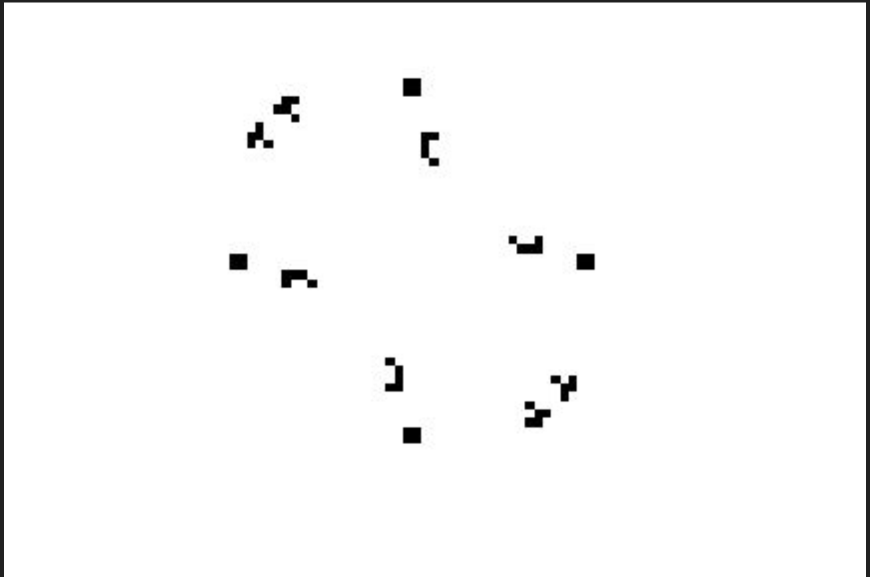
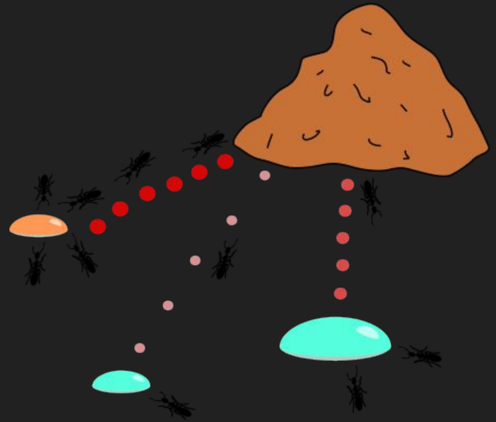
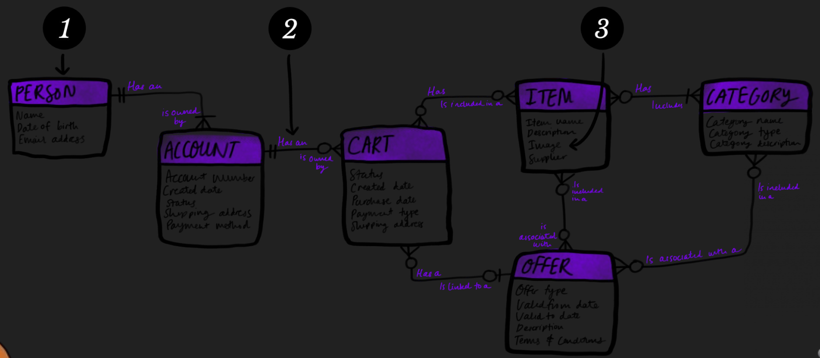
Modelling and Simulation

