A close-up of a sphere

Description automatically generated with low confidence

Implement this simulation on a torus (donut shape) – do not need to deal with boundaries. This means that cells on the top are adjacent to cells on the bottom and the same is true for the left and right sides.

Write a function conway(s, p) that generates a board, which is a square two-dimensional NumPy array of size s. To simplify things, assume the board is a square, and s is length of the side. The board should be randomly populated with probability p. \If p is set to 0, all cells should be 0 (dead). If p is set to 1, all cells should be 1 (alive). Start with p=0.1; about 10 percent of cells should be 1.

Write a function advance(b,t) that accepts a Conway board and advances it t time steps according to the rules:

* Any live cell (marked as 1) with fewer than two live neighbors dies, as if by underpopulation.
* Any live cell (marked as 1) with two or three live neighbors lives on to the next generation.
* Any live cell (marked as 1) with more than three live neighbors dies, as if by overpopulation.
* Any dead cell (marked as 0) with exactly three live neighbors becomes a live cell, as if by
* reproduction.

Write a function to pleasantly display the board.

1. When you "flip" a square in the middle of an iteration, don't let the flipped state impact its neighbors during that iteration. The neighbors should be impacted by the original state for that iteration. To ensure that, you must use two boards – the current one you are traversing over, and the new one which you store the new states of the cells.

2. Write a function simulate\_gameoflife(s, p, n). This function is the top-level call to do the simulation. Create a board using s and p and do a maximum of n iterations. The simulation must terminate when it reaches the specified maximum number of iterations, or if the board configuration didn’t change any cell at all at the end of an iteration (to prevent an infinite loop). In the function, display the (current) board at the end of each iteration.

3. Show the result of a simulation with the maximum 10 iterations. Display using the board of 10 x 10.