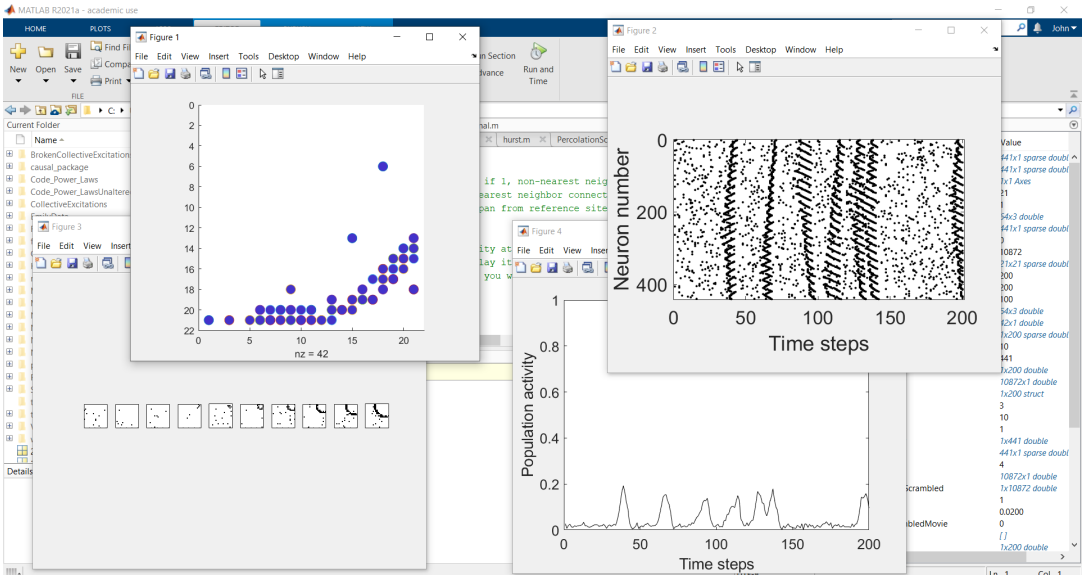
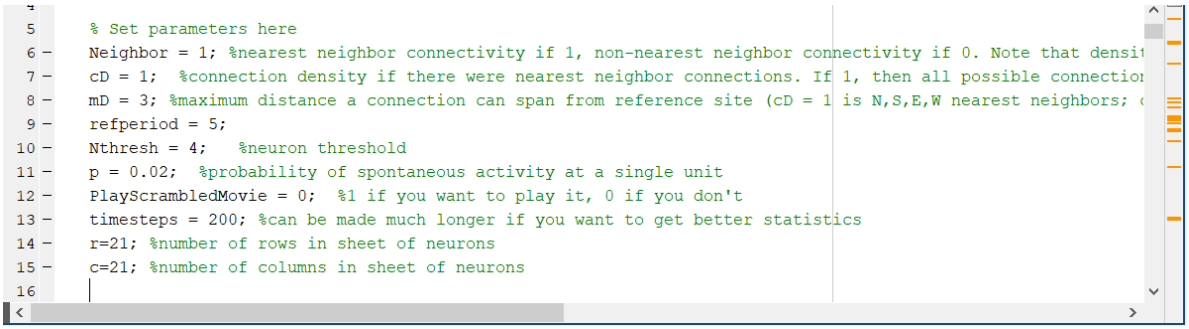
Exercises for Chapter 2, Emergent Phenomena

1. **Short overview**
   The impressive computational power of brains comes not primarily from isolated neurons, but from the emergent, collective interactions that they support. How do basic neuronal parameters affect simple collective interactions? The first Matlab script, NetworkModelCellularAutomaton, is designed to give an intuitive introduction to this area of research.