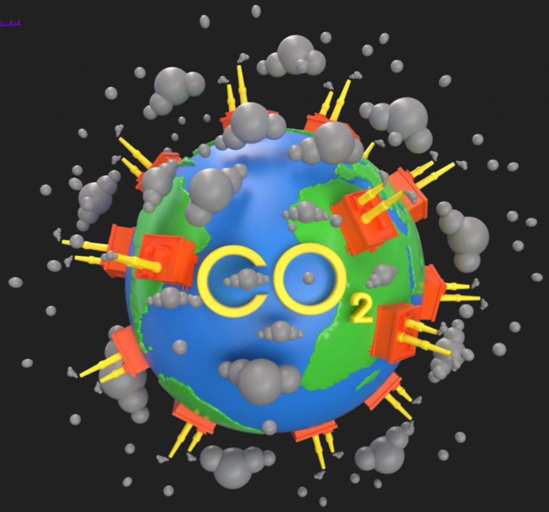
Entering the Matrix



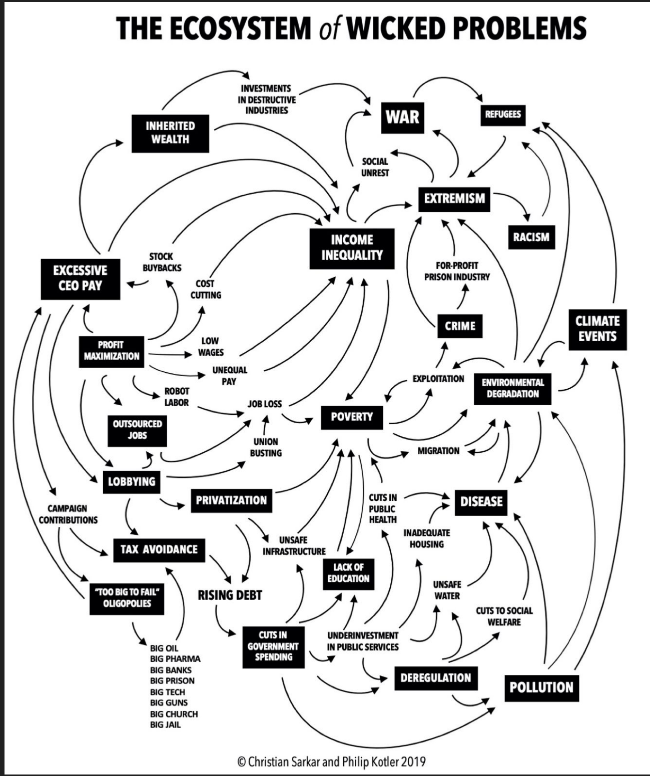
What is a System?







Systems are Complex



Models help us understand Systems

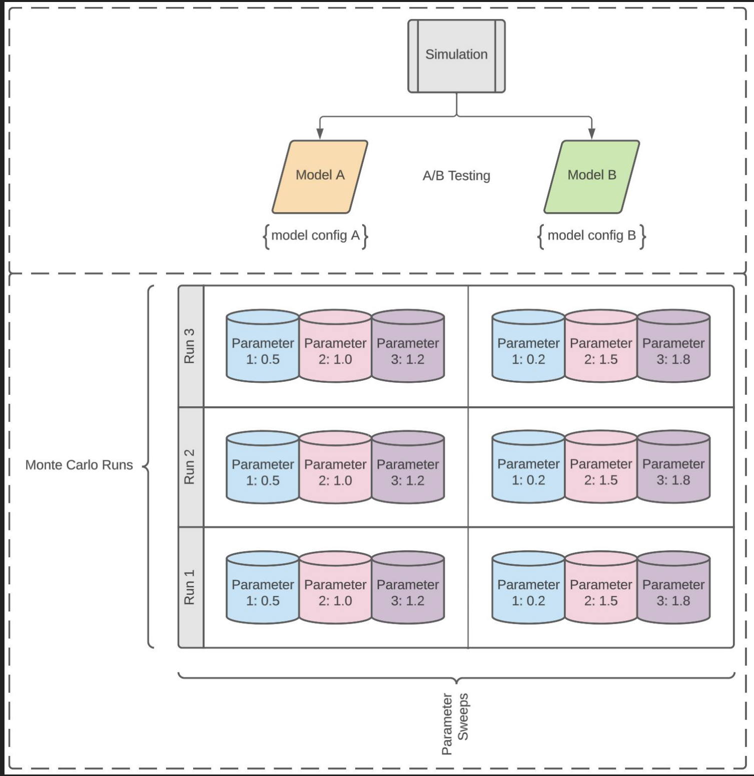
"All models are wrong, some are useful."

