

# Agent-Based Modeling: Week 4

Spring 2024

Jean Clipperton

# Agenda:

- Review of server file
- Data Collector
- Model: Schelling

# Review: Server file

Relevant elements:

- Header: what we import
- Agent portrayal
- Any reporting elements to display
- Canvas / grid
- Elements in the vizualization
- Set up the server

# Data collection and reporting

This is where things will get a bit complicated. We're going to work backwards by starting with what information we WANT to display / have available, and then we'll go over how to get it.

The short of it is that you are able to report / generate charts based on the underlying data you have already tracked / collected using the **data collector**.

# Reporting

Here, we're talking about general ways to display what and how we want to do things.

The main options you have are some combination of

- **Charts:** thinking about tracking statistics in real-time
- **Values / text elements:** here, you might put up a counter, for example, of how many of X are alive / available, etc

# Reporters during the simulation

# Reporters: Text Element

You can generate text elements (I think of this as value reporters) to give live / immediate feedback on some value in the model.

## When to use this?

If you have some kind of metric or stat that you want to include. Maybe there's a measure or index that relates to the analysis -- that would be great to include.

# Reporters: Example pseudocode

To include your reporter, consider whether you want just one value, the value next to another, any text, etc.

```
def nameforThing(model):  
    return (whatever you want it to return)
```



# Reporters: Example code

Here's a simple reporter:

```
def get_happy_agents(model):  
    """  
    Display a text count of how many happy agents there are.  
    """  
    return f"Happy agents: {model.happy}"
```

--

**Task: Add a reporter for average similarity**

# But wait, there's more!

You ALSO need to call the new element later in your server file.

**Step 1:** you refer to your text elements

```
def get_happy_agents(model):  
    """  
    Display a text count of how many happy agents there are.  
    """  
    return f"Happy agents: {model.happy}"
```

**Step 2:** you call these elements in setting up the server

```
server = mesa.visualization.ModularServer(  
    model_cls=Schelling,  
    visualization_elements=[canvas_element, get_happy_agents, happy_chart],  
    name="Schelling Segregation Model",  
    model_params=model_params,  
)
```

# Reporters: ChartModule

These modules are for tracking something over time -- it can be the same statistic you're choosing to report in the text element reports or something different entirely.

## When to use this?

This is valuable when you're trying to not only look at some value but how it changes and/or interacts with other elements.

## How to use this:

Two elements: in the server file and in the model file

## Chartmodule Example: pseudocode (model file)

```
(inside __init__)

self.datacollector = DataCollector(
    model_reporters = {
        "LABEL YOU WANT": (what that thing is)
    },
    agent_reporters = {
        "LABEL YOU WANT": (what that thing is)
    }
)
```

# Chartmodule Example: pseudocode (model file)

```
Chart_name = ChartModule(  
    [{"Label": "VAR YOU WANT", "Color": "COLOR YOU WANT"}])
```

*note: there are variations in syntax depending whether you chose `import mesa` vs, say, `from mesa import visualization`*

## Example:

```
happy_chart = mesa.visualization.ChartModule([{"Label": "happy", "Color": "Black"}])
```

# Chartmodule Example: code (model file)

```
# somewhat extensive data collection
self.datacollector = mesa.DataCollector(
    model_reporters={"happy": "happy", "Avg Similarity": "similarity"}, # Model-level count of happy agents
    agent_reporters={"Number of Similar Neighbors": "similar", # Agent-level reporters can allow for individual measures
                    "Agent type": "type"}
)
```

## Chartmodule Example: code (server file)

```
happy_chart = ChartModule([{"Label": "Pct Happy", "Color": "Black"}])
```

# Server file

Relevant elements:

- Header: what we import
- Agent portrayal
- Any reporting elements to display
- Canvas / grid
- Elements in the vizualization
- Set up the server



# How nice should it be?

This is a tough question to ask and depends on the following components:

1. Your own preferences (some people want things to be always polished)
2. The audience for the model (will the model be used by anyone other than you, for example)
3. The kind of project (is this a proof of concept approach, is this going to be how you generate findings, etc)

# DataCollector: getting the data you want

We are trying to determine what happens over time / overall. To do this, we want to look at multiple runs.

We may also have questions about how parameter values interact with one another.

# DataCollector: BatchRunner pseudocode

I do this in a separate file because I like things tidy.

*(NOTE: it was batchrunner, then fixedbatchrunner, now we seem to be back to batchrunner)*

```
from model import Schelling
from mesa import batch_run
import pandas as pd

parameters = {"param": value #can have some values as static
              "varying param": range(0,x,2)} #can have values that vary

results = batch_run(Schelling,
                    parameters,
                    iterations=y,
                    max_steps=x,
                    data_collection_period = -1) #how often do you want to pull the data (-1 is default)

pd.DataFrame(results).to_csv("output.csv")
```

# DataCollector: BatchRunner example code

(from mesa\_schelling )

```
parameters = {"height": 20,
              "width": 20,
              "density": 0.8,
              "minority_pc": 0.4,
              "homophily": range(0,9,1)}

# what to run and what to collect
# iterations is how many runs per parameter value
# max_steps is how long to run the model
results = batch_run(Schelling,
                    parameters,
                    iterations=10,
                    max_steps=30,
                    data_collection_period = 1) #how often do you want to pull the data

## NOTE: to do data collection, you need to be sure your pathway is correct to save this!
# Data collection
# extract data as a pandas Data Frame
pd.DataFrame(results).to_csv("4_DataCollector/mesa_schelling/data/batch_data.csv")
```

# DataCollector: graphing it!!!

HOW YOU DISPLAY YOUR DATA MATTERS!!!

(see additional slides / resources)

Additional info here as well: [mesa.visulaization](https://mesa.readthedocs.io/en/latest/visulaization.html)

**Questions?**

# TASK: Set up a single and batch run

- What parameters did you decide to vary? (be specific about the values you chose and include the code)
- Why did you choose these parameters?
- How long did it take to run?
- How are you planning to analyze your results?

\*Include a graphic of your results and an interpretation -- how will you use this information?