Agent-Based Simulations for Protocol Design, Tokenomics, and Risk Analysis

ETHChicago 2023

Mingxuan He
mingxuanh.eth

Phoenix graduate scholar (computational economics), University of Chicago
Research fellow, Nethermind

August 4, 2023

Agenda

What are Agent-Based Models?

A programmed world where multiple **agents** live in an enviornment and interact with each other through **actions**.

- Agents can represent:
  - individuals (humans, wallets, network nodes)
  - organizations/abstract entities (DAOs, protocols)
- Actions:
  - economic (send/receive, buy/sell, deposit/withdraw)
  - social (vote, follow/unfollow)

What can you do with an ABM simulation?

- For users / investors:
  Manage risk by stress-testing the protocol with hypothetical market events like hacks and price crashes.

- For protocol / DApp engineers:
  Make design decisions (e.g. fee rates, reward tokenomics) by simulating all possibilities and optimize for the best outcome

Traditional Method vs ABM

Macro-level simulation

- Only measures aggregate outcomes (net gains/losses)
- More assumptions required
- Fixed parameters

Agent-based simulation

- Measures individual-level & aggregate outcomes
- Less assumptions required
- Customizable parameters

"All models are wrong, but some are useful."

Step 0: Understand your protocol

Ask questions like:

- Who are the primary group of actors involved in the ecosystem?
- What can each actor do?
- What rules does the system set for agents?
- Is there any tokens involved and if so how do they flow?

Excercise

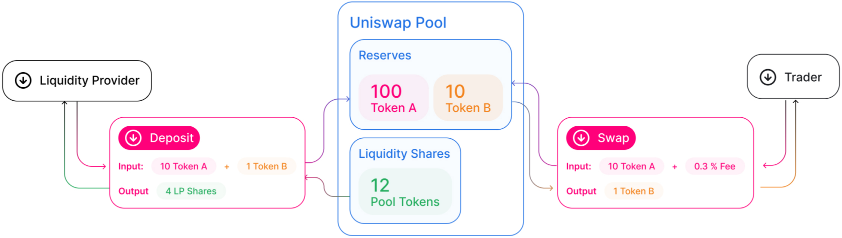
Pick a protocol / DApp and try to answer these questions.

Step 1: Build a flow chart

A flow chart is the best way to start modeling complex systems like DeFi protocols. Be sure to include:

- Agent-agent interactions
- Agent-protocol interactions
- Flow of funds / native tokens

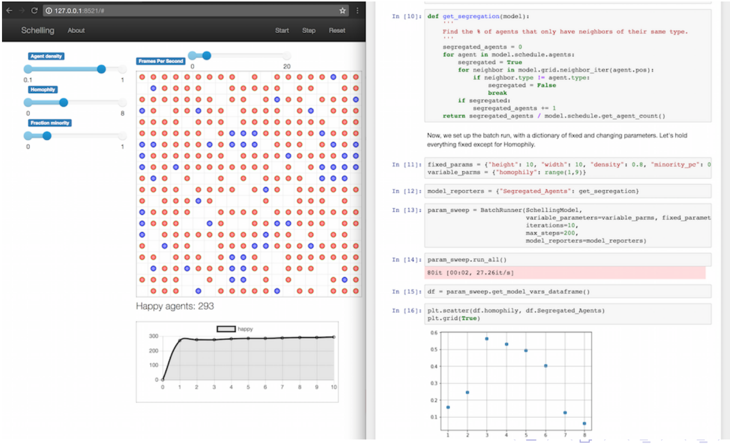
Recommended tool: Draw.io (open source)



Mesa

Open-source Python library for agent based models and simulations

- Easy to use (entry-level OOP knowledge)
- Built-in analysis and visualization modules



Step 2: Create agent classes

Code Structure

```
class TestAgent(mesa.Agent):

  def __init__(self, unique_id, model):
    super().__init__(unique_id, model)
    self.unique_id = unique_id
    self.model = model
    self.attr_X = 0
    self.attr_Y = 0

  def step(self):
    # observe state and perform actions

  def action_name(self, args):
    # single action function
```

- Subclass *mesa.Agent*
- Define **attributes** and **actions**.
- Define a **step** (policy) function

Step 3: Create model class

Code Structure

```
from mesa.time import RandomActivation

class TestModel(mesa.Model):
  def __init__(self, params, seed=None):
    super().__init__()
    self.schedule = RandomActivation()
    for i in range(100):
      agnt = TestAgent(i, self)
      self.schedule.add(agnt)
    self.datacollector = mesa.DataCollector(
      agent_reporters={"x": "attr_X"})

  def step(self):
    self.add_remove_agents()
    self.schedule.step()
    self.datacollector.collect(self)
```

- Subclass *mesa.Model*
- Create an initial state
- A mechanism to add / remove user agents each step
- Assign agents to a scheduler
- Create data collectors

Step 4: Calibration and estimation

Aim: Assigning values to parameters and initial conditions in order to reproduce real-world behavior.

Methods:

- Direct observation from data / protocol parameters
  e.g. 5% inflation rate on token, 15% node commission
- Statistical estimation (minimize distance from real data)
  e.g. monthly new users, daily ETH prices
- Meta-modeling

For a list of data sources and tools for on-chain analytics, see sites.google.com/view/mingxuanhe/resources

Step 5: Simulation

Data tracking: *mesa.DataCollector*
Track both aggregate variables (e.g. TVL, total fees) and individual agent variables (e.g. distribution of token balance, top and bottom performance of agents)

Tips:

- Run multiple batches on different random seeds and aggregate (Monte Carlo)
- Change the protocol parameters to explore "parallel universes"
- Disaster simulation: large drop in prices, large withdrawl of liquidity, etc.

Step 6: Visualization (optional)

Use *mesa.visualization.modules*

Available modules:

- Charts: bar chart, line chart, pie chart
- Grids: canvas grid, hex grid
- Network visualization
- User-settable parameter (slider / choice / number input)

Example: An ABM for Uniswap V2 AMM

- Agents: 1000 traders, 50 liquidity providers, 1 liquidity pool for token pair X and Y
- Actions: traders can swap, LPs can add/remove liquidity
- Agent attributes:
  - Trader: holding of token X and Y
  - LP: holding of token X and Y, holding of LP tokens
  - Pool: balance of token X and Y, fee rate, constant product K
- Performance metrics: TVL, slippage, impermenant loss

Extensions to the Toy Model

- Different types of traders: arbitrageurs, speculators, noise traders - each type has a different trading pattern in response to state variables
- Multi-pool model: Pool competition, triangle arbitrage,

References I

Token Engineering Commons. (2022). Token engineering fundamentals module 4 [https://tokenengineering.net/course/tef-module4/].

Uniswap. (2022). How uniswap works. https://docs.uniswap.org/contracts/v2/concepts/protocol-overview/how-uniswap-works