   The model is a simple cellular automaton, first introduced on pages 35 and 36 of the book. Here, each neuron is a threshold device and that can be either on (blue dot) or off (no color). The network is driven by spontaneous activity and can be given different types of connectivity. The output of the model will be (1) a brief video of activity, as if seen from a 21 x 21 microelectrode array (the movie will be replayed at a rate chosen by fps, the frames per second), (2) a raster view of spikes plotted against time, (3) a series of still frames taken from somewhere in the middle of the run, possibly showing an emergent pattern, (4) a plot of population activity (firing rate) against time. An example of these plots is given in the screenshot below. By changing the parameters of the network and seeing the output, you should quickly get an idea of how the dynamics are affected. As there are many parameter combinations, there is a large state space you can explore.



2. **The parameters**
   The parameters occur in the first few lines of the program and can be seen in the screenshot below.



```
% Set parameters here
Neighbor = 1; %nearest neighbor connectivity if 1, non-nearest neighbor connectivity if 0. Note that densit
cD = 1;  %connection density if there were nearest neighbor connections. If 1, then all possible connectior
mD = 3; %maximum distance a connection can span from reference site (cD = 1 is N,S,E,W nearest neighbors; c
refperiod = 5;
Nthresh = 4;   %neuron threshold
p = 0.02;  %probability of spontaneous activity at a single unit
PlayScrambledMovie = 0;  %1 if you want to play it, 0 if you don't
timesteps = 200; %can be made much longer if you want to get better statistics
r=21; %number of rows in sheet of neurons
c=21; %number of columns in sheet of neurons
```

Neighbor (nearest neighbor coupling)– when this parameter has a one and not a zero, the neurons will be situated on a square lattice and a neuron will be connected only to other neurons that are within a fixed radius. That radius is given by *mD* (maximum distance), explained below. The density of these connections is controlled by *cD* (connection density), explained below.

cD (connection density)– this parameter can vary from 0 to 1. In the case of *Random coupling*, this gives the fraction of all possible connections that will be actually used in the network. If the number is 1, then the network will have perfect all-to-all connectivity. If the number is 0.05, the network will have 5% of all possible connections. In the case of *Neighbor coupling*, this number will set the fraction of possible neighbor connections on the lattice. For example, if there are to be four neighboring neurons connected, but the *Connection density* is 0.75, then on average only three of these connections will be realized per neuron.

mD (maximum distance) – this only applies when *Neighbor coupling* is used. This sets the radius, in lattice units, from each neuron where permissible connections will be made. For example, if the maximum distance is 1.0, then neurons to the North, South, East and West of the given neuron will be connected, as in the Ising model. If the *Maximum distance* is 1.42 (greater than $\sqrt{2}$), then the diagonal connections will be allowed also, so that the neuron will now be connected to eight of its neighbors.

refperiod (refractory period)– this is the number of time steps the neuron must be silent after firing. It should have integer values only.

Nthresh (neuron threshold) – this is the number of active inputs required to make a neuron fire.

p (probability of spontaneous activation) – this is the probability that a neuron will become spontaneously active in a given time step.

PlayScrambledMovie – If you want to see what the activity would look like if the nearest neighbors were scrambled, change this parameter from 0 to 1. It will plot results for an unscrambled network first, followed by those for a scrambled network.