–George Box

"<u>A modeler builds an artificial world that reveals</u> certain types of connections among the parts of the whole—connections that might be hard to discern if you were looking at the real world in its welter of complexity."

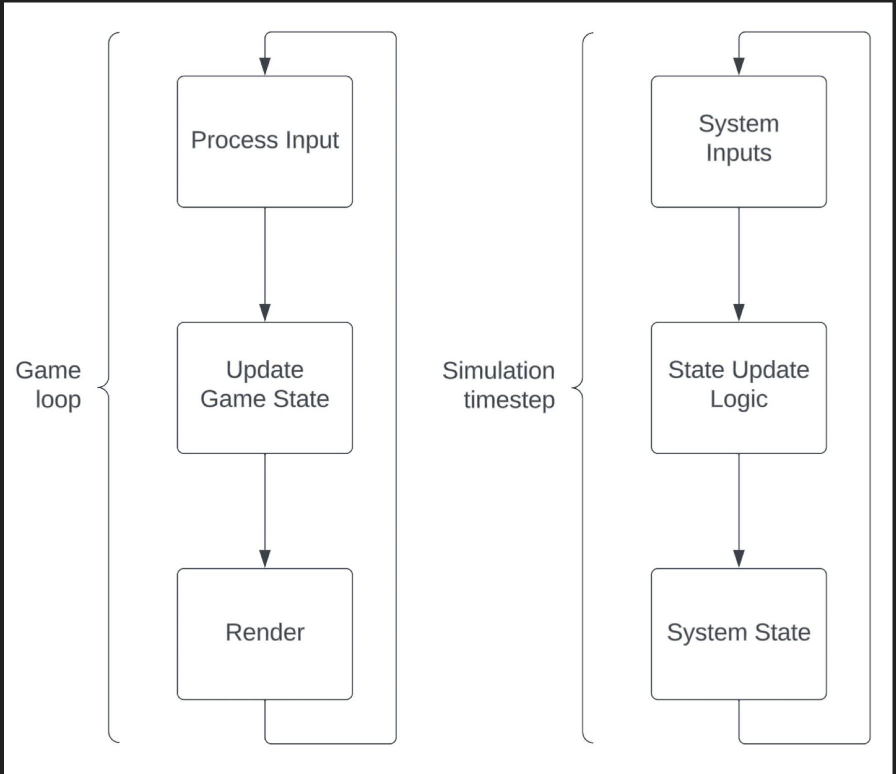
–Dani Rodrik, the book "Economics Rules"

Models are the basis for Simulations



What-if question	Description of experiment	Type of experiment	Variables / parameters	Values / ranges to be tested	Experiment results
What if scenario x occured?	By doing y, test whether z causes x.	Monte Carlo	α	[1.0, 2.0, 3.0]	System became unstable.
...	...	...	...	...	...
...	...	...	...	...	...

Simulations are like Games



Learn a new language

```
results =
  Flow.from_enumerable(1..runs)
  |> Flow.map(fn run ->
    result =
      Enum.map(range, fn
        0 ->
          nil

        _ = timestep ->
          partial_state_update_blocks
          |> Enum.with_index()
          |> Enum.map(fn {%Cadex.Types.PartialStateUpdateBlock{
                            policies: policies,
                            variables: variables
                          }, substep} ->
            policies |> Enum.each(&policy(&1, timestep, substep))
            variables |> Enum.each(&update(&1, timestep, substep))
            %Cadex.Types.State{current_state: current_state} = apply()
            %{timestep: timestep, substep: substep, state: current_state}
          end)
      end)

    flush()
    %{run: run, result: result |> Enum.filter(fn x -> !is_nil(x) end)}
  end)
  |> Enum.into([])
```

