (what order the agents act in), and possibly a space for the agents to inhabit and move through. These are implemented in Mesa's modeling modules:

`mesa.Model`, `mesa.Agent`

mesa.time

mesa.space

The skeleton of a model might look like this:

```
import mesa

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

    def step(self):
        print("{} activated".format(self.name))
        # Whatever else the agent does when activated

class MyModel(mesa.Model):
    def __init__(self, n_agents):
        super().__init__()
        self.schedule = mesa.timeRandomActivation(self)
        self.grid = mesa.space.MultiGrid(10, 10, torus=True)
        for i in range(n_agents):
            a = MyAgent(i, self)
            self.schedule.add(a)
            coords = (self.random.randrange(0, 10), self.random.randrange(0, 10))
            self.grid.place_agent(a, coords)

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

If you instantiate a model and run it for one step, like so:

```
model = MyModel(5)
model.step()
```

You should see agents 0-4, activated in random order. See the tutorial or API documentation for more detail on how to add model functionality.

To bootstrap a new model install mesa and run `mesa startproject`

Analysis modules

If you're using modeling for research, you'll want a way to collect the data each model run generates. You'll probably also want to run the model multiple times, to see how some output changes with different parameters. Data collection and batch running are implemented in the appropriately-named analysis modules:

mesa.datacollection

mesa.batchrunner

You'd add a data collector to the model like this:

```
import mesa

# ...

class MyModel(mesa.Model):
    def __init__(self, n_agents):
        # ...
        self.dc = mesa.DataCollector(model_reporters={"agent_count":
                                     lambda m: m.schedule.get_agent_count()},
                                agent_reporters={"name": lambda a: a.name})

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

The data collector will collect the specified model- and agent-level data at each step of the model. After you're done running it, you can extract the data as a pandas DataFrame:

```
model = MyModel(5)
for t in range(10):
    model.step()
model_df = model.dc.get_model_vars_dataframe()
agent_df = model.dc.get_agent_vars_dataframe()
```

To batch-run the model while varying, for example, the n_agents parameter, you'd use the batchrunner:

```
from mesa.batchrunner import BatchRunner

parameters = {"n_agents": range(1, 20)}
batch_run = BatchRunner(MyModel, parameters, max_steps=10,
                        model_reporters={"n_agents": lambda m:
m.schedule.get_agent_count()})
batch_run.run_all()
```

As with the data collector, once the runs are all over, you can extract the data as a data frame.

```
batch_df = batch_run.get_model_vars_dataframe()
```

Visualization modules

Finally, you may want to directly observe your model as it runs. Mesa's main visualization tool uses a small local web server to render the model in a browser, using JavaScript. There are different components for drawing different types of data: for example, grids for drawing agents moving around on a grid, or charts for showing how some data changes as the model runs. A few core modules are:

mesa.visualization.ModularVisualization

mesa.visualization.modules

To quickly spin up a model visualization, you might do something like:

```
import mesa

def agent_portrayal(agent):
    portrayal = {"Shape": "circle",
                 "Filled": "true",
                 "Layer": 0,
                 "Color": "red",
                 "r": 0.5}
    return portrayal

grid = mesa.visualization.CanvasGrid(agent_portrayal, 10, 10, 500, 500)
server = mesa.visualization.ModularServer(MyModel,
                       [grid],
                       "My Model",
                       {'n_agents': 10})
server.launch()
```

This will launch the browser-based visualization, on the default port 8521.