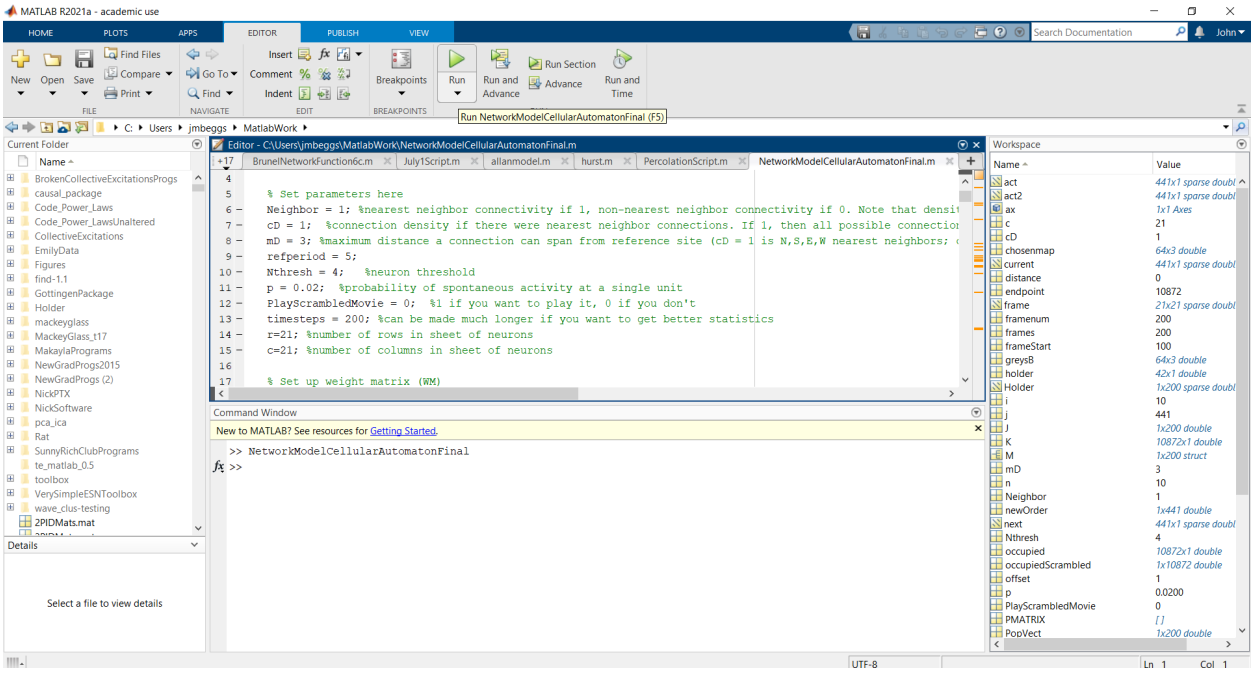
timesteps – this is just the number of timesteps the simulation will run. 200 is a good number to start and will allow you to quickly glimpse activity types. Longer runs are possible for more careful investigation.

r (rows in the sheet of neurons), *c* (columns in the sheet of neurons) – these parameters set the dimensions of the sheet of neurons in the simulation.

fps – this is the number of frames per second in the movie when it is replayed. An fps = 7 is a good pick to allow you to see events unfold.

3. Different types of emergent phenomena

To run the simulation, first download the programs and place them in your Matlab path. Once you have opened the program in the editor, just click on the Run button as seen in the screenshot below.



To describe the parameters for other settings, we will use a vector: (*Neighbor, cD, mD, refperiod, Nthresh, p*). For these exercises, let us assume *r* = *c* = 21, so we have a 21 × 21 grid of neurons.

Here are some example activity patterns to get started:
Square waves: (1, 1, 1.5, 5, 1, 0.001)
Circular waves: (1, 0.95, 2.5, 2, 3, 0.01)
Repeating patterns: (0, 0.1, any, 2, 5, 0.02)
Complementary on-off patterns: (1, 0.7, 3, 1, 1, 0.01)

4. Exercises

A. Set these parameters: (1, 1, 1.5, 7, 2, 0.015). Can you identify particle-like structures? What signatures do they leave on the raster? What happens when these structures collide? What parameter changes disrupt these particle-like structures?

B. Set these parameters: (1, X, 2.5, 5, 1.5, 0.02). Start the connection density, X, at 0.2, and then increase it by increments of 0.05 until you reach 0.7. What is going on here? Why do you think this is happening? If there is a transition point, where would it be?

