CS166 Problem Set 1

Instructions

- Show all your writing, math, code, and results in a Python/Jupyter notebook. Upload the .ipynb (or .zip) file of the notebook.
- Your instructor has to be able to run your code. Do not simply upload a PDF version of your Python notebook.
- Complete all 4 problems below. Some of the problems have optional challenge questions which you may skip.
- You will be graded on the LOs and HCs specified in each problem.
- If you complete an optional challenge very well,[1] you will score a ⑤ on the LO or HC specified in the problem.

Problem 1: Design a model

- LOs: #Modeling
- HCs: #emergentproperties
- Words: 300–600

(Adapted from Exercise 2.4 in H. Sayama, *Introduction to the Modeling and Analysis of Complex Systems*, p. 299.)

Write a few paragraphs (no math required) describing a real-world system of your choice that is made up of many interacting components/agents.

- At which scale do you choose to describe the microscopic components and what are those components? Briefly describe other scales (larger and/or smaller) at which you could have chosen to model the microscopic state and why your chosen scale is the most appropriate.
- What states can the microscopic components take on?
- What are the relationships between the components?
- How do the states of the components change over time? Some of these changes may depend on interactions between the components while other changes may not.

After answering all of the questions above, predict what kind of macroscopic behaviors would arise if you run a computational simulation of your model.

Problem 2: Is the `random` module random?

- LOs: #EmpiricalAnalysis
- HCs: #confidenceintervals (for the optional challenge question)
- Words: 100–200

We check if NumPy's `random` module generates values that appear to be random.[2] If the model is random, we should get integers selected from a range at random approximately equally often. Use the code below to generate an array of 1,000,000 random integers from 0 to 99.

```
import numpy as np
numbers = np.random.randint(100, size=1000000)
```

Then complete the following tasks.

1. Make a properly formatted histogram of the number of times each value appears.
2. Discuss in detail how often you expect each value to appear, how often each value actually appeared, and whether the distribution over the results matches what you would expect according to the Central Limit Theorem.
3. Optional challenge: Derive (analytically) a 95% confidence interval of the mean count of 1,000,000 randomly selected integers from 0 to 99 that equal a particular value. Explain how many of your 100 histogram values should fall within this confidence interval. Count how many of your 100 histogram values do actually fall within this confidence interval. Compare and comment on the results.

Problem 3: Forest fire mean-field approximation

- LOs: #TheoreticalAnalysis
- Words: no specific word count; mostly math

Consider the following forest fire model described in Drossel, B., & Schwabl, F. (1994). Formation of space-time structure in a forest-fire model. Each site of a 2-dimensional grid is occupied by a tree, a fire (a burning tree), or it is empty. During each discrete update, the following rules are applied.

- fire → empty, with probability 1.
- tree → fire, with probability $1-g$ if at least one neighbor in its Von Neumann neighborhood is a fire. (Think of g as the immunity of a tree to catching fire.)
- tree → fire, with probability $(1-g)f$ if no neighbor is burning. (Think of f as the probability of a lightning strike or some other cause of a spontaneous fire – a small probability quite close to 0.)
- empty → tree, with probability p.
- Otherwise, the state of the cell remains the same.

Prove the equations in the paper in (2.1) for the change of the mean-field densities during one timestep.

$$\Delta\rho_e = \rho_f - p\rho_e$$

$$\Delta\rho_t = p\rho_e - \rho_t(1-g)\Big(f + (1-f)(1-(1-\rho_f)^{2d})\Big)$$

$$\Delta\rho_f = -\rho_f + \rho_t(1-g)\Big(f + (1-f)(1-(1-\rho_f)^{2d})\Big)$$

Note that $d = 2$ is the dimension of the grid.

Problem 4: Wireworld

- LOs: #PythonImplementation
- HCs: #algorithms (for the optional challenge question)
- Words: no specific word count; mostly code

The Wireworld model is a cellular automaton that can be used to simulate simple electrical circuits. Even though these circuits are simple, the Wireworld CA is Turing-complete. This means you can use it to simulate a general-purpose computer like your laptop – assuming you have enough storage, computation, and patience to run the CA.

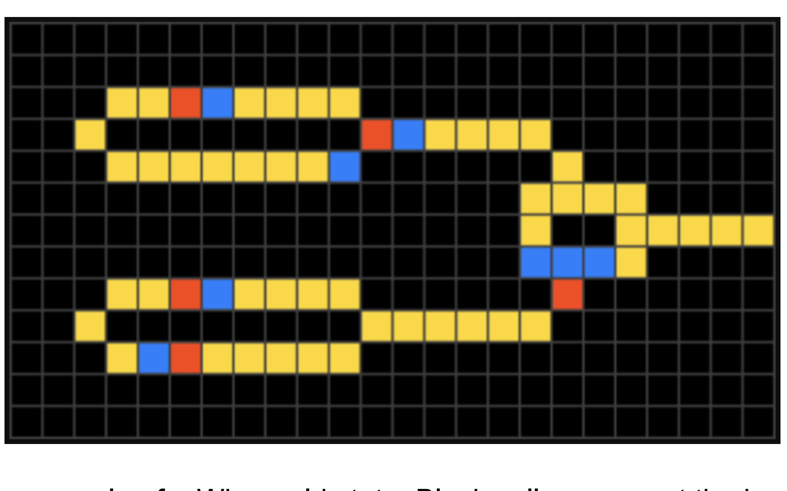

Figure 1. An example of a Wireworld state. Black cells represent the background, yellow cells represent wires, and blue and red cells represent electrons (head and tail, respectively). See the Wikipedia source image for an animation showing how electrons move around the circuit.

Each cell is in 1 of 4 states–

0. background
1. electron head
2. electron tail
3. wire

In each discrete time step, the update rules are as follows.

- *Background* remains *background*.
- *Electron head* changes to *electron tail*.
- *Electron tail* changes to *wire*.
- *Wire* changes to *electron head* if and only if one or two of its neighbors are *electron head*. Otherwise, it remains *wire*. Use a Moore neighborhood to count the number of neighbors.

Implement this simulation in Python and use your implementation to reproduce the animation in Figure 1 above. Include the animation in your Python notebook.

Optional challenge: The Wireworld simulation is Turing-complete because we can use it to construct logic gates. See this MathWorld article for visualizations of the OR, XOR, and AND logic gates. The challenge is to create a circuit that adds two 4-bit binary numbers. To do this, you will need 1 half adder and 3 full adders.

1. Explain which logic gates you can use to implement a half adder and a full adder.
2. Explain how to add two 4-bit binary numbers using the half and full adders.
3. Build a Wireworld simulation that takes two 4-bit binary numbers as input and produces their sum as output. Generate a few animations to show that your 4-bit binary number adder works.
4. Even more optional super challenge: Extend your Wireworld simulation to show the result of the 4-bit binary number adder as a hexadecimal digit using a 7-segment display. The Wireworld Wikipedia article shows how to visualize a decimal digit using a 7-segment display. You need to extend this cellular automaton to visualize a hexadecimal digit. Generate a few animations to demonstrate how your adder and your display work together.

   If you need inspiration, here is a video of a Wireworld CA that displays prime numbers. Anything is possible!

   An extra ⑤ on #PythonImplementation if you pull this off.

[1] "very well" means answering the whole optional question with no errors or bugs and, it needs to be good enough that it can be published on something like Medium (perhaps with a few minor tweaks). See the CS166 Grading Policy for details.
[2] We know these values are not truly random since every pseudo-random number generator is deterministic but, we can check that the pseudo-random numbers have the same statistical properties as true random numbers.