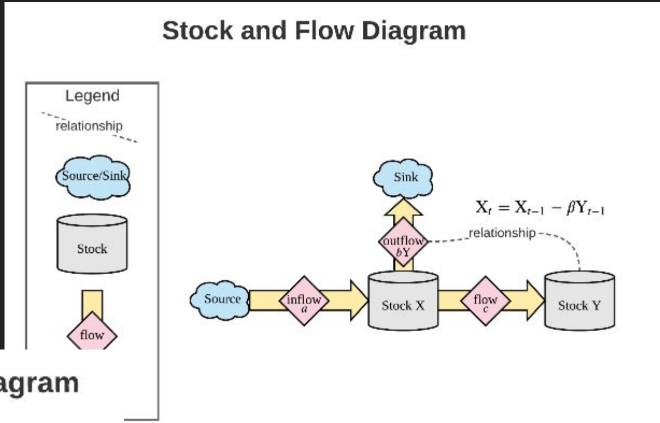
```
def execute(
    self,
) -> SimulationResults:
    self.before_execution()

    initial_timestep = self.timestep if self.timestep else 0
    for timestep in range(initial_timestep, initial_timestep + self.timesteps):
        self.timestep = timestep
        self.before_step()
        self.step()
        self.after_step()

    self.after_execution()
    return self.result
```

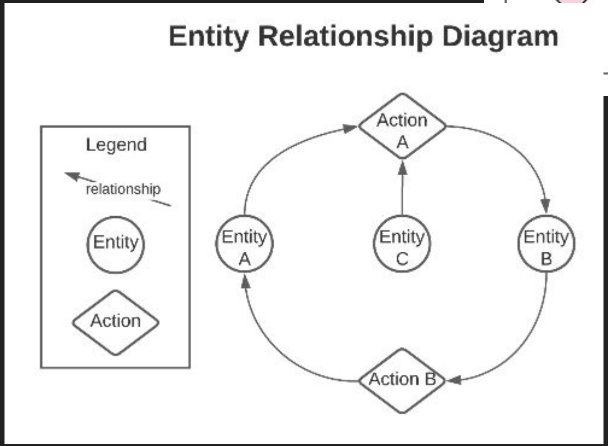
```
partialStateUpdate :: State -> PartialStateUpdateBlock -> State
partialStateUpdate s psub = updateState s $ applyPartialStateUpdate s
    where
        applyPartialStateUpdate :: State -> StateVariable
        applyPartialStateUpdate s = stateUpdateFunction s $ policyFunction s
            where
                policyFunction = head $ policies psub
                stateUpdateFunction = head $ variables psub
        updateState :: State -> StateVariable -> State
        updateState s v = Map.insert (fst v) (snd v) s

policySignalAggregation :: Num b => [(PolicySignalKey, b)] -> [(PolicySignalKey, b)]
policySignalAggregation = map sumGroup . groupBy fstEq . sortOn fst
  where
    sumGroup (x:xs) = (fst x, sum $ map snd (x:xs))
    sumGroup _ = error "This can never happen - groupBy cannot return empty groups"
    fstEq (a, _) (b, _) = a == b

simulation :: Model -> State -> StateHistory
simulation m initialState = take 10 (iterate (\s -> partialStateUpdate s (head m)) initialStat
```

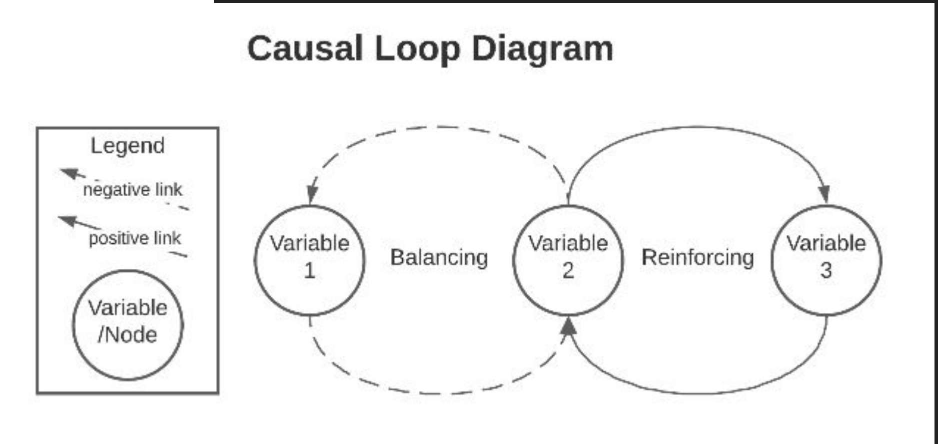
```
for run in 0..runs {
    if !param_sweep.is_empty() {
        for (subset, param_set) in param_sweep.iter().enumerate() {
            result
                .call_method(
                    "extend",
                    (_single_run(
                        result,
                        simulation_index,
                        timesteps,
                        run,
                        subset,
                        initial_state,
                        state_update_blocks,
                        param_set.extract()?,
                        deepcopy
                    )?,),
                    None,
                )
                .unwrap();
        }
```