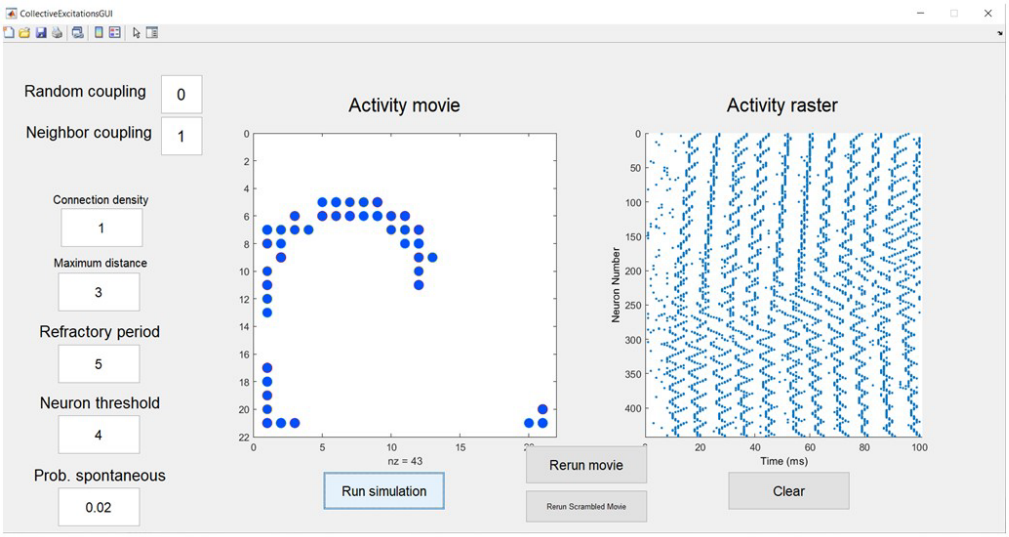
C. Set these parameters: (0, 0.25, X ≥ 6, 4, 7, 0.02). How would you describe the frequency of this activity? What parameter(s) most affects the frequency?

D. Set these parameters: (0, 0.1, 4, 2, 5, 0.02). In what sense could this network be described as having an "attractor?" What might this attractor be useful for?

E. Set these parameters: (1, 1, 3, 5, 4, 0.02). Watch the movie and inspect the raster. What collective excitation did you see? Now set *PlayScrambledMovie* = 1 and watch the movie in scrambled form and examine the raster. How is the pattern you observe now different? How might this be similar to our observations of neuronal activity, where connected neurons are not necessarily nearest neighbors? Under sparse random connectivity, how could you observe collective excitations in the brain if they were actually there?

F. What is the primary difference between random and neighbor activity, in terms of the collective excitations that can be set up? As an example, set these parameters: (1, 1, 1.5, 7, 2, 0.015). Now switch it to random coupling by with an approximately equivalent density (0, 0.8, 1.5, 7, 2, 0.015). What do you see? Which situation is more like that found in the neocortex?

   Included in this chapter are two programs named CollectiveExcitationsGUI.m and CollectiveExcitationsGUI.fig (one is the computer code to simulate the cellular automaton and the other runs the figure). They are basically the same as NetworkModelCellularAutomaton, except they have a graphical user interface. This means they have buttons and boxes for you to type variables into. See the figure below. To use this interface, just type CollectiveExcitationsGUI into the Matlab command line.



When you type the line CollectiveExcitationsGUI into the Matlab command window, the interactive box shown above will appear. This allows you to input variables by typing into the boxes. You can run it by clicking on the Run simulation button.

G. Based on these exercises, how would you define an "emergent phenomenon?" How is an emergent phenomenon different from random activity in a network? What are the defining features of an emergent phenomenon? Is a threshold necessary for emergent phenomena?

H. The book mentions Conway's "Game of Life" on pages 43 – 45 as a cellular automaton rule that has been extensively studied because it produces diverse emergent phenomena (see here for more: https://conwaylife.com/wiki/Conway%27s_Game_of_Life). The script ConwaysGOL allows you to simulate Conway's Game of Life in a manner similar to the script above, generating the same type of plots. The script runs on a larger, 41 × 41 grid. Note that you can control the initial conditions to create different "creatures" from the menagerie (see lines 43 – 52). Try these different initial conditions to get a sense of what can happen.

