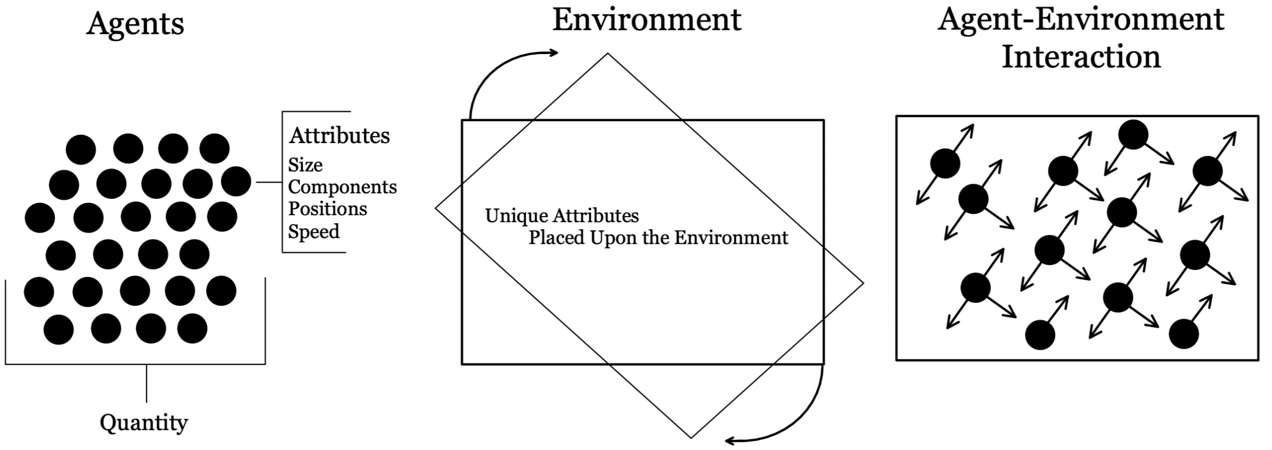
Agent-Based Modeling: A Practical Guide

Will Schenk

Background

There are three core components to all simulations: the agents, their environment, and the nuanced interactions between them. Agent-Based Modeling (ABM) is a computational approach to experimentation. Within an ABM, individual entities, termed "agents," are endowed with distinct characteristics and behaviors. As these agents navigate and interact within their environment, they give rise to emergent phenomena, providing observable insights.

ABM, given its wide applicability, is relevant to researchers, business analysts, and students. Successful creation of a model requires a development environment, proficiency in a programming language, and access to relevant datasets (if modeling real-world scenarios).



Benefits of ABM

- **Emergent Behavior:** Observe collective behavior arising from the individual actions of agents.
- **Flexibility:** Incorporate rules and behaviors on demand.
- **Scenario Testing:** Test various scenarios, rules, interventions to gauge potential impacts.
- **Simplicity:** Decompose a complex system into fundamental components to create a better understanding of interactions.

We will recreate the cellular automaton, "The Game of Life," to guide us in building a simulation. Mathematician John Conway created a 2 x 2 grid of cells, each of which can be alive (On) or dead (Off). By assigning simple rules for when a cell turns on or off, he could model real-world phenomena remarkably.

Warnings, Cautions, and Dangers

- **Validation Against Real-World Data:** Ensure your model aligns with real-world data to prevent deviations from reality, which could lead to false conclusions.
- **Over Tuning:** Caution against excessive fitting: Over-optimizing a model for a specific dataset can compromise its predictive ability.
- **Balance Complexity:** Oversimplifying may lead to unrealistic outcomes while over complicating may obscure insights. Find a balance.
- **Computational Intensity:** Running models with logical loops and many operations may overload the resources on your computer.

Let's create our simulation!

1 Fundamental Steps to Building an Agent-Based Model

1.1 Define Agents

In any simulation, agents are the entities that perform actions or undergo certain behaviors. They can represent anything: people, animals, cells, etc., depending on the context of the simulation.

a) **Determine Agent Attributes:** Attributes are characteristics that describe an agent. Depending on what you're modeling, agents can vary in complexity. *For instance, in our model, agents are cells with exactly one binary attribute → On or Off.*

b) **Initiate Agent States:** Think of states as the conditions or situations an agent can be in. Before a simulation starts, it's crucial to set an initial state for each agent. *In our example, we'll set cells that are active (or 'On') with an asterisk (∗), while setting inactive cells with a period (.), (or 'Off'). We will then place this information into a text file.*

game1.txt Hensel

```
**........**...**...**...**...**...**..
..**.**.....**...**...**...**...**...**
..**...**...**...**...**...**...**...**
**.....**.**...**...**...**...**...**..
.....**................................
```

Pseudocode:

```
Agent: Cell
Attribute: state (values: 'On' or 'Off')

Function DefineAgents():
    For each cell in our 100 x 100 grid:
        If cell is defined in 'game1.txt':
            Set cell state to 'On'
        Else:
            Set cell state to 'Off'
```

1.2 Define the Environment

The environment provides the context in which agents operate.