Can anyone guess the 4 languages?

Models can be conceptual



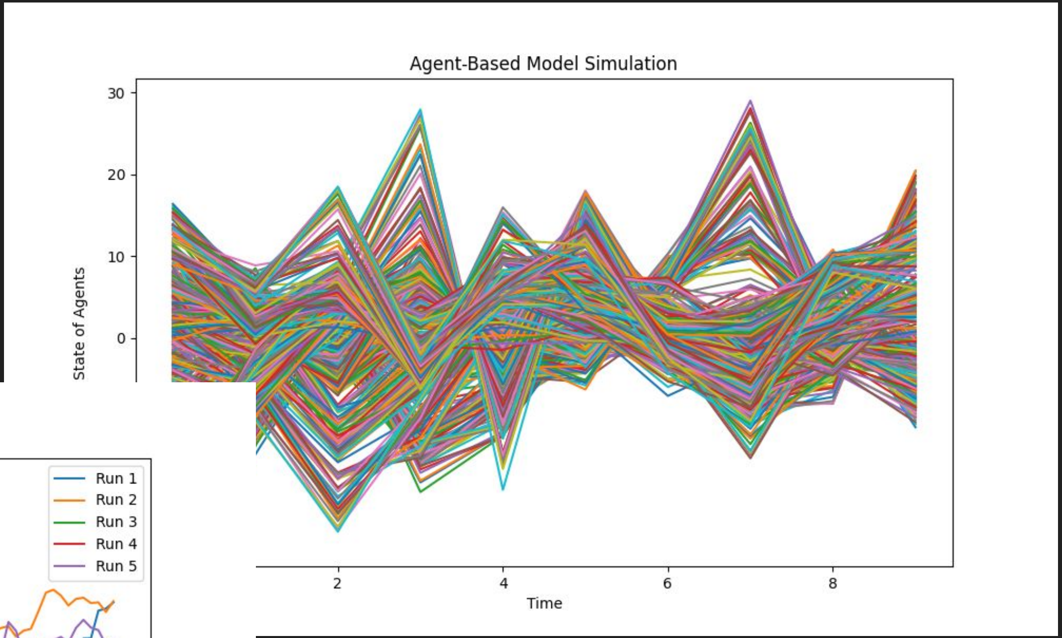


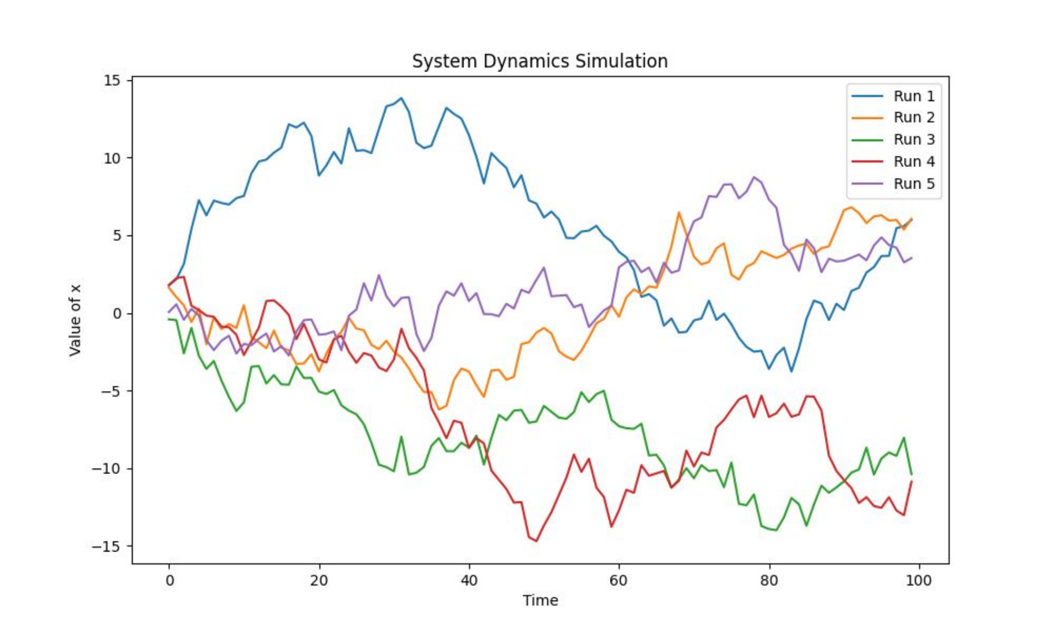


It doesn’t have to be complex!

```python
import numpy as np
import matplotlib.pyplot as plt

timesteps = 100
runs = 5

plt.figure(figsize=(10, 6))

for run in range(runs):
    np.random.seed(run)
    x = 0
    results = []
    for _ in range(timesteps):
        x += np.random.randn()  # Random
influenceresults.append(x)
    plt.plot(results, label=f'Run {run+1}')

plt.xlabel('Time')
plt.ylabel('Value of x')
plt.title('System Dynamics Simulation')
plt.legend()
plt.show()
```

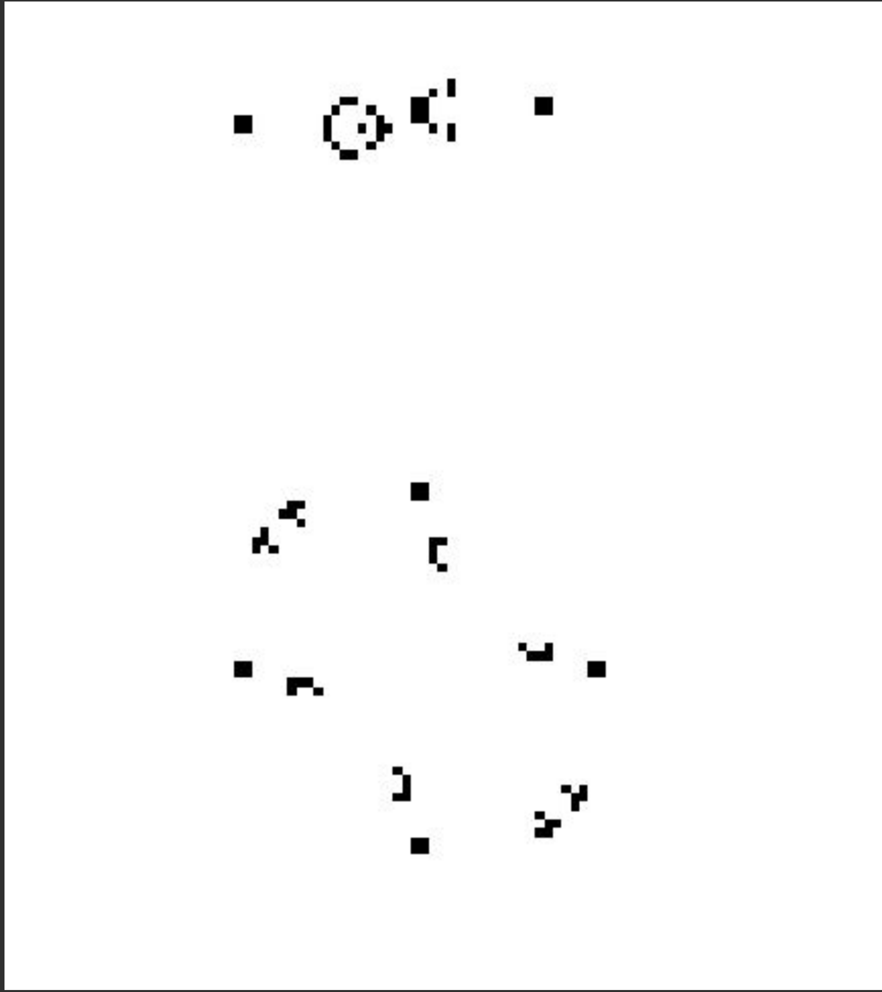
```python
import numpy as np
import matplotlib.pyplot as plt

class Agent:
    def __init__(self):
        self.state = np.random.rand()

    def update(self):
        self.state += np.random.randn()

num_agents = 10
timesteps = 100
runs = 5

plt.figure(figsize=(10, 6))

for run in range(runs):
    np.random.seed(run)
    agents = [Agent() for _ in range(num_agents)]
    for t in range(timesteps):
        for agent in agents:
            agent.update()
        states = [agent.state for agent in
agents] plt.plot(states, label=f'Run {run+1}')

plt.xlabel('Time')
plt.ylabel('State of Agents')
plt.title(f'Agent-Based Model Simulation')
plt.show()
```