I. Note that Conway's rule for the Game of Life is only one of 2^{18} possible rules for a two-dimensional cellular automaton with a neighborhood of eight nearest neighbors that can be either on or off. To see what would happen if a rule were picked at random, run ModifiedLifeFunction, as shown here:

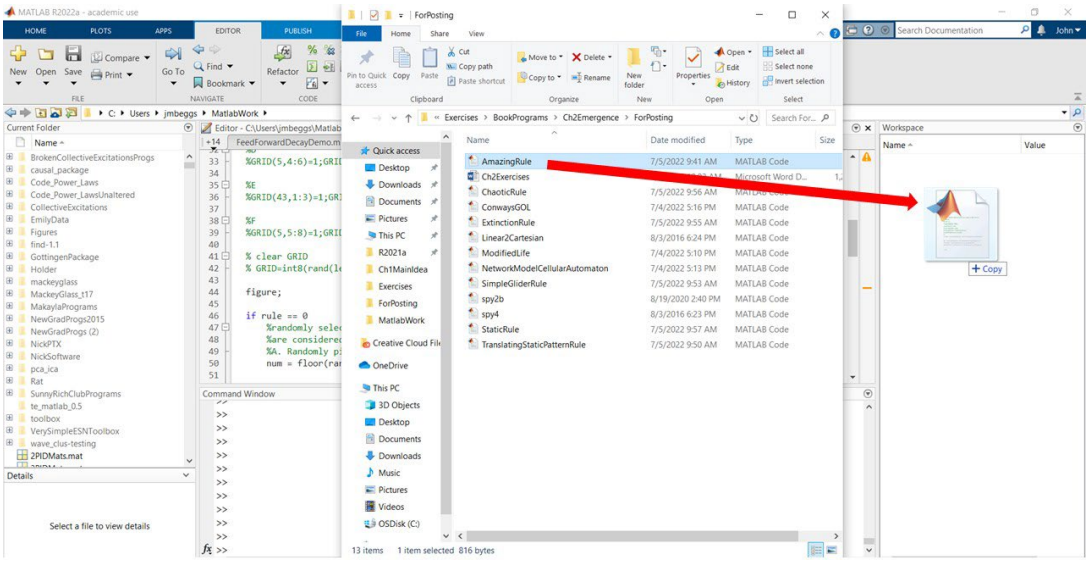
```
[RuleUsed] = ModifiedLifeFunction(len, frames, rule);

[RuleUsed] = ModifiedLifeFunction(100, 100, 0);
```

Where "len" is the edge length of the array in which it will be run, "frames" is the number of steps in the simulation (and frames in the movie it produces), and "rule" is the type of rule you want to run. If rule = 0, it will randomly select