The environment is not a passive backdrop in an ABM; it dynamically **interacts with and shapes agent behaviors**.

a) **Visualization:** Through graphics or animations, visualization translates numerical results into discernible patterns. *In the Game of Life, the grid-based display represents the birth and death of cells.*

b) **Containment in a Defined Space:** Every ABM operates within a defined spatial boundary, guiding how agents navigate, interact, and exert influence. *The Game of Life is set on a grid, where the cellular patterns evolve based on their interactions.*

c) **Environmental Factors:** External factors like temperature or terrain can influence agent behaviors. For example, in a wildlife simulation, seasonal changes might dictate migration. *The Game of Life operates without these externalities. Patterns can still arise purely through agent rules and interactions.*

Pseudocode:

```
Environment: 2D grid (100x100 cells)

Function DefineEnvironment():
    Initialize a 100x100 grid
    For each position in the grid:
        Place the agent (cell) in that position
        (The cell can either be off (0) or on (1))
```

1.3 Define Agent-Environment Interaction

Dictate how agents will behave within the environment.

a) **Interaction Rules:** Specify how agents interact with each other and their environment. *In the Game of Life, this rule determines the state of each cell based on its neighbors.*

b) **Update Mechanism:** This decides when and how the agents' states get updated. *In the Game of Life, this happens for all cells simultaneously.*

Pseudocode:

```
Function AgentEnvironmentInteraction():
    For each timestep in the simulation:
        For each cell in the grid:
            Count number of neighboring cells that are 'On'
            If cell is 'On' and has 2 or 3 'On' neighbors:
                Keep cell 'On'
            Else if cell is 'Off' and has exactly 3 'On' neighbors:
                Turn cell 'On'
            Else:
                Turn cell 'Off'
        Update all cell states simultaneously
```

2 Example Code in MATLAB

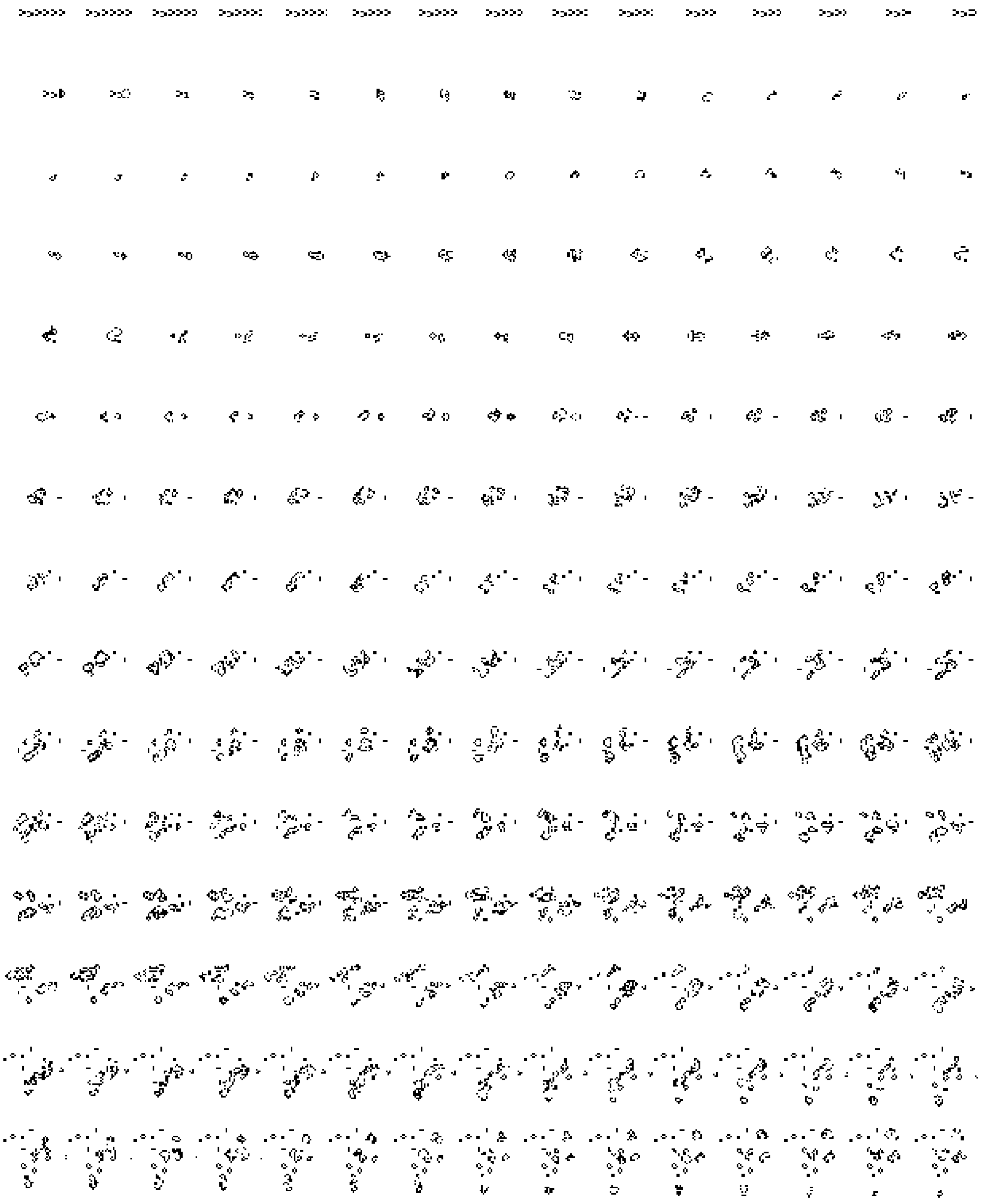
- MATLAB is a computer language and environment for technical computing.
- For the given code, set up initial conditions with dots and asterisk and save them in a separate file, *game1.txt*.
- The program will automatically read this *game1.txt* if it is in the same folder.
- You can then run John Conway's Game of Life cellular automaton model!

