Lesson Plan

1. Implementing CAs (~1 hour)
2. Discussion (~15 minutes)
3. Open-ended coding (~15 minutes)

Implementing Cellular Automata

- Arrays/2D arrays for grids
    - [], literals, .push(), indexing, maybe slicing
    - Looping over 2d arrays
- Design Recipe
    - Intention:
        - Initialize with a random grid.
        - On click, run an iteration (up to N iters defined as const) of the cellular automata defined in the paper.
        - After N iterations, draw walls.
        - Further clicks will restart the process with a new random grid.
    - State:
        - 2D grid of cells
        - Constants - parameters from paper (T1, T2, N_ITERS)
    - Library functions
        - Stroke, fill, rect
        - Array indexing, push, iteration
    - Stuff we need to implement:
        - Randomly initializing the grid
        - Rendering the grid
        - Updating a single cell
        - Updating all cells
        - Detecting condition to stop iterating and render walls
        - Detecting condition to restart
- Coding
    - Starter file: https://editor.p5js.org/chrisamaphone/sketches/_OUwSqhRD
    - Together:
        - Render
    - On your own:
        - Initialize random
    - Together
        - Update cell
    - On your own
        - Update all cells
    - Together
        - Walls

Discussion questions

- (dis)advantages to CAs for PCG
    - Disadvantage: poor control
        - Predictability?
        - Hard to make sure it appears how you want
        - **Controllability**
- Relationship between grammars and CAs

On your own

- Implement your own CA
    - Try in p5.js: modify this example, e.g., to create water puddles, treasure, or stalactites/stalacmites
    - Try in p5.js: a new example. For example, you could try:
        - The 1D “car” automaton from last class
        - A “sandpile” automata
        - Design your own cave generation algorithm
    - Try any of the above in Emoji Simulator: https://ncase.me/sim/

Code written in class:
https://editor.p5js.org/c.martens/sketches/6WyuIiwAH