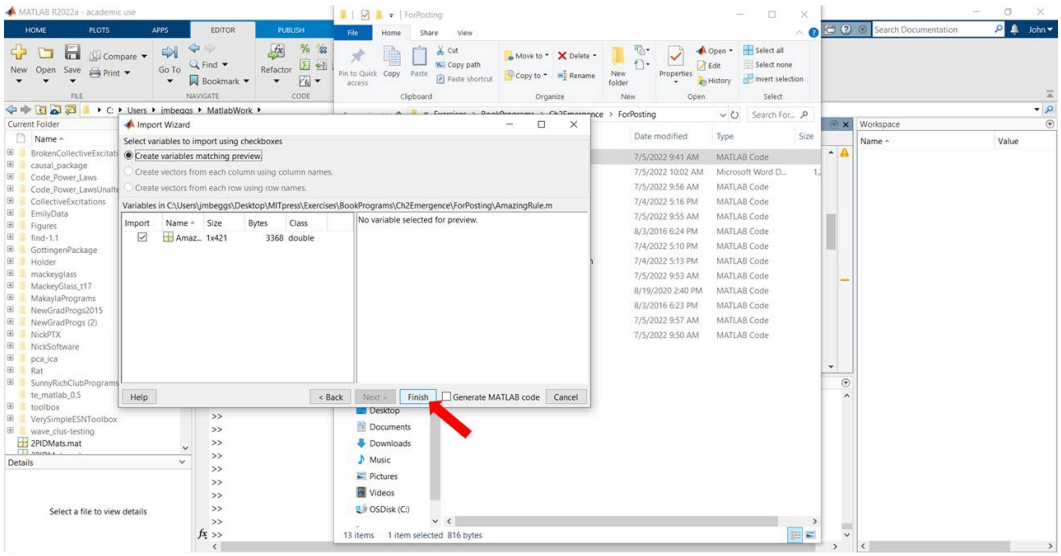
one of the possible rules and then quickly run a movie from a simple initial condition. If you don't want to just randomly pick a rule, you can load one of the saved rules, as described below.

Exercise: Run this script 100 times (it is quick) with rule = 0 and try to classify the different outputs that you see. What fraction of them appear to be "random?" What fraction of them appear to have some basic structure? What fraction of them produce something like what you see in the Game of Life (i.e., glider like structures that do not expand or contract)? Can you describe mechanistically what leads to blank patterns with no activity? What leads to random patterns? What leads to interesting patterns?

Below are some examples of saved rules to give you an idea of the variety of activity patterns that we can see. To load one of these rules, just drag it (it should have a name that ends in "Rule") from the folder into the Matlab workspace, like this:



Then click "Finish" in the box that pops up:

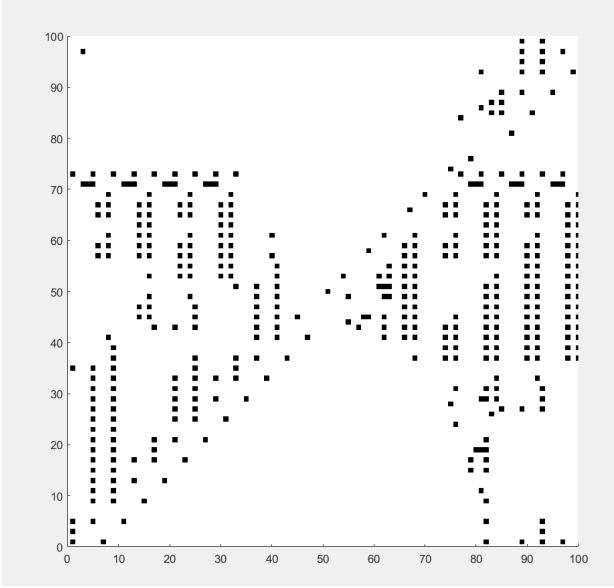


Finally, you can run the function by just using the name of the rule, like this:

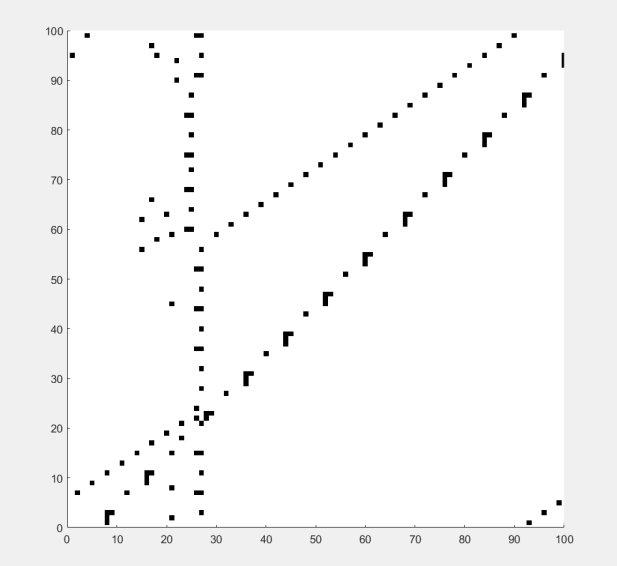
```
[RuleUsed] = ModifiedLifeFunction(100, 300, AmazingRule);
```

(Note that all the saved rules were generated with len = 100, and so will only work under that condition. However, the number of frames is not restricted).