Seibold

```matlab
% Implementation of the popular cellular automaton model: Game of Life by John Conway.
% A cell remains alive if it has 2 or 3 live neighbors, and a dead cell becomes alive
% if it has exactly 3 neighbors. Otherwise, cells die or remain dead.
% Initial conditions are read from a data file.
% To run this example, the following file is needed:
% - game1.txt
% Original adapted from Benjamin Seibold

% Edited and Simplified by Will Schenk

% To run the program, you just need to call the function:
game_of_life()

function game_of_life()
    %% Define the Agents

    % Parameters
    n = [100,100]; % Number of cells per dimension.

    filename = 'game1.txt';
    pos = [40,40]; % Position of object loaded from file (top corner).

    %% Define the Environment

    % Construct initial configuration.
    F = zeros(n); % Initialize empty array.
    S = load_data_file(filename); % Load data file.
    % Add object to 2D array.
    F(pos(1)+(1:size(S,1)), pos(2)+(1:size(S,2))) = S;

    % Initialization.
    shl1 = [n(1),1:n(1)-1]; shr1 = [2:n(1),1]; % Shift index vectors in dimension 1.
    shl2 = [n(2),1:n(2)-1]; shr2 = [2:n(2),1]; % Shift index vectors in dimension 2.

    %% Define the Agent-Environment Interaction

    clf
    for j = 0:3000 % Time loop.

        % Plotting.
        imagesc(~F') % Plot array of cells.
        axis equal tight; colormap(gray)
        title(sprintf('Game of life after %d steps',j))
        pause(0.5) % Wait a bit (and a bit more initially).

        % Update rule.
        neighbors = F(shl1,shl2) + F(shl1,:) + F(shl1,shr2) + ...
            F(:,shl2) + F(:,shr2) + ...
            F(shr1,shl2) + F(shr1,:) + F(shr1,shr2);

        F = (F & (neighbors == 2 | neighbors == 3)) | ...
            (~F & neighbors == 3); % Update cells based on neighbors.
    end

end

function S = load_data_file(filename)
    % Read the data file "filename" and assign the information to 2D array "S".

    fid = fopen(filename,'r'); % Open data file for reading.
    data = double(fscanf(fid,'%c',inf)); % Read data as single string.
    fclose(fid); % Close file.

    data = [10, data, 10]; % Add space at the beginning and end of string.
    ind_data = data == 42 | data == 46; % Indices where actual cell data is stored.
    i_start = find(diff(ind_data) == 1); % Beginning index of row.
    i_end = find(diff(ind_data) == -1); % End index of row.
    row_length = i_end(1) - i_start(1); % Length of each row in data.

    data = data(ind_data); % Remove non-cell information.
    data = data == 42; % Convert data to logical for cell states.
    S = reshape(data, row_length, []); % Store object as 2D array.
end
```

Output on the next page!

Game of Life Collage: 200 Steps
Using Our Input
Read Left to Right, Top to Bottom



Works Cited

Hensel, Alan. “A Brief Illustrated Glossary of Terms in Conway’s Game of Life”, 1995, Technical revisions suggested by Achim Flammenkamp and Al Hensel were added on 7-Nov-95., www.radicaleye.com/lifepage/glossary.html.

Seibold, Benjamin. “Game of Life”, 2016, Professor of Mathematics and Physics, github.com/rujekoc/MATH2121/blob/main/gof_cellular_crowds/temple_abm_cellular_game_of_life.m.