Agent-Based Model Simulation
State of Agents
Time
30
20
10
0
2
4
6
8


System Dynamics Simulation
Value of x
Time
Run 1
Run 2
Run 3
Run 4
Run 5
15
10
5
0
−5
−10
−15
0
20
40
60
80
100

Conway's Game of Life

1. Any live cell with fewer than two live neighbours dies, as if by underpopulation.
2. Any live cell with two or three live neighbours lives on to the next generation.
3. Any live cell with more than three live neighbours dies, as if by overpopulation.
4. Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

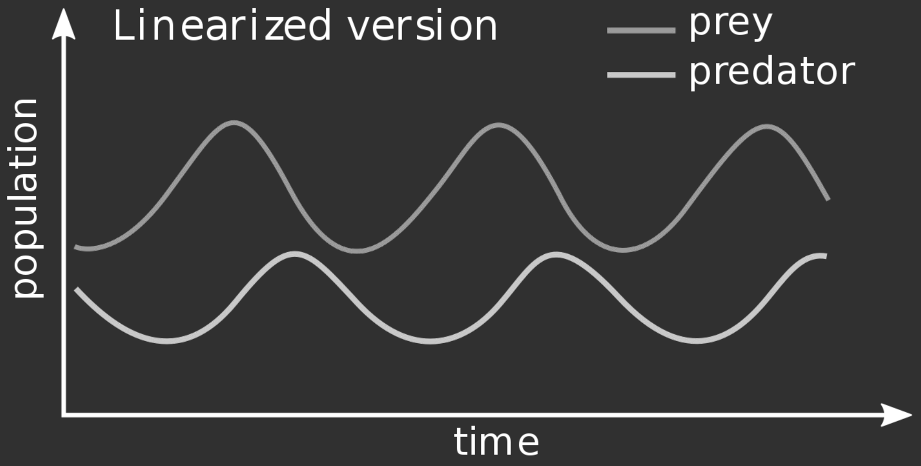


Lotka-Volterra Equations

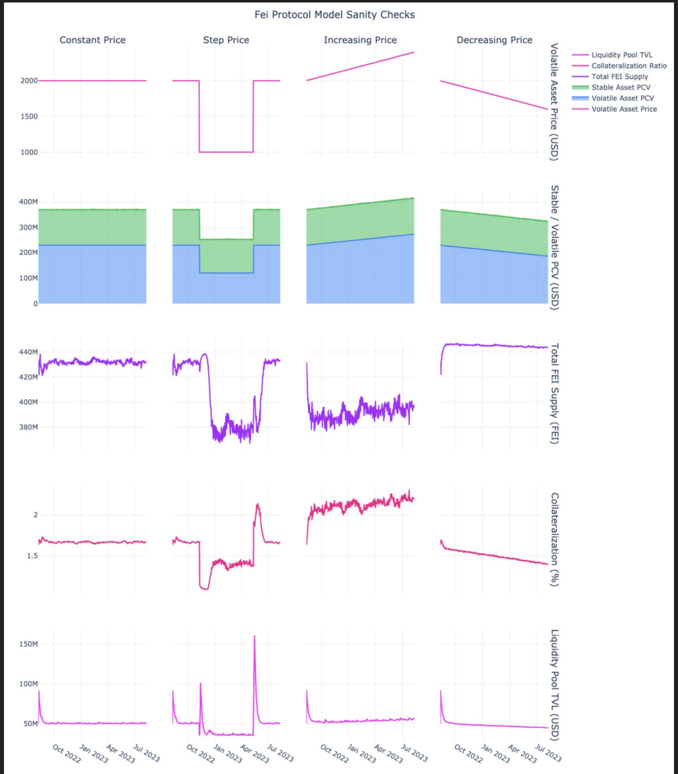
aka Predator-Prey Model

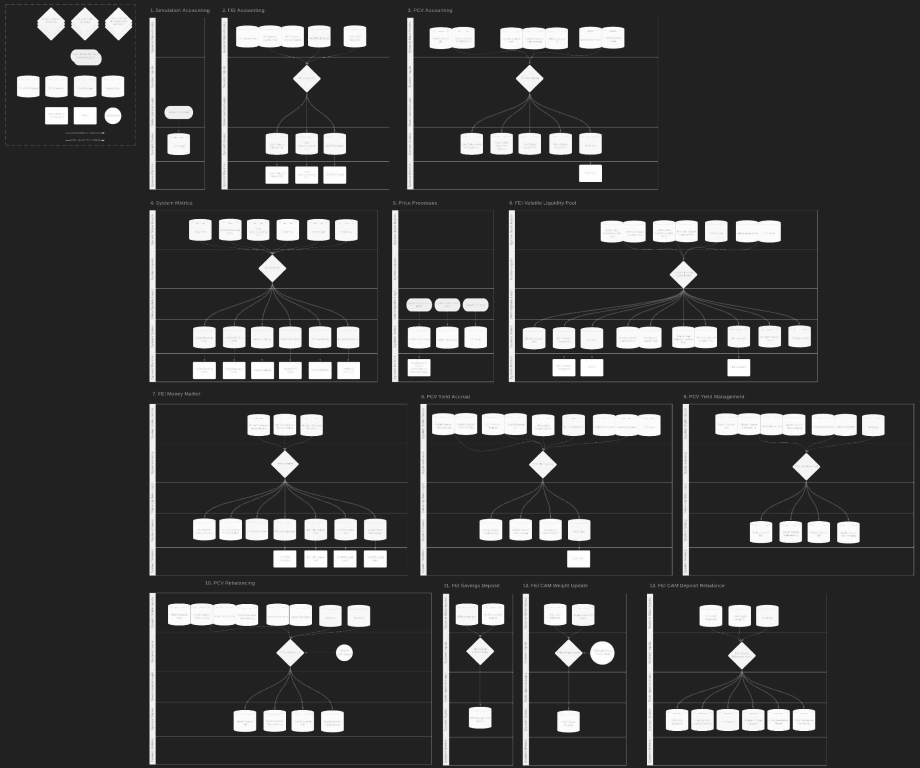
$$\frac{\mathrm{d}V}{\mathrm{d}t} = aV - bVP$$

$$\frac{\mathrm{d}P}{\mathrm{d}t} = -cP + dVP$$



A real-world model





Want to learn more?

- The demo code: https://github.com/BenSchZA/modelling-and-simulation-lightning-talk
- Complex systems engineering bootcamp: https://cadCAD.Education
- Modelling + simulation framework and examples: https://github.com/CADLabs/radCAD

Acknowledgements

- "The Ecosystem of Wicked Problems", Christian Sarkar and Philip Kotler
- SDGs: https://sdgs.un.org/goals
- Modelling and simulation diagrams: https://cadCAD.education
- Lotka-Volterra equations: https://en.wikipedia.org/wiki/Lotka%E2%80%93Volterra_equations