AmazingRule, when loaded and run, produces the very complex pattern shown below. It moves diagonally to the upper right and its components are also growing, then resetting. This is an example of a rule that leads to complex activity.



TranslatingStaticPatternRule, shown below, produces the static pattern and glides to the upper right.



See also these rules: SimpleGliderRule, ExtinctionRule, ChaoticRule, StaticRule as examples of the types of activity the program can produce.

If you find that a randomly selected rule produces interesting output and you want to save it, just take the output variable "RuleUsed" and assign it to a rule name, and then save that rule, like this:

```
[RuleUsed] = ModifiedLifeFunction(100, 300, 0);
NeatRule = RuleUsed;
save NeatRule -mat
```

Now you can go back and use "NeatRule" whenever you want.

5. Project ideas (more difficult)

J. Adapt the code that plots population activity over time (from NetworkModelCellularAutomaton) to characterize how each rule responds to a minor perturbation in inputs. You can proceed along these lines: First, create a random input configuration of active cells and then create a copy of it, with just *N* cells' status changed (e.g., if the cells were on, turn them off and vice versa). Second, run the cellular automaton rule from both starting conditions. Store every frame of the temporal activity for both runs. Third, measure the "distance" between these runs by counting the number of cells that differ over time, starting with *N* cells differing at the start. Note that this distance could do one of three things: it could stay roughly the same, it could grow over time, it could shrink over time. Is there any relationship between the type of activity the rule produces (ordered, complex, disordered) and the way that the distance between outputs changes over time? For example, do

ordered rules tend to produce decreasing distance over time? Note: you may have to average over many input configurations before seeing a trend. See the appendix of the book for a description of the Lyapunov exponent.

K. There is a 3-dimensional version of the Game of Life (try here: Leandro Barajas (2022). Conway's Game of Life in 3D (https://www.mathworks.com/matlabcentral/fileexchange/4892-conway-s-game-of-life-in-3d), MATLAB Central File Exchange. Retrieved July 5, 2022.) Modify this open source code so that it can run rules other than those of the Game Of Life, just as was done in two dimensions with the program ModifiedLifeFunction. Using your new program, survey about 100 randomly chosen rules. Again, try to classify the different outputs that you see. What fraction of them appear to be “random?” What fraction of them appear to have some basic structure? What fraction of them produce something like what you see in the 3D Game of Life? Can you describe mechanistically what leads to blank patterns with no activity? What leads to random patterns? What leads to interesting patterns?

L. Chapter 2 in the book mentioned the topic of “downward causation,” noting that there is not a consensus about it in the scientific or philosophical communities. If you were to try to demonstrate that such a phenomenon exists, how would you do it? Could you devise a computational experiment, using programs like the ones mentioned here?

Matlab code useful for exercises in this chapter, listed in order of use:

NetworkModelCellularAutomaton
ConwaysGOL
ModifiedLifeFunction
spy2b
spy4
Linear2Cartesian
(The two programs below are graphical user interface, GUI, versions of the first program NetworkCellularAutomaton. To use these, just type CollectiveExcitationsGUI into the Matlab command window):
CollectiveExcitationsGUI.fig
CollectiveExcitationsGUI.m

Cellular automaton rules:
AmazingRule
ChaoticRule
ExtinctionRule

NeatRule
SimpleGliderRule
StaticRule
TranslatingStaticPatternRule